

doc. Ing. Pavel Šenovský, Ph.D.

Bezpečnostní informatika 3

skripta
5. vydání



Bezpečnostní informatika 3

5. rozšířené vydání

tento text neprošel jazykovou úpravou

©Pavel Šenovský, Ostrava, 2025

Vysoká škola báňská - Technická univerzita Ostrava, Fakulta bezpečnostního inženýrství

Obsah

Seznam obrázků	7
Seznam tabulek	9
Seznam výpisů kódů	12
Úvod	13
1 Základy průmyslové automatizace	17
1.1 Automatizace	17
1.2 PLC	18
1.3 SCADA	20
1.4 Další zařízení využívána v automatizaci	24
1.5 Průmysl 4.0	26
2 Real-time databáze	29
2.1 Real-time databáze vs relační databáze	29
2.2 Transakce	32
2.2.1 Organizace dat v databázi	32
2.2.2 Jazyk SQL	33
2.2.3 Transakce	37
3 SCADA - příklad v Promotic	41
3.1 Promotic - instalace a projekt	41
3.2 Vizualizace UPS - rozbor funkcionality	43
3.3 Implementace projektu v Promotic	44
3.3.1 Průvodce vytvoření projektu	45
3.3.2 Nastavení projektu a příprava proměnných	46
3.3.3 Návrh obrazu aplikace	52
3.3.4 Oživení aplikace	55
4 Automatizace - otázky bezpečnosti	59
4.1 Případové studie bezpečnosti automatizace	60
4.2 Způsoby izolace automatizační sítě	65
5 Ochrana ICS	75
5.1 Typy CERT a CSIRT	75
5.2 Legislativa kybernetické bezpečnosti	79
5.2.1 Směrnice NIS2 a Zákon o kybernetické bezpečnosti	80
5.2.2 Ostatní legislativní souvislosti	84
6 Data mining	87
6.1 Postup data miningu	88
6.2 Metodiky dolování dat	93
6.3 Zdroje dataminingu	96

7 Úvod do R	101
7.1 Instalace prostředí	101
7.2 RStudio	102
7.3 Základy jazyka R	105
7.4 Knihovna dplyr	112
7.5 Grafy pomocí knihovny Plotly	118
8 Pravidlové metody	123
8.1 Příprava dat	124
8.2 Rozhodovací stromy	126
8.3 Případová studie rozhodovacího stromu - AMES dataset	133
8.4 Rozhodovací pravidla	139
8.5 Asociační pravidla	141
9 Bayesovská klasifikace	149
9.1 Úvod do Bayesovské klasifikace	149
9.2 Případová studie	153
10 Neuronové sítě	157
10.1 Neuron	158
10.2 Neuronové sítě a jejich adaptace	160
10.3 Případová studie	166
11 Moderní umělé inteligence	175
11.1 Umělá inteligence	175
11.2 Robotika a AI	181
11.3 Budoucnost AI	182
Příloha 1 - UPS timer ve VBS	185
Literatura	190
Seznam zkratk	193
Rejstřík	194

Seznam obrázků

Základy průmyslové automatizace	17
1.1 Obecné schéma regulační soustavy	18
1.2 Příklad PLC a řídicího systému (převzato z [10])	19
1.3 Zapojení SCADA systému, databáze, PLC a řízené technologie	21
1.4 Dohledové pracoviště Národní dopravní a informační centrum (NDIC) (převzato z [45])	22
1.5 Řešení monitoringu fotovoltaické elektrárny (převzato z [28])	23
1.6 Možná struktura webové aplikace zajišťující rozhraní člověk - stroj	25
Real-timeové databáze	29
2.1 Databázový klastr	31
2.2 ERD diagram metadat příspěvků v síti Twitter (adaptováno z [81])	32
2.3 Výsledky spojení dvou tabulek	35
2.4 Změna hodnoty transakce v čase	38
SCADA - příklad v Promotic	41
3.1 Úvodní okno Promotic	42
3.2 PmManager - správa projektů	43
3.3 PmManager - správa runtime modulů	43
3.4 UPS FINGERS FR-1250LCD [1]	44
3.5 Nová aplikace Promotic	45
3.6 Nastavení základních vlastností aplikace Promotic	46
3.7 Volba rozložení aplikace Promotic	47
3.8 Volba vzhledu hlavního panelu aplikace Promotic	47
3.9 Promotic - základní rozhraní projektu	48
3.10 Promotic - nový objekt	49
3.11 Promotic - PmNumber - objekt typu číslo	50
3.12 Promotic - PmString - objekt typu textový řetězec	50
3.13 Promotic - PmData - sdružující proměnné různého typu	51
3.14 Promotic - Vytvoření proměnné celočíselného typu v PmData	52
3.15 Promotic - paleta předkonfigurovaných objektů pro použití v obrazech	53
3.16 Promotic - nastavení sloupce pro zobrazení % nabití baterie	53
3.17 Promotic - nastavení měřidla typu kruhový měřič	54
3.18 Promotic - nastavení táhla pro dolní mez napětí	55
3.19 Promotic - celková vizualizace UPS	55
Automatizace - otázky bezpečnosti	59
4.1 Bezpečnost řešená na vnějším perimetru sítě firewallem	66
4.2 Dual home computers	67
4.3 Dual home server	68
4.4 Oddělení sítí pomocí zařízení různých typů	70
4.5 Důvěryhodná vs demilitarizovaná zóna	71
4.6 Důvěryhodná vs demilitarizovaná zóna - řešení s použitím dvojice firewallů	73

Data mining	87
6.1 Technický pohled na postup data miningu (adaptováno z [20])	88
6.2 Manažerský pohled na postup data miningu (adaptováno z [20])	90
6.3 Zjednodušený postup dataminingu	91
6.4 Úlohy data miningu - pokrytí konceptu (adaptováno z [20])	92
6.5 Proces metodiky SEMMA (adaptováno z [20])	94
6.6 Proces data miningu v SAS Enterprise Miner (převzato z [71])	95
6.7 Životní cyklus data miningu (převzato z [40])	96
6.8 Příklad struktury zjednodušené objednávky	97
6.9 Struktura informačních systémů podniku	97
6.10 Datová krychle v OLAP (adaptováno z [20])	98
 Úvod do R	 101
7.1 Rozhraní RStudio v OS X	103
7.2 Podrobnosti o sloupcích tabulek, matic a dataframů v RStudio	104
7.3 Data Viewer - proměnná <code>tornado_data</code>	104
7.4 Párové srovnání vlastností kosatců v datové sadě iris	110
7.5 Délka vs šířka listu kosatců v datasetu iris (bodový graf)	111
7.6 Dostupné typy vykreslování ve funkci <code>plot</code>	112
7.7 Rozložení délky listu dataset iris (histogram)	113
7.8 Délka listu dataset iris (boxplot)	114
7.9 Rozložení počtu prodejů podle jejich úspěšnosti (pie)	115
7.10 Počet případů COVID-19 v jednotlivých dnech - 7-mi denní klouzavý průměr	118
7.11 Exponenciální nárůst nakažených COVID-19 ve vybraných státech - demonstrace logaritmických měřítek os	120
7.12 Vývoj denního počtu nově diagnostikovaných COVID-19 v ČR za poslední 2 týdny	121
 Pravidlové metody	 123
8.1 Filozofie použití trénovací a validační množiny	124
8.2 Klienti banky - ručně vytvořený rozhodovací strom	129
8.3 Klienti banky - strom vygenerovaný toolkitem <i>Tree</i>	132
8.4 Regresní strom pro AMES dataset	137
8.5 Vztah velikosti stromu a přesnosti výpočtu pro AMES dataset	138
8.6 Komplexita stromu a jeho přesnost pro AMES dataset měřeno pomocí RMSE	138
8.7 Vztah velikosti stromu a přesnosti výpočtu pro AMES dataset měřeno pomocí RMSE	139
8.8 Hodnocení významnosti parametrů funkcí <code>vip</code> pro AMES data - model rozhodovacího stromu	140
8.9 Asociační pravidla - dataset Titanic (27 základních pravidel)	146
8.10 Asociační pravidla - dataset Titanic (zaměřeno na přeživší podle věku a třídy)	146
 Bayesovská klasifikace	 149
9.1 Kosatec (iris) - grafický popis významu sloupců (převzato z [83])	153
 Neuronové sítě	 157
10.1 Schéma neuronu (převzato z [21])	158
10.2 Neuron vizuálního kortexu (převzato z [42])	159
10.3 Schématické znázornění modelu perceptronu	160
10.4 Graf sigmoidy	161
10.5 Jednoduchá neuronová síť pro OCR	162
10.6 Schéma vícevrstevné sítě	162
10.7 Neuronová síť pro predikci hodnot funkce $y = x^2$	163
10.8 Funkce $y = x^2$ - srovnání modelu neuronovou sítí a lineární regresí	164
10.9 Přeučení model	165
10.10 Klasifikace klientů banky neuronovou sítí	166

10.11RMSE pro různé modely neuronové sítě datasetu AMES	168
10.12Relativní významnost jednotlivých parametrů modelu neuronové sítě datasetu AMES	169
10.13Ladění hyperparametrů .size (3, 5, 10, 20) a decay (0,1 - 0,9) pro AMES dataset . . .	170
10.14Ladění hyperparametrů .size 3-9 pro standardizované prediktory AMES dataset	171
10.15Srovnání významnosti prediktorů pro nnet síť s normalizovanými prediktory po 1000 iterací (AMES dataset)	172
Moderní umělé inteligence	175
11.1 AI hype cyklus 2025 (převzato z Gartner [33])	176
11.2 Činnost chatovací aplikace konverzační AI	177
11.3 Implementace přemýšlení do LLM	178
11.4 Implementace AI agenta pomocí LLM	179

Seznam tabulek

Základy průmyslové automatizace	17
1.1 Srovnání SCADA a DCS	21
Úvod do R	101
7.1 Počet obyvatel v krajích v roce 2024 - široký formát (data ČSÚ [89])	116
7.2 Počet obyvatel v krajích v roce 2024 - dlouhý formát (data ČSÚ [89])	116
Pravidlové metody	123
8.1 Přidělování úvěrů banky (převzato z [20])	127
8.2 Příjem vs úvěr	127
8.3 Přidělování úvěrů - 2. větvení	128
8.4 Přidělování úvěrů - 3. větvení	129
8.5 Binární strom přidělování úvěrů klientům banky - confussion matice	133
8.6 Párek vs hořčice	142
8.7 Rozložení datasetu Titanic	143
Bayesovská klasifikace	149
9.1 Bayesovná klasifikace pro iris dataset - confussion matice	154
Neuronové sítě	157
10.1 Trénovací množina funkce $y = x^2$	162
10.2 Neuronová síť predikce vs realita ($y = x^2$)	167

Seznam výpisů kódů

Real-timové databáze	29
2.1 SQL: definice tabulek - příklad Twitter	34
2.2 SQL: struktura příkazu CREATE TABLE	34
2.3 SQL: struktura příkazu ALTER TABLE	34
2.4 SQL: výběrový dotaz - příklad výběru konkrétního příspěvku	35
2.5 SQL: výběrový dotaz - příklad výběru příspěvků určitého uživatele	35
2.6 SQL: výběrový dotaz - propojování tabulek vnitřním spojením (INNER JOIN)	35
2.7 SQL: příklad vkládání údajů do tabulek (INSERT INTO)	36
2.8 SQL: struktura příkazu INSERT INTO	36
2.9 SQL: příklad aktualizace záznamu v databázi (UPDATE)	36
2.10 SQL: příklad výmazu záznamů z databáze (DELETE)	36
SCADA - příklad v Promotic	41
3.1 JavaScript: oživení časovače projektu UPS v Promotic 9.0	55
Úvod do R	101
7.1 Příklad instalačního skriptu pro Bash instalujícího nástroje R a RStudio pro distribuce OS Linux na bázi Debian Linux	102
7.2 Základní operátory R a přiřazení výsledku do proměnné	105
7.3 Základní funkce v R	105
7.4 Manipulace s vektory v R	106
7.5 Spojování vektorů v R	106
7.6 Realizace Data framu spojením vektorů v R	106
7.7 Manipulace se sloupce Data framu v R	107
7.8 Aplikace filtrů na sloupce Data framu v R	107
7.9 Řazení v Data framu	107
7.10 Použití faktorů v R	107
7.11 Vytvoření ordinální proměnné v R	108
7.12 Načtení CSV souboru	108
7.13 Zápis dat do textového souboru	109
7.14 Základní statistické funkce v R	109
7.15 Vykreslování základních typů grafů pro dataset IRIS	110
7.16 Funkce <i>plot</i> - jednotlivé typy grafů	110
7.17 Tvorba koláčových grafů v R	111
7.18 Instalace a aktivace tidyverse	112
7.19 Načtení dat o COVID z John Hopkins university do R	113
7.20 Zpracování dat o potvrzených případech pomocí R	114
7.21 Zpracování dat o potvrzených případech úmrtí v důsledku COVID-19 pomocí R	116
7.22 Propojení souvisejících dat do jednoho celku pomocí R	117
7.23 Sedmidenní klouzavý průměr nakažených v ČR, Nizozemí a Švédsku	118
7.24 Logaritmické osy grafu - demonstrace vizualizace exponenciálního růstu	119
7.25 Sloupcový graf - vývoj denního počtu nově diagnostikovaných COVID-19 v ČR za poslední 2 týdny	120

Pravidlové metody	123
8.1 Doplnění chybějících údajů pomocí Hmisc knihovny funkcí impute	125
8.2 Metoda rozděl a panuj - strom klientů banky	129
8.3 Metoda rozděl a panuj - strom klientů banky; převod do podoby rozhodovacího seznamu	130
8.4 Binární strom přidělování úvěrů klientům banky	131
8.5 Binární strom přidělování úvěrů klientům banky - metriky a výsledky	132
8.6 Binární strom přidělování úvěrů klientům banky - hodnocení kvality modelu	132
8.7 Regresní strom pro AMES dataset	135
8.8 Rozhodovací pravidla	141
8.9 Načtení datasetu Titanic do R	143
8.10 Ukládání a načítání dat v R	143
8.11 Odvození asociačních pravidel pro dataset Titanic	144
8.12 Asociační pravidla pro děti (dataset Titanic)	145
8.13 Grafická interpretace asociačních pravidel (dataset Titanic)	145
Bayesovská klasifikace	149
9.1 Naivní Bayesův klasifikátor pro řešení problému klasifikace klientů banky	152
9.2 Naivní Bayesův klasifikátor pro řešení problému klasifikace kosatců na základě vlastností jejich okvětních lístků	154
Neuronové sítě	157
10.1 Řešení funkce druhé mocniny pomocí neuronové sítě	162
10.2 Klienti banky - řešení neuronovou sítí	166
10.3 AMES dataset - řešení pomocí nnet	167
10.4 AMES dataset - řešení pomocí nnet s optimalizací hyperparametrů a vyšším počtem iterací	169
10.5 AMES dataset - řešení pomocí nnet s optimalizací hyperparametrů; vyšším počtem iterací a preprocesingem prediktorů	169
Příloha 1 - UPS timer ve VBS	185
Oživení časovače Promotic (příklad) ve Visual Basic	185

Úvod

Vážený studente, dostává se Vám do rukou učební text předmětu *Bezpečnostní informatika 3* a předmětu *Informatika v bezpečnosti*¹.

Ačkoliv název skript má na svém konci číslovku 3, text skript je samonosný a jeho pochopení proto nevyžaduje předchozí znalosti předmětů Bezpečnostní informatika (BI) a Bezpečnostní informatika 2, ačkoliv určitá znalost zejména v oblasti šifrování a základů počítačových sítí může poskytnout užitečný kontext pro probíranou problematiku.

Z hlediska požadovaných znalostí je tak žádoucí pouze základní (uživatelské) znalosti v oblasti **informační technologie (IT)** a ochota naučit se něco nového.

Skripta se zaměřují především do dvou základních oblastí a to *průmyslové automatizace* – především pak systémů pro vizualizaci technologických procesů, tzv. **Supervisory Control and Data Acquisition (SCADA)** systémů a základů dolování dat. Účelem je umožnit studentům proniknout do nové oblasti řízení (automatizace) procesů, na kterých jsou založeny prakticky všechny výrobní procesy, **Kritická infrastruktura (KI)**, ale např. také řízení klimatu v budovách a řada dalších aplikací.

Bezpečnost i takových systémů tedy má svou IT složku. Technologie samotná tedy může poskytnout řadu informací o svém stavu. Nehoda v technologii může nastat vlivem řady příčin, pokud však není způsobena vnějšími vlivy, jako je např. povodeň nebo teroristický útok, lze očekávat, že takové nehodě budou předcházet změny ve sledovaných veličinách. Tyto by mělo být možné identifikovat pomocí běžných statistických metod popřípadě metod strojového učení.

Právě metodám strojového učení je věnována druhá polovina skript. Tyto metody jsou univerzálně použitelné, tedy jejich využití je širší než pouze pro účely zpracování dat poskytovaných technologie, řídicími systémy apod. Dostanete tedy do rukou užitečný nástroj, který vám může pomoci řešení semestrálních projektů, Vaší diplomové práce i později pro řešení problémů, se kterými se potkáte v rámci své profesionální praxe.

Organizace textu

Pro zpříjemnění čtení jsem se také rozhodl zpracovat tento text formou vhodnou pro „distanční vzdělávání“, tak aby práce s ním byla co možná nejjednodušší. Z tohoto důvodu je text jednotlivých kapitol segmentován do bloků.

Každá kapitola začíná náhledem kapitoly, ve kterém se dozvíte, o čem budeme v kapitole mluvit a proč. V bodech se pokusím shrnout, co byste po prostudování kapitoly měli znát a kolik času by Vám studium mělo zabrat. Mějte prosím na paměti, že tento časový údaj je pouze orientační, nebudte proto prosím smutní nebo naštvaní, když ve skutečnosti budete kapitole věnovat o něco méně nebo více času.

Za kapitolou následuje shrnutí, ve kterém budou zdůrazněny informace, které byste si rozhodně měli zapamatovat (určitě Vám ale neuškodí, pokud si jich zapamatujete více).

To, že jste správně pochopili probíranou látku, si budete moci ověřit pomocí kontrolních otázek a testů, které by Vám měly poskytnout dostatečnou zpětnou vazbu k rozhodnutí, zdali jít dále nebo si vyhradit delší čas na opakování.

Kontrolní otázky a k nim příslušející odpovědi ale neslouží jako náhrada studia textu jednotlivých kapitol! Funkce otázek je tedy čistě zpětnovazební. Pokud tedy prostudujete kapitolu a potom se zamyslíte nad otázkami a jste schopni je odpovědět, máte velmi dobrou šanci, že chápete minimálně základy. Opačně to ale nefunguje! Pokud prostudujete otázky a jejich odpovědi, nemůžete očekávat, že získáte ekvivalentní znalosti studiu textu skript jako takových.

Pro zjednodušení orientace v textu jsem zavedl systém ikon:

¹Předměty Bezpečnostní informatika 3 a Informatika v bezpečnosti jsou obsahově ekvivalentní.

**Průvodce studiem**

Slouží pro seznámení studentů s látkou, která bude v kapitole probírána.

**Čas nutný ke studiu**

Představuje odhad doby, který budete potřebovat k prostudování celé kapitoly. Jedná se pouze o orientační odhad, neznepokojujte se proto, pokud Vám studium bude trvat o něco déle nebo budete hotovi rychleji.

**Vysvětlení, definice, poznámka**

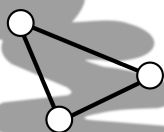
U této ikony najdete vysvětlující text, poznámku k probíranému tématu, která problém uvede do širších souvislostí, popřípadě důležitou definice.

**Kontrolní otázky**

Na závěr každé kapitoly je zařazeno několik otázek, které prověří, zda jste problematice kapitoly dostatečně porozuměli. Pokud nebudete vědět odpověď na některou otázku, je to signál pro Vás, abyste se ke kapitole vrátili.

**Příklad**

Příklady obsahují praktické demonstrace diskutovaného problému.

**Návaznosti**

V tomto segmentu budou zmíněny další návaznosti probíraného tématu na další témata tohoto předmětu, ale také dalších předmětů.

**Shrnutí**

Obsahuje základní myšlenky kapitoly, kterým by měl být věnována zvláštní pozornost během studia.

Text kapitol je doplněn interaktivním seznamem literatury, tabulek, obrázků a také rejstříkem základních pojmů. Používejte je dle potřeb, mají potenciál zjednodušit Vaše studium, minimálně tohoto předmětu.

**Přestávka**

Po obtížné části textu, nebo prostě občas jenom tak je nutné si udělat krátkou přestávku, načerpat síly k novému studiu.

Aktivní odkazy jsou v textu zvýrazněny červenou (a v případě odkazů na literaturu zelenou) barvou.

Na konec skript byl přidán také rejstřík pojmů. Doporučuji, abyste jej v rámci přípravy na zkoušku prošli - zamyslete se nad tím, zda všechny pojmy, které jsem do něj zařadil, chápete a jste je schopni dát do souvislostí. Pokud ne je vedle pojmu odkaz na číslo stránky, kde je pojem probrán a Vy můžete rychle zaplnit případné mezery ve svých znalostech problematiky informačních systémů.

Přeji Vám, aby čas, který strávíte s tímto textem, byl co možná nejpříjemnější a abyste jej nepovažovali za ztracený.

doc. Ing. Pavel Šenovský, Ph.D.

Poznámka autora:

Právě držíte v rukou (ať iž fyzicky nebo virtuálně) páté rozšířené vydání skript. Je možné, že právě studujete na zkoušku, nebo jste se ke skriptům dostali pouze náhodou po delší době. Z tohoto důvodu by se Vám mohlo hodit stručné shrnutí změn mezi jednotlivými vydáními těchto skript.

Novinky v 5. vydání

- provedeny drobné úpravy v příkladu na rozhodovací strom řešení pomocí knihovny tree.
- drobné opravy ve všech částech skript
- v kapitole věnované automatizaci doplněna problematika DCS, vysvětleny také základy Průmyslu 5.0 a 6.0
- Příklad Promotic byl předělán do verze Promotic 9.0 (z předchozího 8.3 a vysvětlením rozdílů proti 9.0)
- v kapitole 5 je diskutován nový zákon o kybernetické bezpečnosti (2025), místo původního (2014)
- přidána nová kapitola věnována použití AI obecně i v automatizaci, včetně omezení a určitých výhledů do budoucna

Novinky ve 4. vydání skript

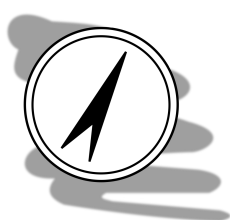
- aktualizována sekce věnovaná neuronovým sítím - vylepšena prezentace skriptů v R
- drobné úpravy v textu a opravy chyb
- v části věnované Promotic doplněny poznámky k verzi 9 systému
- upraveny a doplněny příklady v části věnované dataminingu
- doplněny popisky výpisů zdrojových kódů
- doplněn seznam výpisu zdrojových kódů

Novinky v 3. vydání skript

1. použit jiný systém sazby, což umožnilo přiložit je skriptům rejstřík, různé seznamy a to v interaktivní podobě
2. stávající témata prošla revizí
3. skripta byla zcela zásadním způsobem přepsána

Kapitola 1

Základy průmyslové automatizace



Náhled kapitoly

V této kapitole se podíváme na specifika průmyslové automatizace. Zaměříme se přitom zejména na popis různých druhů zařízení a způsobu, jakým se využívají

Po přečtení této kapitoly budete vědět

1. jaká zařízení se používají pro automatizaci procesů a
2. jakým způsobem se používají (co je jejich účel)



Čas pro studium

Pro prostudování této kapitoly budete potřebovat přibližně hodinu.

1.1 Automatizace

Automatizace je základním nástrojem pro řízení technologických celků. Automatizací rozumíme zavádění automatizovaných systémů pro měření a regulaci procesů. Úkolem regulace je zajistit plynulý (ve smyslu rychlosti procesu i kvality) provoz regulované soustavy.

Nejprve se automatizace zaváděla do velkých výrobních linek, dnes se však automatizují prakticky všechny procesy od procesů výrobních, po systémy klimatizační ve velkých budovách až k regulaci kotle ve Vašem rodinném domě.

Obecně se tedy automatizace stará o automatizovaný chod regulované soustavy tak, že pro řízení jejího provozu automaticky aplikuje předem stanovené postupy a sleduje, zda změna v nastavení vedla k žádoucím změnám v chování regulované soustavy. V případě, že automatizovaná regulace nevede k zastabilizování soustavy, automatizace selhává a na řadu přichází buďto zásah operátora nebo odstavení soustavy. Ta pak musí být následně manuálně znovu spuštěna.

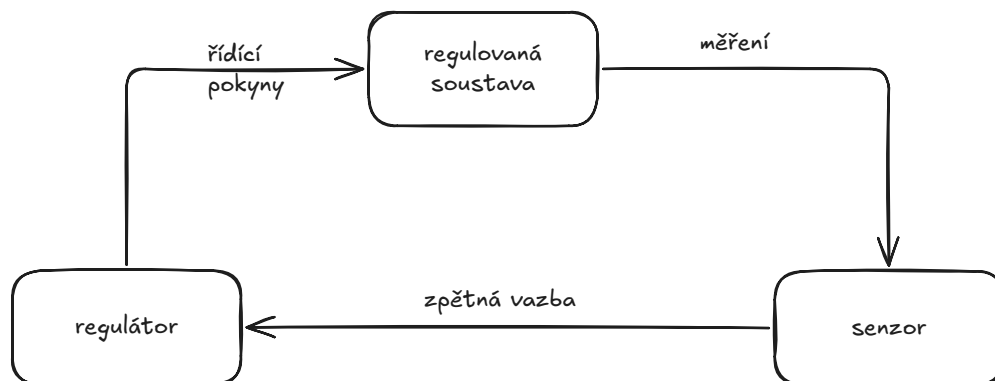
Uvažujme tři základní prvky automatické:

1. regulovaná soustava (to co je regulováno)
2. senzory (čidla schopná sledovat jednotlivé zájmové proměnné v regulované soustavě nebo vlivy na ni působící)
3. regulátor (zařízení, které bude realizovat samotný akt regulace, např. páky, ventily, apod.)

Vizuálně je možné si výše uvedené prvky představit jako na obr. 1.1. Trojici těchto prvků označujeme jako *regulační soustavu*. **Regulační soustavou** přitom rozumíme jakoukoliv soustavu, která může být regulována. V regulační soustavě se obvykle používá velké množství různých senzorů, snímajících různé veličiny, v různé kvalitě a různých místech regulované soustavy, a často také regulátorů.

Regulovaná soustava (v regulační soustavě) je obvykle pouze jedna, ale v praxi takové soustavy často řadíme za sebe, takže na sebe logicky navazují.

V takovém případě výstup jedné soustavy slouží zároveň také jako vstup soustavy druhé. To je možno si představit např. jako výrobní linku. Regulační soustavu pak bude tvořit každé zastavení dílů na této lince.



Obrázek 1.1: Obecné schéma regulační soustavy



Příklad

Tento koncept si lze jednoduše představit na příkladu řízení teploty v místnosti pomocí klimatizace. Předpokládejme že tato klimatizace je vybavena kromě schopnosti chlazení také tepelným čerpadlem a tedy schopností přitopit. Pro ni mohou být nastaveny dvě meze - horní a dolní - v případě, že bude přesáhnutá horní mez, spustí se klimatizace. Senzor teploty umožní řídicímu systému rozhodnout, zdali intenzita chlazení je dostačující a v případě potřeby ji upraví.

Obdobným způsobem bude pracovat také teplotní čerpadlo. V případě, že v místnosti poklesne teplota pod stanovenou dolní mez, zapne se přitápění pomocí tepelného čerpadla. Řídicí systém opět na základě informací podle údajů získaných ze senzoru teploty v místnosti rozhodne, zda je intenzita vytápění dostatečná, popřípadě zda již není žádoucí vytápění zastavit.

Regulace je tedy z tohoto pohledu závislá na informacích poskytovaných senzorem, v tomto příkladu se jednalo o senzor teplotní. Sensory na řízeném systému jsou vyhodnocovány automaticky v určitém časovém intervalu. To umožňuje řídicímu systému regulovat a zároveň mu poskytuje zpětnou vazbu o tom jaký efekt mají regulační zásahy řídicího systému regulované soustavy.

Existuje nepřeberné množství typů **senzorů** - kromě teplotních čidel, lze měřit tlak, otáčky, vlhkost, výšku hladiny a celou řadu dalších veličin nebo stavů systému. Moderní systémy jsou dokonce v současnosti často schopny vyhodnocovat automaticky obraz zachycený kamerou a získat z něj informace potřebné pro regulaci - často s použitím metod strojového učení.

1.2 PLC

Zařízením, které často zajišťuje automatickou regulaci je **Programmable Logic Controller (PLC)**. Určitou představu o vzhledu **PLC** si lze udělat z obr. 1.2.

Na obr. 1.2 jsou PLC zařízení uprostřed (pod zásuvkou). Jednotlivé **PLC** se zasunují do patřičného řídicího systému. Na obr. 1.2 je vidět jedna zatím neobsazená police, kam by případně bylo možné zasunout další jednotku PLC. Počet PLC a velikost řídicího systému se odvíjí od požadavků, které jsou na něj kladeny. Složitě technologické celky proto mohou pro svůj provoz vyžadovat klidně i desítky nebo stovky PLC zařízení a sofistikovaný řídicí systém.

Formálně je PLC počítač. obsahuje tedy procesor, základní desku, paměť a rozhraní umožňující připojování periférií. Existuje zde ale poměrně velké množství rozdílů. K PLC např. není možné



Obrázek 1.2: Příklad PLC a řídicího systému (převzato z [10])

připojit běžné periferie jako je monitor, klávesnice a myš. Komunikace s tím se pak děje buďto po síti nebo zprostředkovaně přes další zařízení jako je SCADA systém nebo RTU (oběma se budeme zabývat za chvíli). Naopak poskytuje připojení pomocí rozhraní, která se u běžných počítačů nepoužívají.

PLC je také přizpůsobeno nasazení v technologických provozech. To znamená, že se předpokládá, že ideálně by se mělo jednat o zařízení bezúdržbové, jehož životnost bude stejná, jako je životnost řízeného systému. PLC tak může pracovat 10, 15 nebo i více let. Životnost PC je v současnosti přibližně odhadována na 6 let (podle konfigurace a způsobu užití).

Delší životnosti je dosahováno tím, že PLC jsou strukturálně jednodušší a obsahují minimum pohyblivých částí, které jsou náchylnější k selhání. Jsou také obvykle umisťována do prostorů, kde existuje určitá kontrola nad provozním prostředím z pohledu vibrací, teploty, vlhkosti, prašnosti apod. Může se jednat např. o speciálně přizpůsobené skříně nebo celé místnosti (serverovny).

PLC také nejsou univerzální, alespoň ne stejným způsobem jako běžné počítače. PLC je sice možné programovat, ale toto programování se děje na běžném počítači ve vývojovém prostředí poskytnuté

výrobce PLC. Připravený program je následně nahráván na PLC pomocí přenosných médií nebo po síti, podle toho, jak je PLC staré a jakým způsobem je zapojeno, popř. konfigurováno. PLC pak následně bude *vykonávat tento*, ale pozor také *pouze tento* program.

To také ale znamená, že PLC různých výrobců (ani jejich řídicí systém) není možné zaměňovat - jedná se o ucelená řešení na míru požadovaného nasazení (řízené technologie). Doba podpory těchto zařízení, ze strany výrobce, ale nepokrývá obvykle celou předpokládanou dobu nasazení PLC. Často přitom nasazení PLC nové generace daného výrobce není možné z důvodu zpětné nekompatibility. U velkých technologických celků organizace řeší tento problém tak, že se prostě předzásobí dostatečným počtem náhradních PLC na začátku životního cyklu celého technologického celku. Následně pak v případě selhání jednotlivých PLC může relativně snadno provést výměnu kus za kus.

Vraťme se ale ke způsobu řízení. PLC je tedy určeno k tomu, aby řízení pokud možno automatizovalo - tedy automaticky prováděla zásahy do technologie s cílem udržet ji v chodu uvnitř nastavených parametrů. Automatizace pokrývá běžné životní situace řízené technologie a je schopna se s nimi automaticky vypořádat. Jelikož ale není možno předem předvídat všechny možné stavy a problémy, kterým technologie z různých důvodů bude v budoucnu vystavena, schopnosti automatizovaného řízení budou vždy omezeny, byť moderní metody strojového učení a širší nasazení technologií tzv. *Průmyslu 4.0* schopnosti autonomní činnosti zařízení výrazně rozšiřuje.

1.3 SCADA

Z předchozích odstavců již víme, že schopnost automatického řízení má řadu omezení a proto pro složitější technologické celky je potřeba tento typ řízení vhodně kombinovat s řízením manuálním. Jelikož tento typ řízení není u řady instalací používán příliš často - bývá realizován obvykle dálkově a centralizovaně formou **dispečerská/dohledové/kontrolní pracoviště**.

Tento typ pracovišť slouží k vizualizaci dat předávaných kontinuálně technologií a její prezentaci tak, aby operátor byl schopen detekovat nežádoucí stavy a efektivně zasáhnout. Pro provoz takového pracoviště je bezpodmínečně nutné, aby dispečer měl přístup k aktuálním datům popisujících *současný stav technologie* a to v takové formě, aby byl schopen:

- identifikovat příčinu problému,
- navrhnout a spustit účinné řešení identifikovaného problému,
- zkontrolovat účinek řídicích operací.

To není zcela triviální problém - i jednoduché technologie jsou schopny generovat velké objemy dat z připojených senzorů. To je dáno tím, že údaje ze senzorů jsou snímány v relativně krátkých časových intervalech (od zlomků sekund až po minuty podle toho, co je řízeno) a připojených senzorů je obvykle velké množství. Zobrazení těchto dat v tabelární formě proto nepřichází v úvahu. Data potřebujeme prezentovat uživateli/operátorovi „inteligentně“ tak, aby operátor nebyl přetížen přemírou informací a byl schopen je efektivně použít.

Z tohoto důvodu se často nasazují tzv. **SCADA** systém. Tyto systémy tvoří další vrstvu v logice průmyslové automatizace. Připomeňme si, že tu nejnižší tvoří **PLC** automaty regulující proces v reálném čase. SCADA systém, jelikož údaje musí, načíst (obvykle po síti), zpracovat a zobrazit funguje ve skoro reálném čase - tedy čase, který se blíží reálnému času, ale už je zde jistě drobné zpoždění.

SCADA systém přitom informace obvykle nebere přímo ze senzorů (prostřednictvím PLC), ale z nějakého centralizovaného k tomuto účelu určeného místa, často výkonného databázového serveru. Takový server označujeme často jako *data historian*.

V některých případech so řídicího systému přidáváme ještě jednu mezivrstvu a to *distribuovaný kontrolní systém (Distributed Control System) (DCIS)*.

Takový systém se liší poměrně zásadně od SCADA systému. Distribuovaností rozumíme že řízení přenášíme blíže k technologii, byť i zde jsou používány PLC a RTU jednotky. Řízení je tak do jisté míry rozděleno - ve smyslu, že jednotlivé komponenty takového řídicího o sobě vzájemně nevědí.

Porovnání základních vlastností SCADA a DCS je dostupné v tab. 1.1.

Tabulkou prezentované rozdíly prosím berte spíše jako orientační. Pro podrobný rozbor nuancí této problematiky nebudeme mít prostor. Z hlediska našeho výkladu a pozdějších experimentů s tvorbou v rámci Promotic se proto zaměříme pouze na SCADA systém.

Na databáze, do kterých jsou údaje ze senzorů vkládány jsou kladeny určité specifické nároky. Informace musí poskytovat s minimálním časovým zpožděním. Primárním cílem je pak, aby poskytované údaje byly aktuální, ale nikoliv nutně úplné. To je poměrně zásadní rozdíl oproti nárokům, které jsou

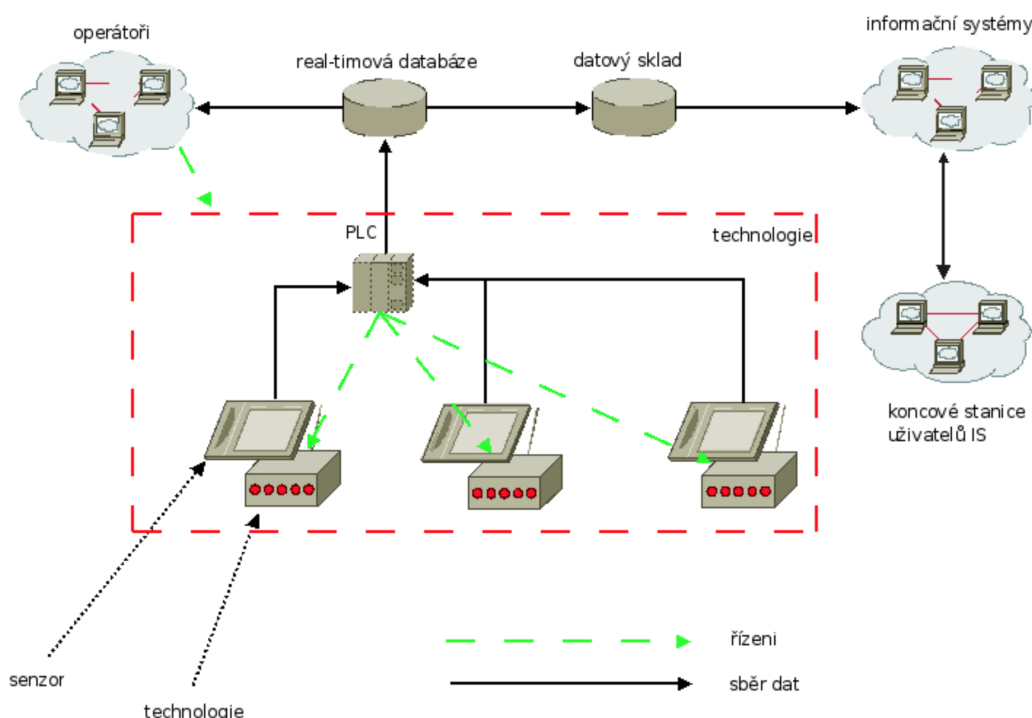
Tabulka 1.1: Srovnání SCADA a DCS

vlastnost	SCADA	DCS
architektura	centralizovaný	decentralizovaný
data	data historian	technologie
řízení	DCS/PLC/RTU	PLC/RTU
lokace	rozsáhlé geog. oblasti	jedna lokalita
čas řízení	spíše dohled, sběr dat	řízení v reálném čase

kladeny na databáze používané v běžných informačních systémech, kde naopak je primárním úkolem udržet úplnost uchovávaných dat.

Pro tyto specifické nároky jsou někdy tyto databáze označovány jako *real-timové*. Problematicke reálnitímových databází se budeme věnovat v samostatné kapitole.

Celkové zjednodušené zapojení SCADA systému, databáze, PLC a řízené technologie je znázorněno na obr. 1.3.



Obrázek 1.3: Zapojení SCADA systému, databáze, PLC a řízené technologie

Na obr. 1.3 operátoři/dispečeři pracují s informacemi poskytovanými dohledovému pracovišti SCADA systémem. Aktuální data jsou shromažďována z automatizované technologie v reálnitímové databázi. Operátoři v případě potřeby mohou do řízené technologie přímo zasáhnout pomocí předem připravených mechanismů dálkového řízení.

V případě, že dálkový zásah není z nějakého důvodu možný, zbývá možnost manuální zásahu, na místě. Manuální zásah je často skutečně až to poslední možné řešení. Je potřeba si uvědomit, že řídicí pracoviště může být geograficky vzdálené, např. dohledové pracoviště ŘSD bdící nad celou dálniční sítí včetně všech silničních tunelů je provozována z jediného místa - Ostravy. Pokud je nutný manuální zásah, jsou vysílány na místo problému technici přímo provozovatele dané technologie nebo technici jiných společností, kteří k tomuto účelu byli nasmlouváni. Při zásahu je potřeba počítat s tím, že doba zpohotovnění zásahového týmu a jeho dopravení na místo může být poměrně dlouhá podle charakteru problému, vyžadovaném materiálu, zařízení a kompetencí techniků.

Všimněte si také druhé linie aktivit využívajících informací z technologie. V tomto případě jsou data o technologii přelévána do datového skladu pro využití v běžných informačních systémech organizace. Technologie může poskytovat řadu cenných informací, např. o spotřebě zdrojů, jejichž dodávky je

potřeba plánovat, např. pomocí systémů **Enterprise Resource Planning (ERP)**. Řada výrobních linek může mít také určité schopnosti identifikovat některé vady výrobků. Informace tohoto typu jsou opět velmi cenné a organizace je může (a měla by) chtít s nimi dále pracovat, např. v systému řízení kvality.

Podle charakteru řízené technologie dispečerské pracoviště může vypadat různě. Ředitelství silnic a dálnic třeba provozuje v Ostravě pro celou dálniční síť **NDIC**. Určitou představu o vzhledu si lze udělat z obr. 1.4.



Obrázek 1.4: Dohledové pracoviště **NDIC** (převzato z [45])

NDIC zpracovává údaje z řady různých zdrojů zejména pak [82]:

- Detekce intenzit dopravy - cca 200 detektorů
- Silniční meteorologický systém - cca 270 meteohlásek
- Vysokorychlostní váhy - 11 kusů
- Systém elektronického mýta
- Systém sčítání dopravy, detekce kolon a sledování dopravního proudu
- Dohledový kamerový systém - více než 600 kamer

Pro srovnání menší instalace SCADA systému by mohla vypadat podobně jako na obr. 1.5. V tomto případě je monitorována menší fotovoltaická elektrárna.

Vizualizace řízeného technologického procesu je vždy realizována na míru řízené technologii. SCADA systém jako takový je obvykle provozován na běžných počítačích s operačním systémem Microsoft Windows. Z hlediska údržby se do SCADA systému obvykle nezasahuje, pokud to nevyžadují změny v provozované technologii samotné. Pro řízenou technologii je tedy vyvinuta aplikace ve vývojovém prostředí zvoleného SCADA systému. A následně je tato aplikace dlouhodobě provozována pro monitoring a řízení této technologie.

Ani SCADA systém (jako platforma) není imunní vůči morálnímu zastarávání. Provoz vyvinuté aplikace na novější (podporované) verzi zvoleného SCADA systému může technicky realizovatelné, ale velmi často je taková aktualizace považována za rizikovou. Přejít mezi různými verzemi stejného systému s sebou může totiž nést nekompatibilní změny, které navíc se mohou projevit pouze za určitých specifických podmínek. Odladění takového systému je tak poměrně náročné a vyžaduje čas a peníze.

Z tohoto důvodu jsou aplikace SCADA provozovány obvykle na platformě, pro kterou byly vyvinuty a to i v případě, že již není nadále podporována výrobcem. K aktualizaci tak buď nedochází vůbec nebo



Obrázek 1.5: Řešení monitoringu fotovoltaické elektrárny (převzato z [28])

jsou aplikovány pouze bezpečnostní záplaty v provozované verzi systému (nedohází tedy k povýšení na novou generaci software).

Všimněte si také, že tím, že SCADA systém je provozován na běžném operačním systému, může být náchylný k stejným problémům, jako jsou běžné počítače v počítačové síti. Tyto problémy mohou být umocněny také případným stářím SCADA, popřípadě verzí operačního systému, na kterém je provozován (provozovaná verze SCADA systému např. nemusí podporovat nejnovější verzi operačního systému).



SCADA aplikace vs platforma

V textu výše se pojem SCADA objevuje ve dvou významech - jednak se jedná o platformu, na které je provozována vizualizace řízeného technologického procesu, jednak se tento pojem používá ve smyslu provozu samotné aplikace - vyvinutá a provozovaná aplikace je označována jako SCADA systém.

Uveďme si příklad: v samostatné kapitole těchto skript a také ve cvičení budeme pracovat se systémem Promotic [54] společnosti Microsys. V tomto systému lze vyvíjet a provozovat vizualizace různých technologických procesů - jedná se tedy o SCADA ve smyslu platformy. V systému Promotic vyvinuté aplikace jsou pak SCADA ve smyslu aplikace.



Efektivita investic vs bezpečnost

Ve výše uvedeném textu se pracuje poměrně extenzivně s životností, podporou apod. Co jsme nezmínili jsou otázky bezpečnosti. **Je možné považovat jednotlivé součásti řídicího systému za bezpečné i v případě, že už nadále nejsou podporované výrobcem?** Odpověď je poměrně složitá - *ne tak docela*. Co organizace mohou dělat, je pokusit se řídicí systém izolovat od „zlého vnějšího světa“ a snížit tak riziko spojené s jeho provozem. K tomu je však potřeba dodat, že úplná izolace všech komponent řídicího systému není také obvykle žádoucí opět z pohledu efektivity investic, zejména pak možnosti dálkové kontroly.

Možnostmi izolace řídicího systému se budeme zabývat později.

1.4 Další zařízení využívána v automatizaci

Existuje celá řada dalších zařízení, které jsou využívány pro automatizaci technologických procesů. Některými z nich se budeme zabývat v této kapitole.

Pro vzdálenou komunikaci jsou využívány často zařízení **Remote Terminal Unit (RTU)**. Terminálem v tomto případě máme na mysli ovládání pomocí příkazové řádky. V původních UNIXových operačních systémech pohánějících sálové počítače, byla příkazová řádka označována jako terminál. Reflektovala se tím situace, že veškerá práce byla realizována ze vzdáleného fyzického terminálu. Příkazy pak šly po síti do sálového počítače, který je vykonával.

Dnes jsou terminály čistě softwarové (někdy se označují také jako emulátory terminálů). Základní idea textové komunikace po síti, ale zůstala stejná. Tato forma komunikace má některé výhody:

- text je dobře komprimovatelný - komunikace tedy vyžaduje pouze minimální přenosovou kapacitu. Pro komunikaci tak může postačovat mobilní připojení dosluhujících sítí druhé generace např. **General Packet Radio Service (GPRS)** nebo **Enhanced Data Rates for GSM Evolution (EDGE)**.
- text lze efektivně zpracovávat - RTU jsou proto sice málo výkonná, ale zato levná.
- spojení lze efektivně šifrovat

Co do použití představují RTU spojení mezi vzdálenou řízenou technologií a řídicím pracovištěm. Další podstatné vlastnosti vycházejí z jiných výkladů zkratky RTU - někdy jsou totiž tato zařízení označována jako Remote Telemetry Unit nebo Remote Telecontrol Unit.

Tyto názvy vedou na vlastnosti - poskytování telemetrických dat o kontrolované technologii a možnosti tuto technologii ovládat.

Textový charakter komunikace nutně nemusí být realizován v celém komunikačním procesu. Na straně operátora může třeba fungovat SCADA systém, který příkazy zadávané operátorem v grafickém uživatelském rozhraní převede do textové formy a odešle do vzdáleného RTU. Data předaná zpět v textové podobě pak opět mohou být převedena do grafické podoby.

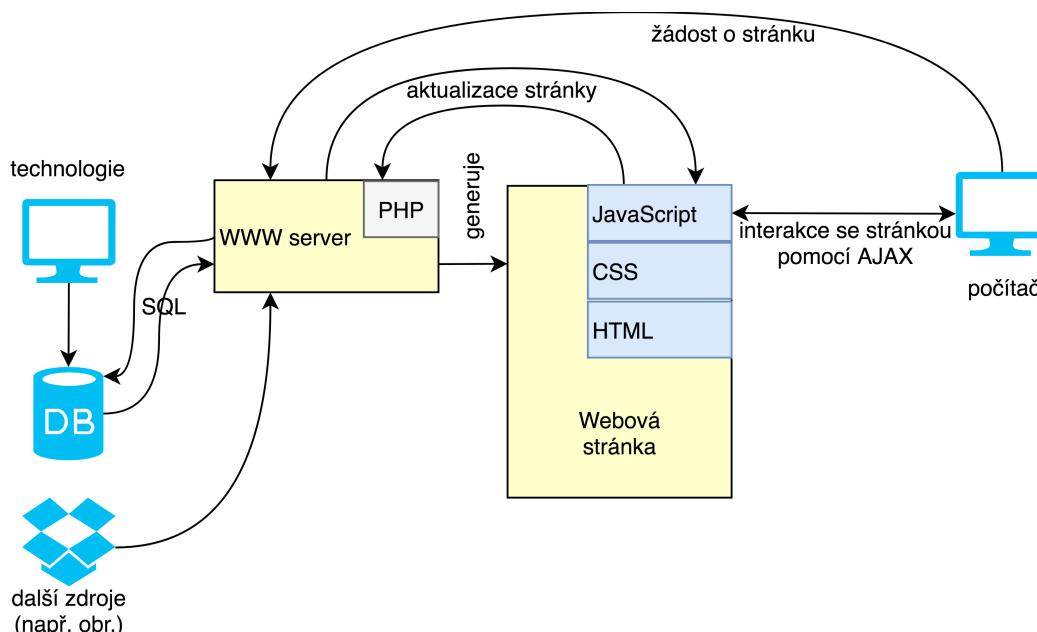
U rozhraní se na chvíli zastavíme. Problematika **rozhraní člověk - stroj (Human Machine Interface (HMI))** je totiž také jedním ze zájmů automatizace. Rozhraní může být realizováno mnoha různými způsoby:

- forma
 - grafické uživatelské rozhraní (**Graphical User Interface (GUI)**)
 - textové rozhraní (viz RTU)
- centralizace
 - centralizované - rozhraní je v řídicí místnosti
 - decentralizované - rozhraní je přímo na řízené technologii
- přístupnost
 - přístupné po síti (např. RTU nebo webové rozhraní)
 - místní
- realizace
 - fyzická (tlačítka, páky, přepínače, další fyzicky realizované regulační prvky ...)
 - virtuální (rozhraní je realizováno softwarově)
- a další ...

Většinu z výše uvedených typů rozhraní není potřeba blíže představovat, neboť tvoří každodenní součást našich životů, přesto si dovoluji věnovat určité místo webovému rozhraní, které v posledních letech nabylo na popularitě díky své relativní malé náročnosti a možnosti využít takové rozhraní prakticky odkudkoliv po síti. Pokud se ale podrobněji zaměříme na implementaci takového rozhraní, zjistíme, že strukturálně není úplně jednoduché, viz obr. 1.6.

Jádrem webového rozhraní je webový server obvykle doplněný a zvolenou platformu, na které je rozhraní implementováno. V našem případě se jedná o programovací jazyk PHP. Pokud chceme k rozhraní přistoupit - zadáme adresu zařízení v síti do webového prohlížeče. Webový prohlížeč vyšle požadavek HTTP nebo HTTPS (podle toho, zda je komunikace šifrovaná) GET a webový server odpoví jedním z kódů a pokud je vše v pořádku (kód 200) vrátí i požadovaný zdroj (stránku).

Webová stránka je obvykle generována v reálném čase webovým serverem na základě implementovaného programu, v našem případě v jazyce PHP. Kód aplikace může být poměrně složitý a přistupovat



Obrázek 1.6: Možná struktura webové aplikace zajišťující rozhraní člověk - stroj

k dalším zdrojům. Může se jednat o fragmenty souborů, ale třeba také data obsažená v nějaké databázi. Schopnost manipulovat s určitým typem databází může být přímo integrální součástí zvoleného programovacího jazyka, nebo se k manipulaci s ní používá obecnější rozhraní jako ODBC nebo JDBC.

Bez ohledu na to, jaké rozhraní k databázi je použito, samotné dotazování bude realizováno (až na určité výjimky) pomocí jazyka **Structured Query Language (SQL)**.

Vygenerovaná stránka pak obvykle obsahuje:

- **Hypertext Markup Language (HTML)** pro sémantický popis struktury a obsahu stránky
- **Cascading Style Sheets (CSS)** - přiřazující HTML stránce určitý vzhled
- stránka může být doplněna JavaScriptem umožňujícím implementaci určitých interakcí jinak typickým pro aplikace běžné na desktopu

JavaScript např. může pomocí **Asynchronous JavaScript and XML (AJAX)** aktualizovat pouze určité specifické části stránky, aniž by bylo nutné načíst znovu stránku celou. Tímto způsobem lze poměrně efektivně obejít technické omezení vynucené realizací komunikace na protokolech TCP/IP, které jsou bezstavové. Veškeré interakce se stránkou jsou tak řešeny samostatným požadavkem na server. Při použití AJAX jsou ale tyto požadavky omezené pouze na manipulaci s daty apod. - struktura stránky a větší část jejího obsahu tak obvykle zůstává nezměněna.

AJAX tak nejen zvyšuje komfort při použití, ale fungování webové aplikace také výrazně zrychluje.

I pro jednoduché webové rozhraní je nutno najednou provozovat spoustu nástrojů a každý z nich jednotlivě je potřeba specifickým způsobem z podle bezpečnosti udržovat.

Po přečtení předchozích odstavců Vás možná napadne otázka **PROČ?** Proč bych měl vědět něco o struktuře a fungování webových aplikací? Jakou to má souvislost s oborem, který studují? A možná Vás napadnou v podobném duchu i další otázky.

Důvod je prostý. V praxi ať už se budete zabývat řešením bezpečnostních plánů, nebo běžnými provozními záležitostmi ve smyslu požární ochrany nebo bezpečnosti práce, nebo dokonce pokud budete řešit bezpečnost systémů kritické infrastruktury - budete muset pro svou práci přijmout jisté předpoklady o způsobu chování řízeného systému.

Bezpečnostní specialista postupuje obvykle tak, že pracuje s různými scénáři možných havárií. Pro každý scénář pak odvozuje riziko a zkoumá způsoby jak s ním naložit. Variant řešení je obvykle k dispozici velké množství. Podobně velké množství je ale také různých scénářů selhání technologie.

Z hlediska možných opatření lze jít po příčinách, tedy zabránit iniciaci nebezpečí nebo lze přerušit popř. minimalizovat následky takové poslušnosti. K tomuto účelu je ale obvykle potřeba do technologie určitým způsobem zasáhnout - tedy regulovat ji.

Jak ale takovou regulaci realizovat? Už víme, že technologické procesy mají řadu vrstev. Některé fungují automaticky (PLC), na základě nahraného programu, jiné předpokládají dohled popř. zásah člověka (SCADA). Pokud budu muset zasáhnout, budu toho schopen? Co dělat v případě, že z nějakého důvodu nelze technologii řídit dálkově? Mám na místě člověka, který může provést požadovaný manuální zásah, nebo budu muset na místo vyslat pracovníka/specialistu? Jak dlouho bude trvat, než se tam dostane a když se dostane na místo, jak dlouho bude trvat, než začne pracovat na lokalizaci problému?

V případě, že bychom nevěděli nic o podstatě automatizace musíme se spolehnout čistě na názor specialisty na automatizaci. Tento specialista může být vynikajícím odborníkem na automatizaci, ale velmi pravděpodobně jeho znalosti z pohledu práce s rizikem nebudou příliš valné. Určitá úroveň porozumění problematice automatizace nám neumožní takového specialistu nahradit, z pohledu řízení bezpečnosti, ale můžeme přestat na něm být zcela závislí - můžeme se stát jeho partnery - pro oblast řízení bezpečnosti.

Získané znalosti z tohoto předmětu by Vám tedy měly umožnit, aby jste mohli klást otázky a abyste, alespoň trochu rozuměli odpovědím. (aby odpověď pro Vás nebyla v řeči vyhynulých kmenů.)

1.5 Průmysl 4.0

V poslední části této kapitoly se zaměříme na pojem *Průmysl 4.0*. Tento pojem je v posledních letech využíván velmi často, obvykle v souvislosti se zdůrazněním změn, které probíhají v současnosti v transformaci procesů ve výrobních, ale také nevýrobních společnostech. Tyto změny jsou někdy označovány také jako *4. průmyslová revoluce*.

Zkusme projít jednotlivé „revoluce“ v minulosti a zkusme odhadnout celkový transformační potenciál revoluce této.

1. *průmyslová revoluce* proběhla v 18. století a transformovala tehdejší společnost zaměřenou na zemědělství a venkov do podoby industrializované společnosti centralizované do měst. Tento přerod byl umožněn nástupem velkých výrobních celků schopných sériové produkce zboží. Jako pohon těchto zařízení sloužily nově vyvinuté parní stroje.

2. *průmyslová revoluce*, někdy označovaná také jako technologická revoluce, se vyznačuje rychlou standardizací a industrializací. Tuto revoluci obvykle datujeme někdy mezi léta 1870 - 1914 (začátek 1. světové války). Pokroky dosažené ve výrobních procesech umožnily široké nasazení systémů jako je telegraf, železnice, inženýrské sítě (plynovody, vodovody, kanalizace). Taktéž byla představena řada nových technologií z nichž nejvýznamnější je elektrifikace a telefony.

3. *průmyslová revoluce* se datuje do druhé poloviny 20. století, konkrétně po roce 1969 a je obvykle spojována s elektronizací společnosti. Tato revoluce tak zahájila transformaci společnosti s čistě analogové do digitální. Tato transformace byla umožněna rozvojem výpočetní techniky - především osobních počítačů a jejich propojení počítačovými sítěmi. Ačkoliv proces transformace společnosti do její digitální podoby nebyl zcela dokončen, lze říci, že transformační potenciál nástupu počítačů byl již vyčerpán.

V oblasti průmyslu jsou pro tuto etapu typická zařízení PLC, která jsme již v této kapitole popisovali, a nástup robotizace. Robotizací se rozumí nasazování jednoúčelových automatizovaných systémů, automatizující úkon ve výrobním procesu, který byl do té doby závislý na práci člověka. Tato etapa robotizace však nemá potenciál nahradit člověka ve výrobním procesu zcela. Automatizovány jsou tak především jednoduché kroky ve výrobě, které lze dobře popsat zefektivnit její výkon nasazením automatizovaného systému.

4. *průmyslová revoluce* ... není exaktně definovaná. Jedním z důvodů je, že probíhá v současnosti a tak lze pozorovat jisté počínající revoluční změny ve společnosti, ale zároveň není možné úplně přesně odhadnout kam tyto změny povedou. Z tohoto pohledu je proto označení 4. průmyslová revoluce (Průmysl 4.0) spíše marketingovým označením.

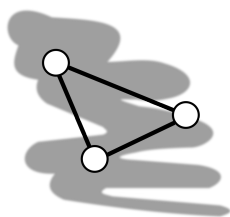
Co však lze udělat již dnes je popsat hnací motor těchto změn - což jsou čtyři základní komponenty, které lze pak dále rozpracovávat do větších podrobností:

- kyber-fyzické systémy
- **Internet of Things (IoT)**, tedy internet věcí (výměna informací mezi IoT zařízeními v cloudu, big data)
- dostupnost výpočetních zdrojů na vyžádání (cloudové služby, širokopásmové připojení k Internetu)
- strojové učení (big data, metody strojového učení včetně umělé inteligence)

Společně výše uvedené komponenty mají potenciál posunout procesy nejen ve výrobních společnostech na novou úroveň. Shromažďování dat z různých typů senzorů včetně např. obrazové informace pořízené kamerami mohou být zpracovány pomocí pokročilých metod *strojového učení* k odvození požadovaných činností automatizovaného systému.

Průběh procesu tak nemusí být explicitně popsán, musí být pouze kvalitně zachyceny, v dostatečné míře podrobnosti, data popisující tento proces. Metody strojového učení, především pak metody tzv. *hloubkového učení* jsou schopny tento proces formalizovat.

Takový přístup umožňuje automatizovat činnosti, které v minulosti automatizovat nebylo možné. K tomu je ale potřeba dodat, že v současnosti není možné automatizovat všechny činnosti ve výrobních procesech. Výstupem transformace je tak v současnosti pouze užší spolupráce člověka a stroje. V současnosti není zcela jasné, zda se jedná pouze o přechodný stav, který bude postupně s vývojem technologií ustupovat, nebo se jedná o stav trvalejší determinovaný samotnou podstatou používaných procesů.



Metody strojového učení

V tomto předmětu se budeme vybranými metodami strojového učení zabývat v druhé polovině semestru a tím pádem také druhé polovině těchto skript.

Z výše uvedeného je potřeba si vzít jisté poučení. Lze předpokládat, že transformace společnosti vyvolaná nástupem různých technologií a postupů souhrnně označovaných jako Průmysl 4.0 zasáhne všechny aspekty společnosti a to včetně oblasti bezpečnosti, která je primárním předmětem zájmu našeho studia.

V budoucnu tak budeme pravděpodobně mít k dispozici výrazně větší množství dat popisující chování zájmových systémů nebo situace v zájmovém prostoru. To ale automaticky neznamena, že tato informace bude zpřístupněna všem osobám, které by z ní mohly profitovat - tedy např. bezpečnostním pracovníkům. Aplikace pro bezpečnost proto budou muset být teprve vyvinuty. Takový vývoj je však možný pouze v úzké spolupráci mezi výrobcí/dodavateli těchto systémů a specialisty na bezpečnost.

Technologie je pouze nástroj a my musíme najít efektivní způsob jak jej využít.

V posledních několika letech v souvislosti s masivním nástupem umělé inteligence do všech oblastí našeho života a také všech aspektů podnikové praxe, se začíná hovořit o *Průmyslu 5.0* a tedy s ní související páté průmyslové revoluci. Ta vychází logicky ze čtvrté, ale aplikuje do ní nové technologie a přístupy, kterými se výrazně odlišuje. Mezi základní principy řadíme obvykle:

- zaměření na člověka (personalizace výrobků a služeb)
- resilience - odolnost ve smyslu schopnosti podniku/systému předvídat, reagovat a přizpůsobovat se flexibilně různým interním a externím vlivům
- udržitelnost - ve smyslu environmentální odpovědnosti

Z pohledu používaných technologií se Průmysl 5.0 zaměřuje na:

- kolaborativní roboty (koboty)
- digitální dvojčata a umělou inteligenci
- rozšířenou realitu
- apod.

Je zajímavé, že Průmysl 4.0 a 5.0 je implementován téměř současně. Tzn., že ne všechny podniky začaly vůbec s přechodem k Průmyslu 4.0 natož 5.0, jsou ale vysoce progresivní podniky, které tyto technologie Průmyslu 5.0 již úspěšně nasazují, např. Amazon, ale také řada dalších.

Obecně lze říci technologie Průmyslu 4.0 jsou k dispozici téměř v „krabicové“ formě - tedy zájemce ji může pořídit a rutinně nasadit pro řešení svých potřeb. V případě Průmyslu 5.0 se stále ještě jedná spíše o pilotní projekty které jsou dělané více-méně na míru. Implementace je tedy spíše v počátcích.

V současnosti se dokonce začíná hovořit o *Průmyslu 6.0*. Zde řadíme technologie jako jsou kvantové výpočty, integrace člověka a stroje (bio-integrace) a řada dalších z dnešního pohledu futuristických technologií. Zatímco technologie Průmyslu 4.0 a 5.0 jsou již prakticky nasazovány, Průmysl 6.0 představuje spíše vizi budoucna, možná dokonce relativně vzdálené budoucnosti. Technologie, se kterými

pracuje tak ještě neexistují ve formě, která by nás opravňovala zatím vůbec přemýšlet o tom, že by se prakticky nasazovaly.



Shrnutí

V této kapitole jsme se seznámili s řadou různých zařízení, které jsou využívány pro automatizaci technologických procesů. **PLC** slouží pro automatizované řízení technologie, na základě předem připraveného programu. Podstatou takového řízení jsou změny v nastavení regulačních prvků na základě údajů ze senzorů popisující stav řízené technologie a následné kontroly vývoje hodnot senzory monitorovaných veličin - zda regulace vede k cíli.

Ne vždy je možno technologii uregulovat čistě automaticky, proto zejména pro složitější systémy se používá systém SCADA, který umožňuje stav řízené technologie v člověkem snadno pochopitelné podobě vizualizovat. Operátor/dispečer pak může případný problém v technologii identifikovat a manuálně jej korigovat.

Vzhledem k tomu, že řízení je většinou centralizované a často pro geograficky vzdálené, je nutné řešit způsob, jakým může být řízení realizováno dálkově. Vzdálené připojení lze realizovat pomocí běžně dostupných síťových technologií (které nebyly probírány v této kapitole :-), v automatizaci se ale využívá také zařízení **RTU** umožňující řízení připojené technologie v čistě textovém režimu, pomocí „terminálu“.

Ovládání systému může ale nabývat různých podob. Všechny systémy umožňující člověku manipulovat se systémem souhrnně označujeme jako **HMI** - rozhraní člověk - stroj.



Kontrolní otázky

1. Co je automatizace a k čemu slouží?
2. Vysvětlíte regulační smyčku.
3. Co je PLC?
4. K čemu slouží SCADA systém?
5. Co je RTU?



Správné odpovědi

1. Jedná se o technologie a postupy umožňující automaticky řídit systém, tak aby jeho chod byl udržován v určitých (předem stanovených parametrech).
2. Regulace je prováděna na základě snímání veličin ze senzorů instalovaných na řízené technologii, jejich vyhodnocování a automatizovaných změnách v nastavení provozu technologie. Senzory jsou zpětně sledovány pro zjištění zda regulace konverguje k cíli.
3. Zjednodušené odpovědi na otázky 3 - 5 naleznete také ve shrnutí kapitoly.

Kapitola 2

Real-timové databáze



Náhled kapitoly

V této kapitole navážeme na kapitolu předchozí a zaměříme se především na funkci databáze umožňující shromáždit a spravovat informace o stavu technologie na jednom místě - v databázi. Na databáze plníci tuto úlohu jsou ale kladeny odlišné nároky, proto hovožíme o tzv. *realtimových databázích*.

Po přečtení této kapitoly budete vědět

1. čím se liší databáze pracující v reálném čase od běžně používaných databází
2. jaké nástroje a strategie lze použít pro zrychlení databázových operací

Získáte

1. základní znalosti o funkci SQL jazyka



Čas pro studium

Pro prostudování této kapitoly budete potřebovat přibližně hodinu.

2.1 Real-time databáze vs relační databáze

V minulé kapitole jsme se seznámili se základní funkcí některých zařízení pro automatizaci technologií. Nutnost řídit technologii nám ale vyvolala jeden poměrně závažný problém - *efektivní řízení je možné pouze pokud je prováděno na základě aktuálních údajů*. Tedy řízení může uspět pouze pokud víme v jakém stavu se nachází technologie teď, v okamžiku řízení.

Jak ale takový požadavek zajistit? Požadavek očividně vyvíjí tlak na to, aby údaje ze senzorů byly ukládány co možná nejdříve od okamžiku jejich vzniku (pořízení) tak, aby řídicí systém s nimi mohl dále pracovat. Existuje řada možností, jak toto zajistit - řídicí systém může třeba pracovat se senzory přímo, ale takové řešení obvykle není dobře škálovatelné a tak obvykle přistupujeme k takovým údajům spíše zprostředkovaně. Data ze senzorů jsou v takovém případě konsolidována na jednom místě, které bývá označováno jako třeba *data historian*.

Data historian může být organizován různě - může se jednat o „ploché“ textové soubory obsahující data v tabelární formě (např. ve formátu CSV), pro složitější technologie se ale častěji používají databáze. Databáze jsou rutinně využívány většinou informačních systémů pro ukládání dat, která tyto systémy spravují. S provozem databázových systémů jsou tak rozsáhlé zkušenosti. Oproti požadavkům, které jsou ale kladeny na ukládání dat v běžných, provozních informačních systémech, klademe na zpracování dat pro účely řízení technologií poněkud jiné požadavky. Zkusme tyto požadavky porovnat.

Požadavky běžného informačního systému

1. všechny údaje musí být uloženy do databáze (údaje nelze vynechat)
2. hodnota dat v čase je konstantní (neklesá v čase)
3. vysoká rychlost uložení je žádoucí, nikoliv však nezbytně nutná

Požadavky na data historiana (real-timová databáze)

1. hodnota dat v čase klesá
2. ne všechny data je proto nutné ukládat, byť zajištění co možná největší kompletnosti dat v databázi je žádoucí
3. vysoká rychlost manipulace s daty je nutnost

Požadavky kladené na databázi pro nasazení v řídicích systémech jsou tedy poměrně odlišné, což vede k tomu, že někdy databáze pro automatizaci jsou označovány jako reálné, tedy databáze pracující s daty v reálném čase.

Pokud se ale nad tím zamyslíme je samotný název *reálná databáze* více méně protimluv. Každý proces měření a následného ukládání dat vyžaduje určitý čas. Tím v celém procesu nevyhnutelně vznikají různé prodlevy, latence. V zájmu efektivity řízení, je, aby tyto prodlevy byly co možná nejmenší - odtud tedy tlak na zpracování v reálném čase. Existence latence ale způsobuje, že zpracování dat databází nebude nikdy ve zcela reálném čase, maximálně se můžeme přiblížit k něčemu, co můžeme označit jako *skoro reálný čas* zpracování.

Skoro reálný čas znamená, že data nejsou sice zpracována v reálném čase, prodlevy a latence v systému jsou ale natolik malé, že z pohledu efektivity řízení je můžeme zanedbat. To co je a není zanedbatelné se přitom bude lišit technologie od technologie.

Technicky jsou reálné databáze, alespoň v současnosti, *databázemi relačními*. Otázka tedy je vzhledem k tomu, že potřebujeme zajistit zpracování dat v co možná nejkratším čase, jak postupovat, abychom práci databáze urychlili? Metod, které k tomuto účelu lze použít je velké množství: volbou vhodné databáze a její správné nastavení, investicí do výkonnějšího hardware, na kterém bude databáze provozována, apod.

Postup je vždy stejný. V každém systému existují určitá úzká hrdla, která je následně možné řešit, aby mohlo být dosaženo vyššího výkonu. Z hlediska hardware může být problém v:

- šířce použité přenosové cesty
- rychlost operací **Input-Output (IO)**
- rychlost a kapacita paměti
- počet procesorů a jejich vlastnosti (např. počet jader procesoru, provozní frekvence atd.)
- apod.

V případě software pak může být problém ve:

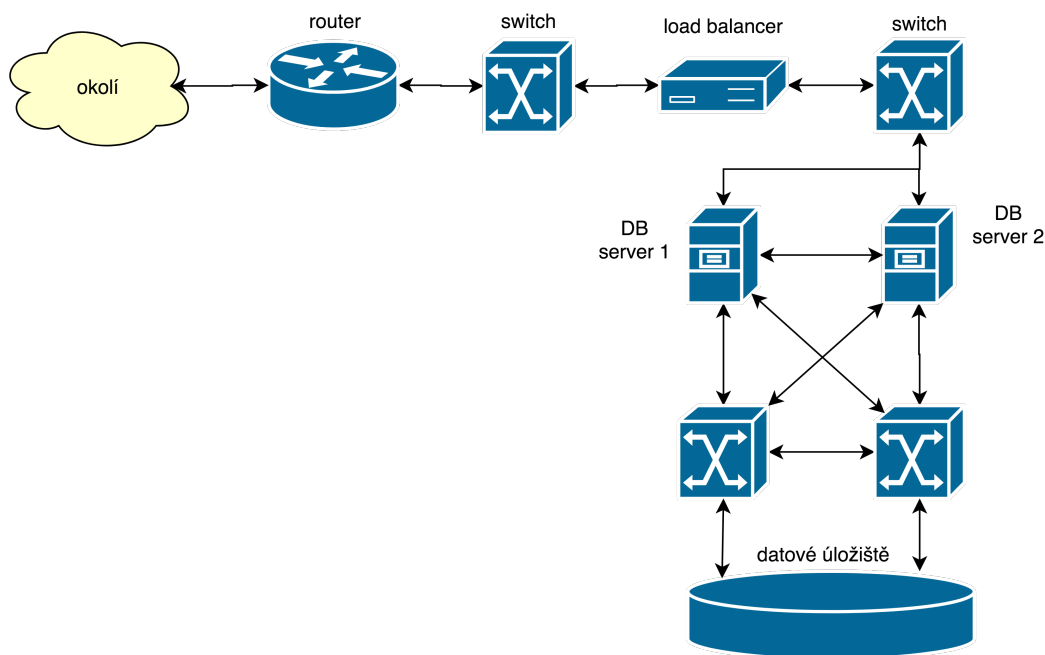
- volbě/nastavení operačního systému (např. volba souborového systému)
- volbě/nastavení databázového systému
- volbě rozhraní, které bude využíváno pro manipulaci s databází (např. **Open Database Connect (ODBC)** nebo **Java Database Connect (JDBC)**)
- apod.

Výkonnost systému manipulace s daty je tedy závislá na kombinaci řady faktorů - neexistuje jedno „univerzálně správné řešení“, které by bylo možné aplikovat na všechny situace bez rozdílu.

Z hlediska možných řešení ve výše uvedeném seznamu není obsažena možnost škálování výkonu pomocí použití tzv. *databázového klastru*. Hlavní myšlenka tohoto řešení je ta, že možnost navyšování výkonu databázového serveru jako jednoho zařízení je omezená. Je možno zvětšit paměť, použít výkonnější procesor, nebo přidat další procesor, pokud to v serveru použitý motherboard podporuje apod. Ve finále jsme ale vždy limitováni technickými možnostmi rozšiřitelnosti daného systému a ty jsou vždy omezené.

Pokud ale nestačí jeden server, může být výhodnější použít servery dva, nebo tři nebo ještě více. Takové nasazení je obvykle realizováno pomocí databázového klastru. Vizualně bychom si mohli takový kluster mohli představit podobně jako na obr. 2.1.

Klíčovou roli v klastru hraje tzv. *load balancer*. Jedná se o zařízení, které kontroluje rozložení zátěže serverů, v tomto případě tedy serverů databázových. Práce na jednotlivé servery je přidělována podle aktuálního zatížení serverů tomu nejméně zatíženému. Účelem rozkládání zátěže, je zajistit, že všechny servery v klastru jsou využívány rovnoměrně.



Obrázek 2.1: Databázový klaster

To ale znamená, že databázové operace nejsou prováděny zároveň na všech databázových serverech v klastru, ale pouze na jedné z nich a na ostatní se pak změna zaznamenává později procesem *replikace*.

Load balancer představuje určitou úroveň abstrakce databázové architektury. Externí zařízení vnímají celý klaster jako jedno zařízení - nevidí do vnitřní organizace klastru. To nám dává poměrně široké možnosti pro škálování výkonu, ale také přípravu na případná selhání jednotlivých komponent klastru. V takovém případě obvykle také zároveň řešíme vzájemnou zastupitelnost uzlů v klastru v případě výpadku což na jedné straně v běžném provozu zvyšuje celkový výkon klastru, na druhé straně v případě selhání komponent umožňuje postupně degradovat výkon na místo totálního výpadku služby v tomto případě databázového systému.

Synchronizace dat napříč databázemi pak probíhá procesem nazývaným *replikace dat* s tím, že replikovány jsou pouze změny v uložených datech. Tímto způsobem se minimalizují požadavky na datové přenosy mezi jednotlivými uzly klastru.

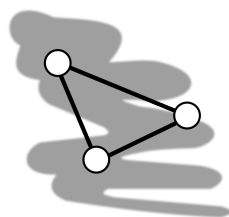
To, že jednotlivé servery v klastru neobsahují zcela stejná data představuje určitou komplikaci z hlediska čtení. V našem případě je o to závažnější, že pro účely řízení vyžadujeme primárně poslední/aktuální data o řízeném systému a právě ta nemusí být nutně dostupná na všech serverech v klastru.

Pro tento problém existují určitá technická řešení, která ale z pohledu podrobnosti přesahují naše možnosti v těchto skriptech.

Kromě možnosti navýšení výkonu je také výhodou vzájemná zastupitelnost těchto zařízení i v případě výpadku. Např. pokud v klastru selže jeden z databázových serverů, load balancer rozdělí jeho zátěž na zbývající servery. Řešení pomocí klastru je tak z mnoha pohledů odolnější (ne jen výkonnější). Realizace klastru je jednou z metod jak zvýšit resilienci provozovaného systému.

Všimněte si také, že na obr. 2.1 data nejsou uložena přímo na databázovém serveru, ale na připojeném datovém úložišti. Takové úložiště bývá realizováno často pomocí diskového pole. Diskové pole lze pak poměrně jednoduše škálovat z pohledu požadovaného výkonu, ale také odolnosti proti hardwarovému selhání jednotlivých disků.

Rozdíl mezi reálním a běžnou databází je taktéž v samotné organizaci dat. K tomu ale abychom plně ocenili důsledky těchto rozdílů, je potřeba pochopit způsob, jakým jsou v databázích realizovány tzv. *transakce*.



Klastry, souvislosti

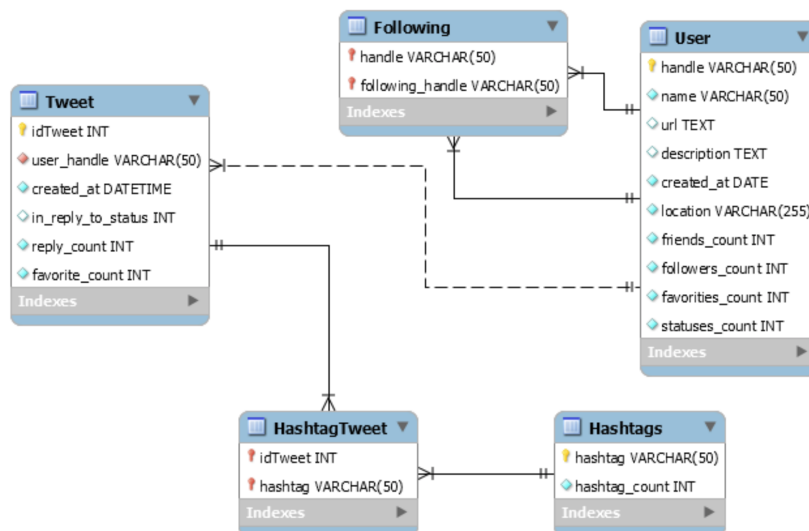
Architektura představená na obr. 2.1 je použitelná univerzálně pro různé typy serverů. Podobným způsobem tak lze škálovat např. servery webové a nejen je.

2.2 Transakce

2.2.1 Organizace dat v databázi

Relační databáze mají dva konstruktory - *tabulky*, kde jsou shromažďována data, a *relace* definující vazby mezi jednotlivými tabulkami. Vzhledem k tomu, že z předchozí kapitoly už víme, že pro účely automatizace potřebujeme velmi specifická data, je otázka, zda tato specifická se projeví také na struktuře dat ukládaných v databázi a pokud ano, tak jakým způsobem?

Zkusme vizualizovat základní údaje udržované sociální sítí Twitter, viz obr. 2.2. Vizualizaci prosím berte jako ilustrativní. Je zjednodušená, tak aby obsahovala pouze metainformace o tweetech, nikoliv text tweetu, případně další média s ním spojená.



Obrázek 2.2: ERD diagram metadat příspěvků v síti Twitter (adaptováno z [81])

Databáze z obr. 2.2 by mohla být použita např. pro analýzu vazeb mezi jednotlivými uživateli. Na obr. hrají hlavní roli tabulky User a Tweet, vše ostatní se pak vztahuje k nim. Propojení tabulek se děje pomocí relací. Úkolem relace je specifikovat způsob, jakým spolu souvisí tabulky na obou stranách definovaných relací.

Relace mezi tabulkami Tweet a User je definovaná přes položky user handle z tabulky Tweet a handle z tabulky User. Získávání dat z databáze se pak děje právě přes tyto související sloupce tabulek - propojení je tak obvykle realizováno tam, kde jsou hodnoty souvisejících polí stejné.

Relace tedy umožňují vytvářet komplexní 2D struktury v datech, a to v okamžiku kdy to potřebujeme.

Jaké informace ale do databáze ukládá automatizovaná technologie? To bude samozřejmě záviset na charakteru technologie a zvolených senzorech, které ji monitorují. V obecné rovině budou ale data následujícího charakteru:

- údaj ze senzoru (může se jednat o text, číslo, nebo třeba binární data, jako jsou fotografie)
- údaj o čase, kdy bylo provedeno měření
- případně údaj o senzoru jako takovém (např. identifikátor)

Lze uvažovat o různých přístupech. Jednotlivým sensorům můžeme např. vytvořit samostatné tabulky. Struktura takové tabulky by mohla být následující: Sensor1(DatumAČas, hodnota). Jméno

tabulky pak identifikuje senzor, ze kterého bylo pořízeno měření.

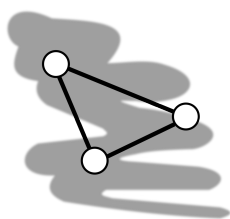
Jinou možností by bylo zobecnění výše uvedeného řešení pro větší množství senzorů měřících stejnou veličinu. Struktura tabulky by mohla vypadat následovně: `SenzoryTeploty(DatumAČas, teplota, IDSenzoru)`. K identifikaci senzoru pak slouží sloupec `IDSenzoru`. K `IDSenzoru` lze také případně vést relaci k další tabulce se servisními informacemi k senzoru, popř. dalšími informacemi nutnými k řízení.

Třetí možností je konstrukce tabulky: `Hodnoty(DatumAČas, HodnotaSenzor1, HodnotaSenzor2, ..., HodnotaSenzorN)`. V této konfiguraci jsou hodnoty senzorů evidovány samostatně v jednotlivých sloupcích tabulky. Tato konfigurace ale předpokládá, že interval měření bude konstantní a univerzální pro všechny v tabulce zavedené senzory.

Z pohledu databázové teorie je optimální pouze druhé řešení a např. řešení třetí nerespektuje ani základní požadavky na konzistenci dat formulované do tzv. *první normální formy*. Ta říká, že tabulka by neměla obsahovat opakující se sloupce, což není v posledním případě splněno, protože účel všech sloupců `HodnotaSenzorX` je stejný - evidovat hodnotu určitého senzoru.

Jelikož je ale použití databáze pro účely řízení v reálném čase odlišné, mohou být všechny výše uvedené možnosti (a řada dalších) přijatelné. Oproti použití databáze jako backendu běžného informačního systému lze předpokládat pouze minimální zásahy do struktury tabulek. Ta tak zůstane neměnná prakticky po celou dobu provozu řízené technologie. Z tohoto důvodu nemusí fakt, že databáze není ani v první normální formě, představovat závažnější problém.

Uvedená struktura databáze má také pozitivní dopady na výkon databázového serveru. Největší zátěž databázového systému totiž obvykle představují operace výběru dat realizované napříč tabulkami. My ale už víme, že v real-timeových databázích preferujeme spíše samostatné tabulky, proto tento typ dotazů není potřeba realizovat. Dotazy realizované nad jednotlivými tabulkami jsou pak velmi rychlé.



Databáze - návaznosti

Diskuze o databázích je užitečná nejen pro pochopení fungování automatizačních systémů, ale také fungování běžných informačních systémů a možnosti propojování a analýzy dat. Vhodné spojení dat z různých zdrojů představuje první krok v procesu dolování dat, kterým se budeme zabývat v druhé polovině skriptu.

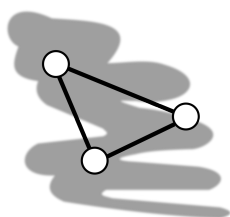
2.2.2 Jazyk SQL

Jakým způsobem lze s databází manipulovat? Relační databáze pracují s jazykem **SQL**. Příkazy (dotazy) jazyka lze rozdělit podle toho, co dělají:

- Dotazy definující nebo měnící strukturu databáze (`CREATE TABLE ...`, `ALTER TABLE ...`)
- Dotazy výběrové (`SELECT FROM ...`)
- Dotazy modifikující data (`INSERT INTO ...`, `UPDATE ... WHERE ...`, `DELETE ...`)
- Dotazy pracující s tzv. programátorskými objekty jako jsou pohledy, uložené procedury a další (v těchto skriptech se tomuto typu dotazů nebudeme věnovat)

Definice struktury tabulek

Účelem této podkapitoly je dozvědět se něco o jazyku SQL, propojování dat a podobných záležitostech. jednotlivé typy příkazů budeme demonstrovat na tabulkách `Tweet` a `HashtagTweet` z obr. 2.2. Nejedná se sice o příklad související s automatizací, ale jednodušeji si jej prakticky představíte a principy zůstávají stejné.



Databáze - základy

Pokud se v následujícím textu nebudete „chytat“, doporučuji projít první kapitulu skriptu *Bezpečnostní informatika 1 - Návod do cvičení* [91], kde je rozebrán úvod do databázové teorie, problematika normalizace báze dat a způsob odvození báze dat s demonstrací pro databáze MS Access a LibreOffice Base. Zatímco v BI definice tabulek probíhala naklikáním, v tomto případě je vytváříme přímo pomocí SQL dotazu.

Výpis 2.1: SQL: definice tabulek - příklad Twitter

```

1 CREATE TABLE Tweet (
2   idTweet INT NOT NULL AUTOINCREMENT,
3   user_handle VARCHAR(50) NOT NULL,
4   created_at DATETIME NOT NULL,
5   in_reply_to INT,
6   reply_count INT,
7   favorite_count INT,
8   PRIMARY KEY (idTweet)
9 );
10
11 CREATE TABLE HashtagTweet (
12   idTweet INT NOT NULL,
13   hashtag VARCHAR(50),
14   PRIMARY KEY (idTweet, hashtag)
15 );

```

Příkaz SQL pro vytvoření tulký začíná vždy klíčovými slovy CREATE TABLE jméno tabulky. Do kulatých závorek jsou pak vkládány datové definice jednotlivých sloupců ve formátu jméno sloupce, datový typ, případné další modifikátory.

Např. sloupec idTweet je celočíselného datového typu (INT), musí být vyplněn (NOT NULL) a vyplnění bude probíhat automaticky (AUTOINCREMENT).

V příkazu CREATE je možno také specifikovat primární klíč, který jednoznačně bude identifikovat každý jeden řádek dané tabulky. V případě tabulky Tweet se jedná o sloupec idTweet, v případě tabulky HashtagTweet je pak použit složený primární klíč - z obou sloupců tabulky.

Datových typů databáze podporují celou řadu, my jsme použili INT pro celé číslo, VARCHAR(délka řetězce) pro textové řetězce o maximální zadané délce, maximální délka je specifikována v závorce, DATETIME - datum a čas. Populární jsou také datové typy DOUBLE pro desetinná čísla, TEXT pro dlouhý text (nad 255 znaků), BLOB pro ukládání binárních dat v databázi.

Databáze od různých výrobců se mohou používanými datovými typy drobně lišit, berte proto prosím výše uvedený přehled jako orientační.

Pro úplnost uvádím také obecnou strukturu CREATE SQL příkazu.

Výpis 2.2: SQL: struktura příkazu CREATE TABLE

```

1 CREATE TABLE Jmeno_tabulky (
2   jmenoSloupce datovyTyp dalsiModifikatory,
3   dalsiJmenoSloupce ...,
4   PRIMARY KEY (vsechny sloupce definujici klic oddelene carkami)
5 );

```

Již vytvořené je možno změnit pomocí SQL příkazu ALTER TABLE.

Výpis 2.3: SQL: struktura příkazu ALTER TABLE

```

1 ALTER TABLE Jmeno_tabulky
2   ADD jmenoSloupce datovyTyp modifikatory,
3   DROP COLUMN jmenoSloupce,
4   ALTER COLUMN jmenoSloupce datovyTyp;

```

Do tabulky i po vytvoření tak lze doplňovat nové sloupce (ADD), nebo je odebírat (DROP COLUMN). Každá databáze pak umožňuje také modifikovat definici již existujících sloupců. Různé databáze ale mohou využívat k tomuto účelu odlišná klíčová slova. ALTER COLUMN využívá např. MS Access nebo MS SQL Server, databáze MySQL ale používá klíčové slovo MODIFY COLUMN.

Datové typy odpovídají datovým typům popsáním v části věnované vytváření tabulek.

Výběrové dotazy

Výběry jsou realizovány pomocí příkazu SELECT. Zkusme vybrat jeden záznam z tabulky Tweet - např. tweet s ID 25

Výpis 2.4: SQL: výběrový dotaz - příklad výběru konkrétního příspěvku

```
1 SELECT * FROM Tweet WHERE idTweet = 25;
```

Všechny Tweety uživatele @uzivatel:

Výpis 2.5: SQL: výběrový dotaz - příklad výběru příspěvků určitého uživatele

```
1 SELECT * FROM Tweet WHERE user_handle LIKE '@uzivatel';
```

Ve výše uvedených příkladech je postup stále stejný: SELECT sloupce FROM jménoTabulky WHERE podmínka. V předchozích příkladech jsme specifikovali, že chceme vybrat všechny sloupce pomocí zástupného znaku *. V případě, že bychom chtěli ale vybrat pouze některé ze sloupců, bylo by potřeba místo * napsat jejich seznam oddělený čárkami.

Výběr v obou předchozích příkladech byl omezen pomocí operátory porovnání. V případě idTweet byl použit operátor =, protože idTweet je celočíselného datového typu (INT). Podobným způsobem bychom pro čísla mohli použít operátory >, >=, <, <=.

V případě porovnání proti textu je ale využíván operátor LIKE, hledaný textový řetězec je ohraničen 'znakem apostrofu'. Pokud nehledáme přesnou shodu, lze použít znak %, např. LIKE 'ABC%' vyhledá všechny řetězce (v daném sloupci) začínající na ABC.

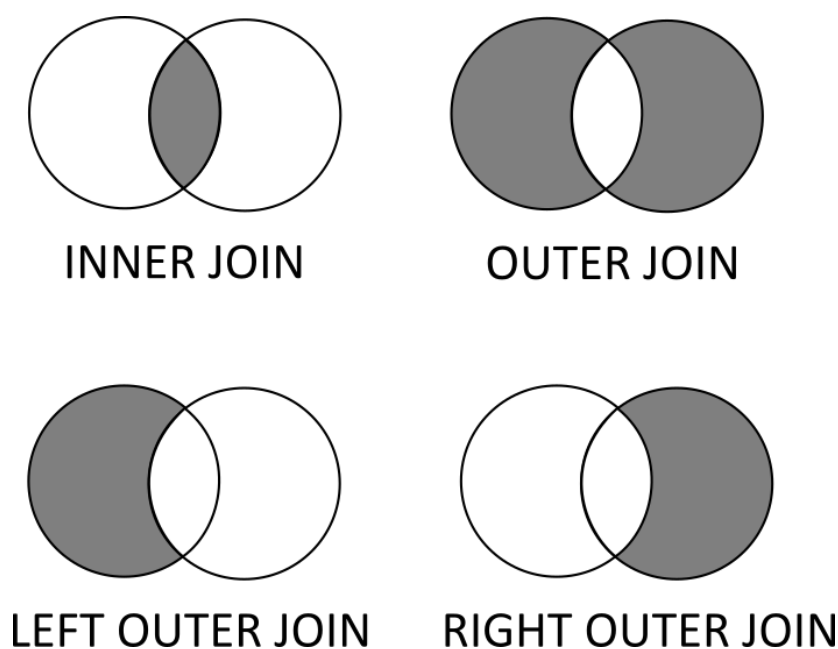
Podmínky pak lze řetězit kombinovat pomocí logických spojek AND a OR.

Výběry lze realizovat také napříč tabulkami. Řekněme, že bychom chtěli vybrat všechny tweety s haštagem #vyber. Řekněme, že nás ale budou zajímat pouze sloupce user handle a created at:

Výpis 2.6: SQL: výběrový dotaz - propojování tabulek vnitřním spojením (INNER JOIN)

```
1 SELECT
2     Tweet.user_handle,
3     Tweet.created_at
4 FROM Tweet
5 INNER JOIN HashtagTweet ON HashtagTweet.idTweet = Tweet.idTweet
6 WHERE hashtag LIKE '#vyber';
```

Spojení tabulek je realizováno pomocí INNER JOIN - tedy vnitřního spojení. Tímto spojením se bude vybírat průnik obou tabulek. INNER JOIN ale není jedinou možností propojení. Výsledek různých spojení je v grafické podobě dostupný na obr. 2.3.



Obrázek 2.3: Výsledky spojení dvou tabulek

Kromě vnějších (OUTER) spojení, jsou používány také prosté levé (LEFT JOIN) a pravé (RIGHT JOIN). Tato spojení se liší od spojení z obr. 2.3 tím, že ve výběru je obsažen také průnik obou tabulek.

Tímto způsobem lze spojovat libovolné množství tabulek v databázi. Je ale potřeba mít na paměti, že se vzrůstajícím počtem spojovaných tabulek rostou nároky na spotřebovávané zdroje databázového serveru a to výrazně.

Aktualizace tabulek

Aktualizací tabulky rozumíme obvykle vložení nových řádků do tabulky (INSERT), aktualizaci řádků v tabulce (UPDATE), nebo jejich výmaz (DELETE).

Vložení nového řádku do tabulky Tweet by mohlo vypadat následovně:

Výpis 2.7: SQL: příklad vkládání údajů do tabulek (INSERT INTO)

```
1 INSERT INTO Tweet (user_handle, created_at)
2 VALUES
3 ('@uzivatel', '2018-08-08 08:15:22'),
4 ('@uzivatel', '2018-08-08 09:01:34');
```

Všimněte si, že není nutné vždy zadávat všechny sloupce tabulky, pouze ty, jejichž zadání je povinné (jsou definované s příznakem NOT NULL). V tabulce Tweet jsou ale povinné 3 sloupce, tím třetím je idTweet, přesto jej ve výše uvedeném příkazu nenaleznete. Tento sloupec je totiž nadefinován jako automatické číslo, databáze jej proto vyplní sama, neočekává zadání uživatelem.

Najednou lze do tabulky vložit také více řádků. Jednotlivé řádky jsou vkládány za klíčové slovo VALUES do kulatých závorek. Jednotlivé hodnoty sloupců jsou odděleny čárkami. Jednotlivé řádky jsou taktéž oddělovány čárkami.

Výše uvedený příklad bychom mohli zobecnit:

Výpis 2.8: SQL: struktura příkazu INSERT INTO

```
1 INSERT INTO jmenoTabulky (sloupec1, sloupec2, ...)
2 VALUES
3 (hodnotyRadek1),
4 ...,
5 (hodnotyRadekN);
```

Aktualizace je možno provádět nad celou tabulkou nebo její částí specifikovanou klauzulí WHERE.

Výpis 2.9: SQL: příklad aktualizace záznamu v databázi (UPDATE)

```
1 UPDATE Tweet SET created_at = '2018-08-08 05:00:00';
2
3 UPDATE
4   Tweet
5 SET favorite_count = 0
6 WHERE user_handle LIKE '@uzivatel';
```

První příklad nastaví datum a čas vytvoření tweetu na 8. 8. 2018 5 hodin ráno. V druhém příkladu bude pro všechny tweety uživatele @uzivatel nastaven počet lajků na 0.

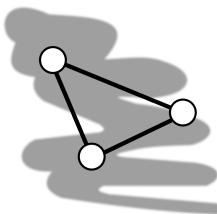
V případě, že je potřeba aktualizovat více než jeden sloupec, zadávají se všechny do klauzule SET a oddělují se čárkou.

Výmaz z tabulky se provádí pomocí příkazu DELETE. Pokud výmaz není omezen klauzulí WHERE, provede se výmaz obsahu celé tabulky (tabulka samotná zůstane, ale bude prázdná).

Výpis 2.10: SQL: příklad výmazu záznamů z databáze (DELETE)

```
1 DELETE FROM Tweet;
2
3 DELETE FROM Tweet WHERE user_handle LIKE '@uzivatel';
```

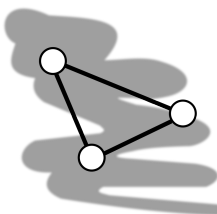
První příklad smaže obsah tabulky Tweet. Příklad druhý pak smaže pouze ty řádky tabulky Tweet, kde je autorem @uzivatel.



Jazyk SQL - další studium

V této podkapitole jsme se seznámili pouze se základy jazyka SQL. Na Internetu existuje celá řada výborně zpracovaných materiálů pro Vaše případné další studium. Velmi dobře je např. zpracovaný SQL Tutorial na w3schools.com [84].

Vhodným zdrojem informací je pak také manuál Vámi zvoleného databázového systému.



SQL - Proč

Pokud jsme si před chvílí řekli, že řídicí systémy používají především samostatné tabulky, proč bychom se vůbec pokročilým dotazováním měli zabývat? Čtě z akademického pohledu Vám to neuškodí :-), ale to jste pravděpodobně nechtěli slyšet.

Tak tedy znovu. Pokročilé dotazování je ve skutečnosti jeden z nejuniverzálnějších použitelných dovedností, kterou využijete všude tam, kde potřebujete maniplovat s daty, ve smyslu výběru, spojování dat napříč tabulkami nebo různými zdroji dat, změna struktury dat apod.

V kontextu této kapitoly jsme zašli trochu dál, než jsme potřebovali - pro účely řídicích systémů. Pro účely dolování dat, které nás ale čekají později je to znalost, kterou zužitkujeme. Informace týkající se databází jsme tak shromáždili na jednom - protože spolu souvisí.

2.2.3 Transakce

Konečně se dostáváme k transakci - *co je tedy transakce?* **Transakce je posloupnost souvisejících databázových operací.**

Výše uvedená definice je velmi stručná, na druhou stranu v sobě obsahuje relativně velké množství prvků, které neznalé mohou zmást. Takže zkusme definici rozebrat. Začneme nejprve databázovou operací - tu můžeme zjednodušeně vnímat jako jeden (jakýkoliv) příkaz jazyka SQL z předchozí podkapitoly.

Posloupností pak rozumíme fakt, že příkazy jsou vykonávány postupně v jistém určeném pořadí - operace v jedné transakci tak nemohou probíhat paralelně. Předpokládáme totiž, že databázové operace v rámci jedné transakce spolu souvisí, popř. na sebe navazují. Musí proto proběhnout sekvenčně (postupně). Na druhou stranu paralelně může databáze zpracovávat více (nezávislých) transakcí.

Účelem transakce je tedy transformovat databázi z jednoho konzistentního stavu do druhého. Jako si to představit? Řekněme, že potřebujeme v databázi z obr. 2.2 změnit identifikátor uživatele (sloupec handle). Mohli bychom postupovat tak, že aktualizujeme pomocí příkazu UPDATE tabulku User a handle přenastavíme.

To ale nestačí - identifikátor uživatele je totiž použit také v tabulkách Following (sloupec handle) a Tweet (sloupec user_handle). Pokud nepřenastavíme také je, údaje v těchto tabulkách se „rozpoj“ a tím se naruší integrita databáze.

Správný postup je pak takový že musíme aktualizovat handle ve všech třech tabulkách a to najednou, v rámci jedné transakce. Přitom potřebujeme zajistit, aby aktualizace proběhla ve všech těchto tabulkách nebo v žádné. Co to znamená? Aktualizaci v tomto případě provedeme pomocí tří příkazů UPDATE. Jenomže, co když některý z těchto příkazů z nějakého důvodu selže? Takových důvodů může být celá řada - mohli jsme udělat chybu v příkazu (např. překlep), nebo zrovna v okamžiku provádění příkazu byl problém na databázovém serveru, což zabránilo dokončení transakce v daném čase. Pokud bychom přijali pouze částečnou aktualizaci databáze narušili bychom její konzistenci a to obvykle nechceme.

Proto potřebujeme, aby se tyto operace provedly celé - pokud se to nepovede, potřebujeme mít schopnost odvolat změny v databázi a vrátit ji do stavu jako předtím, než jsme začali s úpravami. Efektivní cestou, jak toto zajistit je zařadit takto související příkazy do jediné transakce.

Transakce - základní požadavky

Základní požadavky, které jsou kladeny na databázi z hlediska provádění transakcí se skrývají pod zkratkou ACID (atomicita - konzistence - izolovanost - nevratnost).

Atomicitou rozumíme, že transakce probíhá jako celek, je tedy nedělitelná. Prakticky to znamená, že transakce proběhne úspěšně celá, anebo neproběhne vůbec. Jinými slovy, v případě, že některá operace v rámci transakce selže, předchozí, úspěšně provedené operace v rámci transakce jsou revokovány (pomocí příkazu ROLLBACK) a databáze se dostane do stavu jako před započítím transakce.

V případě, že transakce proběhne úspěšně celá potvrdí se transakce pomocí příkazu COMMIT.

Konzistence zaručuje, že databázový systém je transakcemi transformován z jednoho konzistentního stavu do druhého, opět konzistentního, stavu. Tato vlastnost souvisí s úplným provedením transakce. Nekonzistence jsou do databáze vnášeny buď chybným návrhem databáze, tento problém však transakce neřeší, nebo provedením pouze částečných změn (jako třeba pouze polovina požadovaných operací). Protože transakce se provede celá nebo vůbec.

Transakce tedy efektivně brání problémům s konzistencí databáze.

Izolovanost – transakce je izolována od zbytku databáze. Ostatním operacím nebo transakcím nejsou dostupné výsledky operací prováděných v rámci transakce do doby úplného dokončení této transakce.

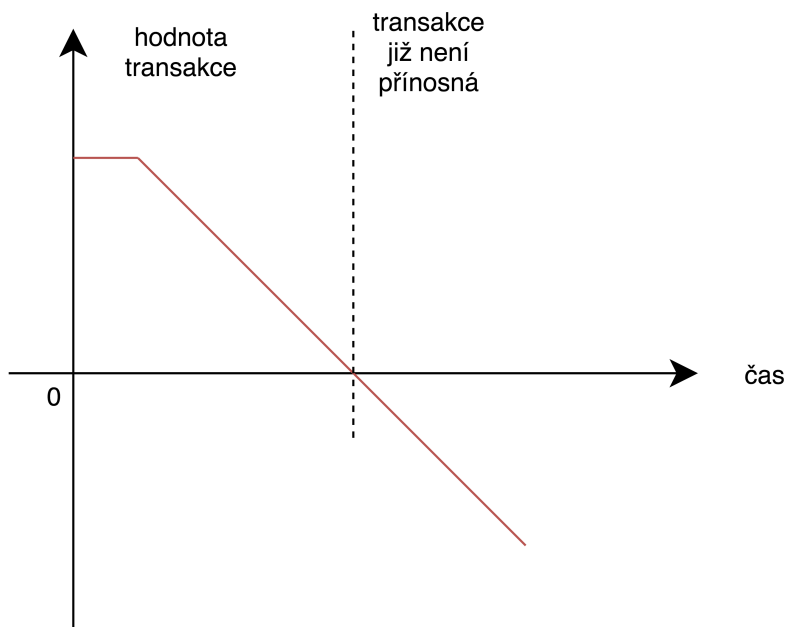
Nevratnost – pokud transakce byla úspěšně dokončena (příkazem COMMIT), není možné ji odvolat. Změny transakce je možné zvrátit pouze v průběhu transakce.

Pokud je potřeba po dokončení transakce odvolat změny, je nutné jít cestou mimo databázi – nejčastěji se používá obnova ze záloh.

Realtimové databáze k těmto principům přidávají ještě **časové omezení**. Z předchozí kapitoly již víme, že z hlediska automatizace mají největší význam aktuální informace, které popisují stav, ve kterém se nachází řízená technologie teď.

Pokud tedy má transakce mít maximální přínos, musí být dokončena v jistém, omezeném časovém intervalu. Tento interval obvykle nastavují tvůrci aplikací, které s těmito systémy pracují. Možná Vás napadne otázka, co s transakcemi, které neskončí včas. Důvodů, proč transakce trvá déle než je žádoucí přitom může být celá řada od technických – proces zachycený transakcí včas neskončil (stále trvá) až po to, že došlo k přetížení serveru, který pak nestačí vyřizovat transakce.

Přínos transakce graficky si lze představit podobně jako na obr. 2.4.



Obrázek 2.4: Změna hodnoty transakce v čase

Z tohoto pohledu je potřeba rozlišit přínos, jaký má dokončení této opožděné transakce:

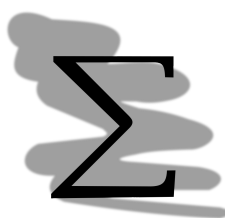
1. opožděné dokončení transakce je přínosné – hodnota přínosu se ale obvykle nerovná přínosu ze včas dokončené transakce (např. aktuální hodnota senzoru výrobní linky vs hodnota téhož senzoru z minulého dne). V takovém případě se databázový systém snaží transakci dokončit

- obvykle ale opožděné transakci přiřazuje nižší prioritu tak, aby nebyla ohrožena schopnost databáze vkládat aktuální, nově zadávaných (čerstvých) údajů.
- 2. V okamžiku, kdy transakce ztratí hodnotu úplně nebo její hodnota klesne pod určitou, předem stanovenou mez obvykle vyjádřenou časem od spuštění transakce v sekundách, může databázový systém její dokončení stornovat, tak aby si uvolnil systémové zdroje nutné pro vyřizování přínosných transakcí. Ne všechny typy transakcí ale lze stornovat.
- 3. Hodnota nedokončené transakce může dokonce poklesnout pod nulovou hodnotu – nedokončení transakce v takovém případě indukují další náklady. V takovém případě systém může buďto transakci stornovat (je-li to možné) nebo naopak navýšit prioritu výkonu transakce, tak aby transakce byla přednostně dokončena a nepřinášela další náklady.

Správné nastavení doby expirace jednotlivých transakcí je poměrně složitou záležitostí. Schopnost systému zpracovávat data je proto potřeba dlouhodobě sledovat a vyhodnocovat. Ke změně rychlosti práce s daty může docházet také v souvislosti se změnami v životě databázového serveru nebo řízené technologie.

Dopad na výkon mohou mít také bezpečnostní aktualizace operačního systému. Např. záplaty na chyby v procesorech společnosti Intel známé pod jmény Spectre [44] a Meltdown [49] mohou způsobit pokles rychlosti určitých operací až o 15 %, zejména na starších typech procesorů.

Nastavení a údržba databázového systému je záležitostí, které se musí dlouhodobě věnovat IT specialista.



Shrnutí

V této kapitole jsme se seznámili se způsobem fungování relačních databází. Pro účely řízení se využívají databáze přizpůsobené co možná nejrychlejší odezvě – tyto označujeme jako *realtimové*. Tyto databáze jsou sice stále relační, využívají ale některé postupy pro zkrácení doby zpracování.

V úvahu takové databáze berou především hodnotu zpracovávaných dat. To je také největší rozdíl proti databázím používaných v běžných informačních systémech, kde hodnota dat je v čase stálá. Oproti tomu u realtimových databází hodnota zpracovávaných dat v čase klesá. Pro databázi tak může být výhodné rezignovat na nejstarší nezpracované transakce a uspořádaný strojový čas pak může být využit pro vyřízení transakcí aktuálních.



Kontrolní otázky

1. Čím se liší realtimová a relační databáze?
2. Jakými SQL příkazy lze aktualizovat data v tabulce?
3. Co rozumíme atomicitou transakce?
4. K čemu lze použít údaj o době zpracování transakce?
5. Jak se liší INNER JOIN spojení a LEFT JOIN?



Správné odpovědi

1. viz shrnutí.
2. INSERT, UPDATE, DELETE.
3. Transakce se provede celá nebo se neprovede vůbec (musí úspěšně proběhnout celá).
4. Staré nezpracované transakce, jejichž přínos zpracování je minimální, mohou být zahozeny aby se uvolnil výkon databázového systému pro zpracování transakcí nových.
5. INNER JOIN vybírá data tvořící průnik spojovaných tabulek, LEFT JOIN pak obsahuje všechna data z levé tabulky a data z pravé tabulky obsažená v průniku obou tabulek.

Kapitola 3

SCADA - příklad v Promotic



Náhled kapitoly

V této kapitole se seznámíme se systémem Promotic [54] Ostravské společnosti MicroSys. Jedná se o nástroj umožňující realizovat **SCADA** systém, popř. realizovat **HMI**.

Po přečtení této kapitoly budete vědět

1. jakým způsobem jsou realizovány SCADA systémy
2. možnosti, jak vizualizovat různé typy údajů
3. jako funguje systém Promotic

Umět

1. vytvořit základní vizualizaci technologického procesu v systému Promotic



Čas pro studium

Odhad času potřebného pro prostudování této kapitoly je problematický, neboť tentokrát je očekáváno, že informace získané ve skriptech prakticky využijete při přípravě semestrálního projektu. Proto je potřeba jednotlivé kroky prakticky zkusit v Promoticu a experimentovat s prostředím ... různým lidem přitom taková práce trvá různě a některým pak výrazně déle než jiným.

K tomu Vám mohu pouze popřát, abyste u studia vytrvali ... látka není složitá, je pro Vás však nejspíše nová.

3.1 Promotic - instalace a projekt



Průvodce

V této části se zaměříme na přípravu prostředí Promotic pro naši práci, vytvoříme projekt a podíváme se na jeho jednotlivé komponenty.

Instalace

Před použitím je nutno Promotic instalovat. Promotic je dostupný pouze pro operační systém MS Windows. Podporované verze operačního systému se liší podle verze Promotic. V době poslední aktualizace těchto skript (IX 2025) jsou dostupné dvě základní verze systému:

- 9.0.x - nová verze systému, která vyšla ve stabilní verzi počátkem roku 2020, podporovány jsou operační systémy Windows 7 nebo novější - tato verze je aktivně vyvíjena
- 8.0 - 8.3 - jsou starší verze systému. Tato verze již není aktivně vyvíjena, je ale stále podporována opravou chyb.
- 7.5 - již nepodporovaná verze, která ale v praxi je někdy stále ještě používána

Na stránkách <https://www.promotic.eu> je možno stáhnout všechny výše uvedené verze prostředí Promotic. Z hlediska vývoje nových systémů lze jednoznačně doporučit použití poslední verze systému, což v době psaní těchto řádek byla verze 9.0.32. Tuto verzi použijeme také pro naše experimentování s tímto systémem.

Z hlediska kompatibility mezi verzemi je Promotic tzv. *dopředně kompatibilní*. Tím se rozumí, že novější verze systému je schopna přečíst a zpracovat projekty vytvořené ve starších verzích systému. Promotic načte projekt z předchozí verze programu a převede jej do formátu nové verze. Promotic tak **není zpětně kompatibilní**, tedy ve starších verzích programu projekty vytvořené novějšími verzemi neotevřete.

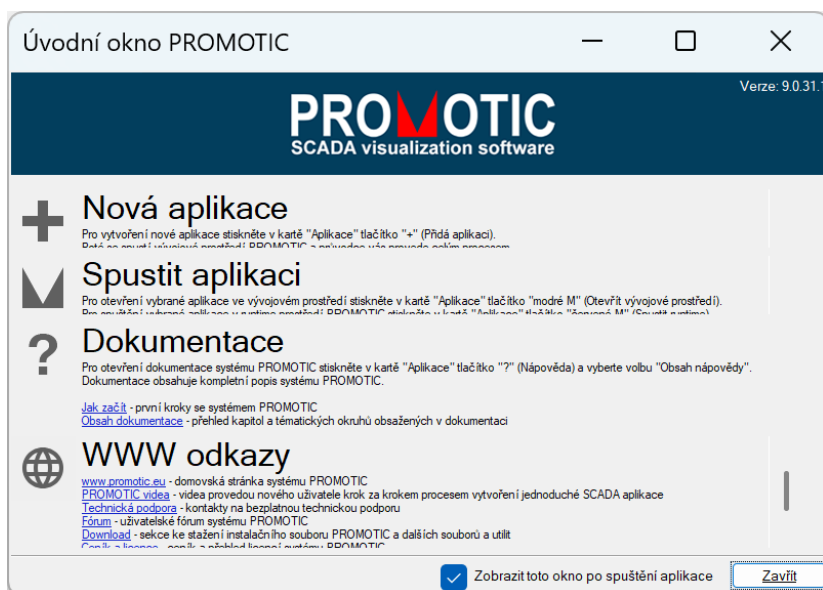
Pro zajištění možnosti vrátit se zpět ke starší verzi systému Promotic při povýšení projektu na novější verzi vytvoří kopii projektu. Tato kopie, ale nebude upravována a bude reprezentovat stav projekt v okamžiku provedení povýšení. Pokud tedy v projektu, v nové verzi provedeme změny a zjistíme nějaké problémy, které nás donutí vrátit se k některé z předchozích verzí. Tyto změny po obnově projektu ze zálohy již nebudou v projektu dostupné.

Ačkoliv je funkcionality povyšování projektů na nové verze poměrně dobře promyšlena a obvykle funguje dobře, z hlediska nasazení v praxi se jedná vždy o riziko. Po upgradu by tak mělo následovat otestování funkčnosti celého řešení. Funguje vizualizace procesu tak, jak očekáváme?

Někteří provozovatelé řeší tento problém tak, že vyvinutou aplikaci ponechávají na staré verzi prostředí - v tomto případě Promotic, ale obecně to platí pro všechny obdobné systémy. Jak víme z předchozích kapitol, je rozhodnutí zůstat na starší verzi ale také rizikové, protože v takovém případě provozujeme systém se známými a výrobcem opravenými chybami (které jsme se rozhodli ale neopravovat v produkčním prostředí).

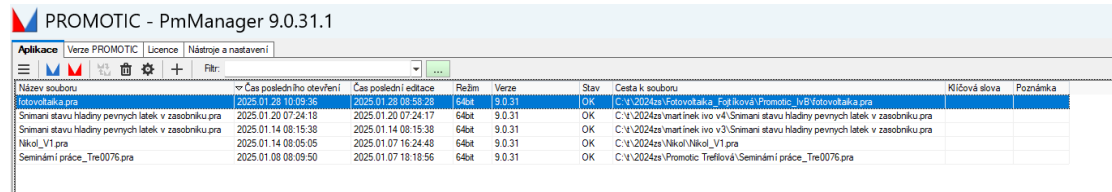
K instalaci jsou potřeba administrátorská práva k systému. Instalace samotná je ale přímočará, průvodce instalací je v češtině, takže by s instalací neměl nastat žádný problém. Podle stavu a verze operačního systému může být po dokončení vyžadován restart systému.

Systém Promotic je pak následně přístupný spuštěním PmManager z menu start. Po spuštění zobrazí program dialogové okno umožňující rychle vytvořit nový projekt, otevřít stávající nebo zobrazit dokumentaci systému, viz obr. 3.1.



Obrázek 3.1: Úvodní okno Promotic

PmManager se stará o management projektů, ale také běhových prostředí Promotic. Managementem projektů rozumíme, že vidíme seznam projektů, které jsou vyvíjeny na dané pracovní stanici se základními údaji k nim příslušejícími, např. cestou k projektu, datem vytvoření a poslední změny, ale také verzí vývojového prostředí, ve kterém byl daný projekt vyvinut, viz obr. 3.2.



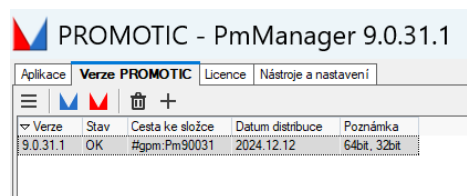
Název souboru	Čas poslední otevření	Čas poslední editace	Řeším	Verze	Stav	Cesta k souboru	Klíčová slova	Poznámka
Simulace	2025.01.20 07:24:15	2025.01.20 07:24:17	64bit	9.0.31	OK	C:\v\2024ta\Simulace\Simulace_Promotic_Vizualizace.pra		
Simulace stavu hladiny pevných látek v zásobníku.pra	2025.01.20 07:24:15	2025.01.20 07:24:17	64bit	9.0.31	OK	C:\v\2024ta\Simulace\Simulace_Promotic_Vizualizace.pra		
Simulace stavu hladiny pevných látek v zásobníku.pra	2025.01.14 08:15:38	2025.01.14 08:15:38	64bit	9.0.31	OK	C:\v\2024ta\Simulace\Simulace_Promotic_Vizualizace.pra		
Nikol_V1.pra	2025.01.14 08:05:05	2025.01.07 16:24:48	64bit	9.0.31	OK	C:\v\2024ta\Nikol_V1.pra		
Seminární práce_Tre0076.pra	2025.01.08 08:09:50	2025.01.07 18:18:56	64bit	9.0.31	OK	C:\v\2024ta\Promotic_Trefilová\Seminární práce_Tre0076.pra		

Obrázek 3.2: PmManager - správa projektů

V praxi se poměrně rutinně stává, že vyvíjeny jsou různé vizualizace využívající různé verze běhového prostředí Promotic. Např. můžeme vyvíjet novou vizualizaci technologického procesu v nejnovější verzi Promotic 9.0 a udržovat stávající verzi vizualizace v Promotic 8.3.

Instalace běhového prostředí probíhá spuštěním instalátoru této verze. Instalátor nové běhové prostředí doplní do systému a zpřístupní jej také v rámci PmManager. PmManager umožňuje také již nepotřebné verze běhového prostředí odinstalovat. My v průběhu semestru budeme pracovat pouze s jedinou verzí Promotic, takže management verzí běhového prostředí není pro nás tak zajímavý.

Náhled rozhraní je dostupný na obr. 3.3.



Verze	Stav	Cesta ke složce	Datum distribuce	Poznámka
9.0.31.1	OK	#gpm:Pm90031	2024.12.12	64bit, 32bit

Obrázek 3.3: PmManager - správa runtime modulů

Zkusme vytvořit nový projekt. V rámci jeho vytváření se budeme postupně seznamovat s jednotlivými aspekty, nástroji a postupy tvorby vizualizace. Začneme rozбором problému.

3.2 Vizualizace UPS - rozbor funkcionality

Připravíme jednoduchý projekt vizualizace fungování systému **Uninterruptable Power Source (UPS)**. Simulovat budeme tedy funkci nepřerušitelný zdroj napájení.

UPS funguje tak, že stojí mezi napájeným zařízením a elektrickou sítí. V případě, že dojde k výpadku elektrické energie nebo napětí v síti bude tzv. „špinavé“ - tedy bude výrazně odlišné od běžné úrovně napětí. UPS přeruší odběr energie ze sítě a napájenému zařízení místo toho bude dodávat elektrinu ze své baterie až do doby, kdy se napětí v síti vrátí do normálu, nebo kapacita baterie bude vyčerpána.

Představu o vzhledu takového zařízení si můžete udělat z obr. 3.4.

Pro náš projekt budeme předpokládat, že odběr elektriny napájeným zařízením bude konstantní. Pro funkci budeme potřebovat sledovat některé veličiny.

- stav UPS bude determinován tím, nakolik bude baterie UPS nabitá - řekněme v procentech.
- stav elektrické sítě bude determinována napětím v síti (ve voltech).
- každé napájené zařízení má odlišné požadavky na stabilitu napětí v síti - hranice toho, co je přijatelné (dolní/horní) by proto měly být nastavitelné. Údaj bude vedený ve Voltech.

Výše uvedené veličiny bude potřeba ve vizualizaci zpracovat. Za normálních okolností by zdrojem těchto údajů byla buďto technologie jako taková, resp. senzory, které dané veličiny měří a dálkově je zpřístupňují nebo data historian, kam jsou aktuální údaje o dané technologii v pravidelných časových intervalech aktualizovány.

Tedy údaje by byly generovány reálnou technologií. My bychom je přečetli, zobrazili bychom je a nějak bychom na ně reagovali.



Obrázek 3.4: UPS FINGERS FR-1250LCD [1]

V rámci zjednodušení, ale do našeho projektu, žádnou reálnou technologii nebudeme připojovat. Místo toho si vytvoříme jednoduchý simulátor. Pomocí kterého budeme tyto veličiny měnit. Obdobný přístup se používá i při vývoji reálných SCADA systémů. Většinou se ale pro tyto účely použije samostatný počítač, který simuluje technologický proces. My budeme mít simulaci integrovanou přímo v naší vizualizaci. Přece jenom, náš projekt bude jednoduchý.

Z pohledu počátečního nastavení veličin můžeme začít s úrovní napětí třeba na 230 V, což je standard a uvažovat dolní (DMN) a horní (HMN) mez přijatelného napětí ± 30 V, Tedy $DMN = 200$ V a $HMN = 260$ V.

Baterii nastavíme třeba na plné nabití, tedy 100 %.

V rámci simulace budeme chtít, aby se nám napětí v síti náhodně měnilo každou sekundu o 5 V směrem nahoru nebo dolů. Následně budeme vyhodnocovat stav napětí a podle něj budeme buďto dodávat elektřinu ze sítě a případně dobíjet baterii, pokud to potřebuje, nebo budeme dodávat elektřinu z baterie a ta se tím pádem bude vybíjet.

Pro zjednodušení předpokládejme, že ke změně bude docházet každou sekundu, kdy budeme vybíjet nebo nabíjet baterii o 1 %, podle toho, jestli je z ní dodávána elektřina nebo ne.

Jsou to realistické předpoklady? Popisuje to přesně jak funguje reálná UPS? **Absolutně ne!** Ale umožňuje nám to zaměřit se na zvládnutí nástroje, tedy Promotiku, aniž bychom se museli příliš zabývat technologií, kterou se snažíme řídit.



Rozbor problému

Předtím, než začnete zpracovávat projekt, vždy se snažte pochopit problém, který vizualizace má řešit. Pokud víte, co řešíte, jaké jsou cíle, které touto vizualizací plníte, tak se Vám bude projekt zpracovávat lépe. Postup uvažování podobný tomu nastíněnému v odstavcích výše proto budete muset absolvovat i pro třeba účely zpracování semestrálního projektu.

Obecně lze také říci, že podobný přístup je univerzální aplikovatelný a užitečný i při zpracování třeba také diplomové práce. Prostě platí, že pokud máte dobře rozmyšleno, co chcete řešit a jak, bude se Vám to řešit lépe, bez ohledu na technologii, nebo metodu, kterou při řešení použijete.

3.3 Implementace projektu v Promotic

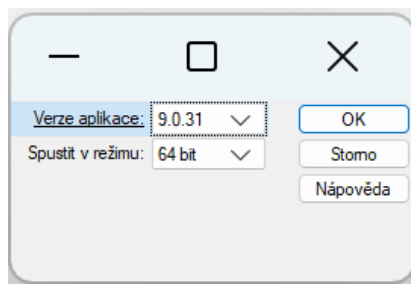
Implementace projektu probíhá v několika fázích, které postupně projdeme:

1. rozbor problému - již jsme realizovali v předchozí podkapitole
2. vytvoření projektu

3. nastavení projektu a příprava proměnných
4. návrh obrazu
5. oživení a ladění aplikace

3.3.1 Průvodce vytvoření projektu

Vytvoříme novou aplikaci pomocí menu (ikona „hamburgeru“) Přidat aplikaci → Přidat novou aplikaci. V dialogovém okně (viz obr. 3.5) vybereme Verzi runtime prostředí a režim. Předvolena je obvykle poslední verze (my máme nainstalovanou pouze jednu verzi) a 64 bitů. Naše volby potvrdíme kliknutím na tlačítko OK.



Obrázek 3.5: Nová aplikace Promotic

Spustí se průvodce nastavením nové aplikace, který nám umožní nastavit základy chování aplikace. V prvním kroku máme tři možnosti toho, jak má aplikace být vytvořena:

- základní aplikace
- aplikace s příklady
- kurz Promotic

Základní aplikace vytvoří novou, prázdnou aplikaci - to je přesně to, co potřebujeme.

Zbývající dvě možnosti nám umožňují vytvořit si vzorové/demonstrační aplikace, na který je demonstrováno, co Promotic umí. Tyto jsou vhodné pro experimentování s programem a inspiraci. My jsem ale již dostatečně inspirováni předchozí kapitolou, vytvoříme si proto základní aplikaci.

V druhém kroku definujeme základní vlastnosti nové aplikace, viz obr. 3.6.

Zde specifikujeme cestu k souboru s příponou .pra, který bude obsahovat náš nový projekt. **Doporučujeme pro každý projekt vytvořit samostatnou složku.** Většina obsahu projektu je sice v uvedeném PRA souboru, ale ne nutně úplně všechno. Vizualizace může např. vyžadovat externí ilustrační obrázky. Ty se vkládají do samostatné složky.

Pokud si vytvoříme pro projekt složku a PRA soubor uložíme do ní, nemusíme řešit co přináleží ke kterému projektu. Vše k projektu je v jedné složce. Jedná se o jednoduché a přitom elegantní řešení.

Já jsem složku projektu, PRA soubor a název aplikace nastavil stejně na UPS.

Z hlediska dalších nastavení volíme Skriptovací jazyk aplikace. Předvolen je Java skript s možností změny na Visual Basic skript (VBS). VBS je v Promotic považován za výběhový skriptovací jazyk, který v budoucích verzích Promotic přestane být pravděpodobně podporován. Při našem výkladu se proto zaměříme na Java skript. Ostatní nastavení na této obrazovce nemá pro naše účely smysl měnit.

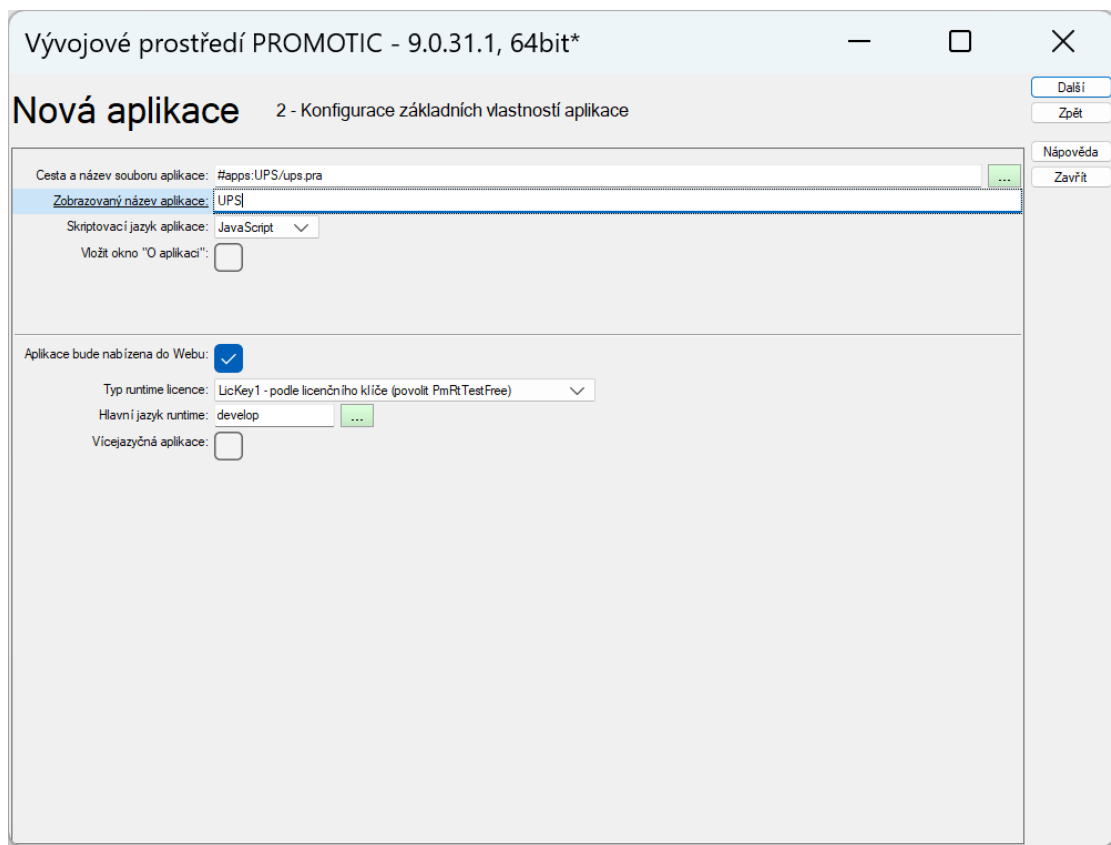
V dalším kroku volíme rozložení aplikace ve smyslu nastavení toho, kde se má nacházet hlavní panel nástrojů - může se nacházet nahoře, nebo vlevo. Pro účely tohoto projektu jsem zvolil předvolené rozložení s panelem nástrojů nahoře, viz obr. 3.7. Volba odlišného rozložení nebude mít vliv na funkčnost. Budeme mít tedy ve finále dostupné stejné nástroje, pouze mohou být umístěny na jiném místě.

Následně volíme, jak má tento panel nástrojů vypadat. Jelikož se jedná o čistě estetickou volbu, zvolte to co se Vám líbí, já jsem zvolil poslední možnost.

Toto byl poslední krok průvodce, po jehož dokončení nám bude vygenerována nová aplikace s nastaveními, která jsme zadali v rámci průvodce. Předtím, než se pustíme do práce s Promotic podívejme se jak vypadá projekt na disku.

Pomocí průzkumníka jděte do projektové složky, uvidíte v ní následující strukturu složek projektu:

- zdroje (Resource)
- datové soubory (Data)



Obrázek 3.6: Nastavení základních vlastností aplikace Promotic

- konfigurační soubory (Cfg) a
- dočasné soubory (Temp)

V hlavní složce projektu nalezneme náš PRA soubor, který obsahuje strukturu projektu. Z výše uvedených složek věnujte pozornost především složce *Resource*, do které vkládáme zdroje, které má Promotic použít při vývoji a provozu aplikace. Jedná se především o obrázky a videa.

Promotic sice obsahuje poměrně rozsáhlou paletu grafických prvků, ze kterých lze jednoduše sestavit efektní vizualizaci, někdy ji ale potřebujeme rozšířit o prvky které nejsou součástí takové palety. Podporovány jsou formáty:

- obrázky - bitmapy: JPG, PNG, GIF
- obrázky - vektorové: SVG
- videa: AVI, MPG, WMV, DivX a další, za předpokladu, že v operačním systému jsou nainstalovány kodeky umožňující jejich přehrávání

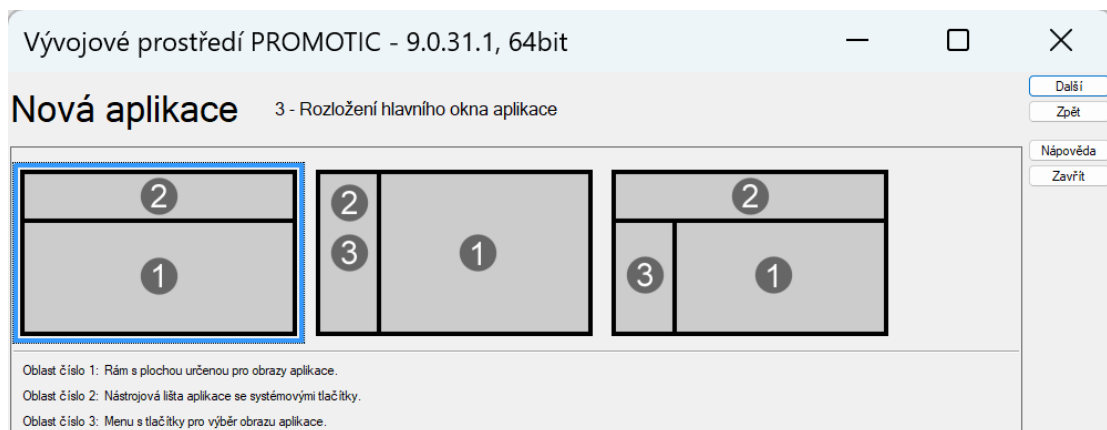
Drobné upozornění, pokud doplníte do složky *Resource* jakýkoliv soubor, je potřeba výslednou aplikaci Promotic distribuovat/přenášet včetně této složky. Bez ní nebudou k dispozici objekty, což může u uživatelů způsobit zmatení (aplikace vypadá jinak) nebo dokonce může způsobit nefunkčnost aplikace.

Nejjednodušším způsobem jak se těmto problémům vyhnout je přenášet celou složku projektu, což zajistí spolehlivě a jednoduše, že aplikace bude mít k dispozici všechny prvky, které potřebuje ke své funkci.

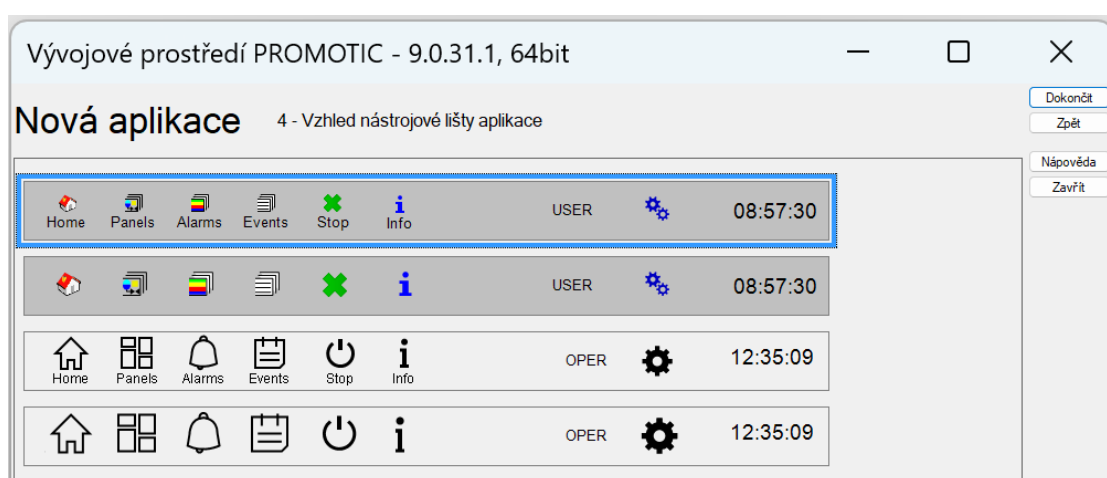
Technicky ostatní složky (*Data*, *Cfg* a *Temp*) obvykle nejsou vyžadovány, resp. Promotic je schopen v rámci své činnosti si je rekonstruovat dle svých potřeb.

3.3.2 Nastavení projektu a příprava proměnných

Vraťme se do aplikace rozhraní Promotic. Nový projekt vytvořený ve verzi 9 programu bude vypadat strukturálně stejně jako na obr. ??.



Obrázek 3.7: Volba rozložení aplikace Promotic



Obrázek 3.8: Volba vzhledu hlavního panelu aplikace Promotic

Vývojové prostředí má tři základní části:

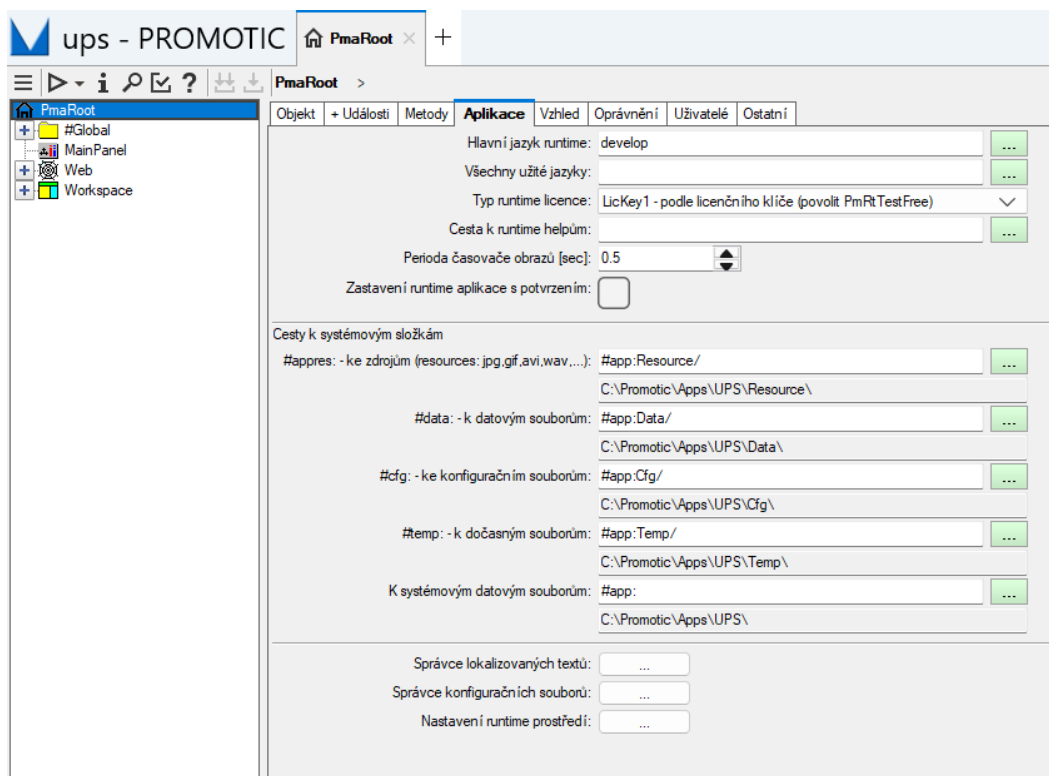
- panel nástrojů (nahore),
- průzkumník projektu (vlevo) a
- největší část obrazovky pak zabírají vlastnosti právě zvoleného objektu (vpravo).

Vlastnosti objektu se mění v závislosti na tom, jaký objekt je vybrán. Na obr. 3.9 je vybrán objekt PmRoot, který je základním stavebním kamenem projektu. Odtud také název Root (kořen). Vlastnosti jsou tedy kontextově senzitivní, pokud si vyberete tedy jiný objekt, bude mít jiné vlastnosti.

Vzhledem k tomu, že naším cílem je seznámit se pouze se základy tvorby projektů, zaměříme se při výkladu pouze na základní vlastnosti nejčastěji používaných objektů. Předtím, než začneme se ale ještě podíváme na strukturu projektu jako takovou. V projektu se vytvořil kořen (PmRoot), všechny další objekty projektu jsou pod ním. PmRoot tedy funguje jako základní složka projektu.

Projekt dále vytvořil obrazy Main Panel a Toolbat. Main panel je základní obrazovka aplikace, Toolbar pak obsahuje panel nástrojů, který se bude přidávat do aplikace. Co kam patří je definováno ve Workspace.

Tyto tři objekty by měly být v každém projektu (byť různé verze Promoticu je organizují trochu odlišně). V žádném případě byste proto tyto objekty neměli z projektu mazat. Pokud tak učiníte, Vaše aplikace s největší pravděpodobností nebude fungovat. Z tohoto stavu je možné se zotavit doplněním výše uvedených objektů a jejich znovunastavením, ale Vám jako začátečníkům nejspíše bez jakýchkoliv znalostí Promotic, se tato obnova bude realizovat velmi obtížně.



Obrázek 3.9: Promotic - základní rozhraní projektu

PmRoot

Jedná se o kořenovou složku projektu - zde je možno nastavit vlastnosti projektu jako celku. Nej důležitější nastavení se nacházejí na záložce *Aplikace*. Zde je možno třeba provést změnu nastavení režimu runtime. Z dalších důležitých vlastností je *perioda časovače obrazů*. Jedná se o hodnotu v sekundách určující jak často se překreslí obraz na obrazovce. Předvolená hodnota 0.5 znamená, že obraz se překreslí 2x za sekundu.

Perioda časovače by měla být nastavena menší než je interval aktualizace dat ze senzorů, které mají být vizualizovány. Hodnota 0.5 obvykle pro běžnou práci dostačuje a není potřeba ji měnit.

Pozornost věnujte také sekci *cesty k systémovým složkám*. Tato skupina nastavení umožňuje specifikovat kam, se má Promotic dívat při hledání určitých dat, nebo zdrojů. Samostatně je možno nastavit:

V PmRoot můžete také přepnout jazyk, ve kterém budou napsané aktivní části aplikace, pokud Vám nevyhovuje jazyk zvolený pomocí průvodce. Tuto volbu naleznete na záložce Objekt → skriptovací jazyk objektu. V našem případě máme předvolen Java Script. Tuto volbu je možné změnit na Visual Basic Script. V této kapitole ale budeme pracovat s Java Scriptem a tak v současnosti nebudeme tuto volbu měnit.

Obecně platí, že všechny volby, které nám Promotic umožňuje nastavit v rámci průvodců, je možno změnit později změnou vlastností jednotlivých objektů. Jednotlivé objekty Promotic, ale obvykle mají k dispozici podstatně větší množství vlastností než nám představil průvodce. Pozdější nastavení je proto méně pohodlné. Logicky, pokud by to tak nebylo, nebylo by potřeba, aby takový průvodci vůbec existovali.

Správnému nastavení vlastností objektů v rámci průvodců proto věnujte zvýšenou pozornost - může Vám uspořit velké množství času.

PmPanel

Objekt typu *PmPanel*, česky uživatelská grafika umožňuje do projektu přidávat obrazy. V našem projektu máme „od novoty“ připraveny dva panely a to konkrétně hlavní panel, nazvaný MainPanel, který bude obsahovat hlavní okno grafiky naší aplikace.

Každý obraz by měl mít specifikovaný titul, aby jej v případě potřeby bylo možné při běhu aplikace najít. To je zejména důležité, pokud aplikace obsahuje větší množství obrazů, což většina SCADA

systémů má.

Pro náš jednoduchý projekt, ale bude postačovat toto jedno okno.

Druhý objekt PmPanel se skrývá ve Workspace a nazývá se Toolbar.

Toolbar obsahuje základní ovládací panel pro aplikaci. Tento panel je pro každou aplikaci stejný a proto bychom jej neměli měnit. Technicky to ale samozřejmě jde. Před zásahem do tohoto obrazu byste měli, ale vědět, co děláte.

Pro obraz je možno nastavit také skriptovací engine - implicitní nastavení odpovídá jazyku zvolenému při vytváření projektu. V rámci projektu tak můžeme kombinovat oba skriptovací jazyky. Z praktického pohledu to ale nemá smysl.

Editace obrazu v minulých verzích Promotic byla realizována samostatnou aplikací, ve verzi 9 je ale již integrovaná do hlavního vývojového prostředí a je dostupná na záložce *Grafika*.

Problematické editace grafického obsahu se budeme věnovat podrobněji v samostatné podkapitole.

PmWeb a workspace

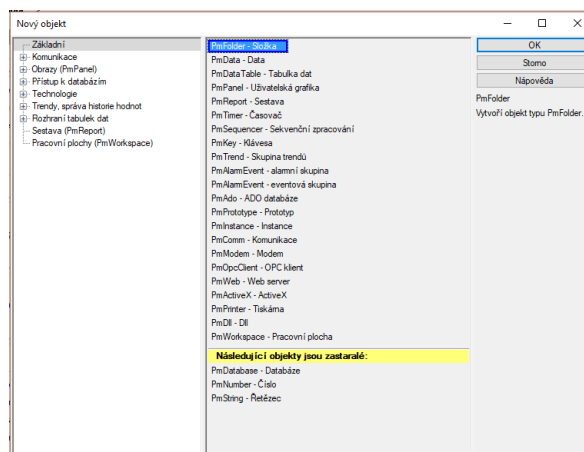
Komponenta web umožňuje nastavení aplikace tak, aby její obsah bylo možné zobrazit a ovládat dálkově pomocí webového prohlížeče. Toto je způsob, který se v reálných aplikacích používá velmi často. Pro naše experimenty si ale můžeme dovolit tuto komponentu ignorovat.

Podobné workspace slouží pro nastavení pracovního prostoru - tedy např. jaký toolbar se použije, kde se bude nacházet, jaké hlavní okno má aplikace použít, atd. Pro naše účely je tato komponenta již dostatečně nastavena průvodcem vytvoření projektu a nebudeme proto její vlastnosti měnit.

To jsou tedy objekty, které nám vytvořil Promotic sám, nyní začneme pro naši aplikaci přidávat objekty sami. Objekty můžeme vytvářet ve složkách, tzn. v PmRoot, nebo jakékoliv další složce, kterou jsme si v projektu vytvořili.

Nový objekt do projektu můžeme přidat buďto z kontextového menu složky v průzkumníku projektu (v našem případě klikneme na objekt PmRoot, protože to je jediný objekt typu složka, který v projektu máme) a v kontextovém menu klikneme na položku *Nový objekt* Vzhledem k tomu, že objektů budeme přidávat více, může být výhodné použití klávesové zkratky Ins (klávesa insert), která panel objektů spustí také.

Objeví se paleta dostupných objektů, viz obr. 3.10.



Obrázek 3.10: Promotic - nový objekt

Ve skriptech si představíme pouze malou podmnožinu těchto nástrojů - neprobereme ani všechny základní nástroje. Objekty typu PmWorkspace a PmPanel jsme už probrali (viz výše), jaké další objekty nám tedy nabízí Promotic?

PmFolder - složka

Složka slouží pro organizaci dalších objektů v projektu. Hodí se zejména v případě, že pracujeme s rozsáhlými projekty obsahující desítky nebo stovky objektů různého typu. Dobrou volbou může být použití systému složek pro rozčlenění projektu - buďto tématického nebo podle typu objektu.

Tématickým členěním projektu rozumíme postup v rámci kterého do složek vkládáme objekty různého typu týkající se určité části řešené technologie/problému.

Členění podle typu pak používá strukturu složek pro rozdělení objektů podle jejich typu. Naše projekty budou poměrně malé, takže se bez použití složek obejdeme.

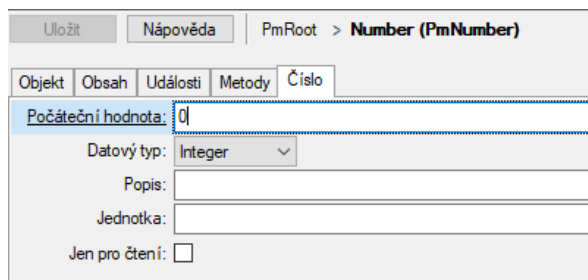
PmData, PmString, PmNumber

Tento objekt slouží pro management proměnných, se kterými bude Promotic pracovat. Proměnné obvykle odpovídají buďto měřeným veličinám nebo slouží pro interní potřeby aplikace.

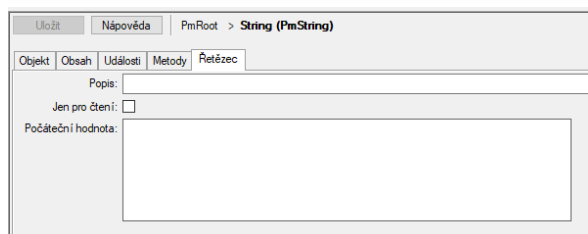
Proměnné je možné vytvářet také jako samostatné objekty, konkrétně typu PmNumber pro čísla nebo PmString pro řetězce. Od verze 8.1 Promotic jsou ale tyto objekty označovány jako zastaralé a preferováno je použití objektu PmData.

U zastaralých objektů lze očekávat, že v budoucích verzích bude odstraněna jejich podpora. V současnosti ale není zcela jasné jak dlouho budou tyto typy objektů ještě podporovány.

Na obr. 3.11 jsou zobrazeny vlastnosti objektu PmNumber, na obr. 3.12 jsou pak zobrazeny vlastnosti objektu PmString.



Obrázek 3.11: Promotic - PmNumber - objekt typu číslo



Obrázek 3.12: Promotic - PmString - objekt typu textový řetězec

Jak je z obrázků 3.11 a 3.12 patrné oba objekty využívají podobných vlastností. Každý objekt má vlastní jméno, které by mělo být v rámci dané složky unikátní (toto se týká všech typů objektů). Pro proměnné lze nastavit výchozí hodnotu, popis a specifikovat zda se jedná o proměnnou nebo konstantu.

Počáteční hodnota je poměrně důležitá - jedná se o hodnotu, na kterou se daná proměnná nastaví při spuštění aplikace. Tato hodnota pak následně může být měněna z technologie (načítána z externího datového zdroje) nebo může být měněna uvnitř samotné aplikace, např. nějakým skriptem.

Často pokud se aplikace chová odlišně než čekáte je problém právě v počátečním nastavení - tento problém je ale zároveň velmi jednoduchý na opravu, takže rozhodně nemůže škodit se na tuto vlastnost při hledání chyb podívat.

Popis slouží pro vnitřní dokumentaci proměnné. Hodnota, kterou do této vlastnosti zadáte tedy bude sloužit Vám k lepší orientaci v tvořené aplikaci. Do popisu se obvykle vkládá účel proměnné, popř. další informace, které by později mohly být při manipulaci s aplikací užitečné.

Popis není povinný, pokud jej tedy u proměnné neuvedete, nebude to mít vliv na funkci Vámi vytvářené aplikace.

Proměnnou lze nastavit jako konstantu zaškrtnutím volby *jen pro čtení*. Po zaškrtnutí hodnota, která byla nastavena jako počáteční, již nemůže být změněna.

U čísel (PmNumber) je možno nastavit o jaký typ čísla se jedná:

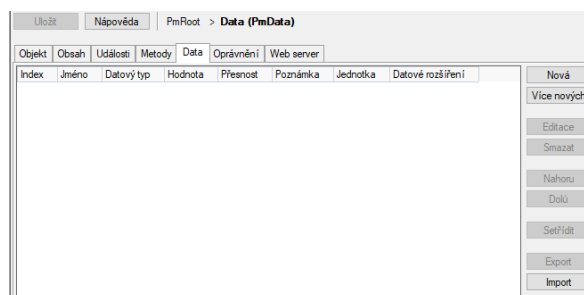
- boolean (0/1)
- byte (0 - 255)
- integer - celočíselný datový typ (16-ti bitové, -32768 - 32768)

- long - celočíselný datový typ (32 bitový, -2 147 483 648 - 2 147 483 648)
- single - desetinné číslo, jednoduchá přesnost (32 bitové $-3.4 \cdot 10^{38} - 3.4 \cdot 10^{38}$)
- double - desetinné číslo, dvojitá přesnost (64 bitové, $-1,79 \cdot 10^{308} - 1.79 \cdot 10^{308}$)

U čísel je možno nastavit také jednotku. Podobně jako popis také jednotka není povinná. Tato vlastnost slouží pro předcházení problémům s interpretací jednotlivých veličin. Pokud u číselné proměnné tlak jednotku neuvedeme, můžeme předpokládat, že je vedena v Pascalech (Pa), ale co když zbytek aplikace předpokládá, že je vedena v kPa nebo dokonce v MPa?

Vyplnění jednotky samo o sobě tento problém zcela neodstraní, umožní ale aby v průběhu vytváření aplikace jste klikli na proměnnou a měli ten AHA moment ... veličina je v té a té jednotce (a nemusíte o tom přemýšlet).

Rozhraní objektu PmData vypadá odlišně, viz obr. 3.13.



Obrázek 3.13: Promotic - PmData - sdružující proměnné různého typu

Nové proměnné do objektu PmData zadáváme kliknutím na tlačítko *Nová*. Rozhraní definice proměnné je univerzální. Kromě datových typů podporovaných v proměnných PmNumber a PmStrin podporuje tento objekt také datové typy Date (datum a čas), Object (objektový datový typ) a Variant (neurčený datový typ, datový typ se určuje až ze zadané hodnoty a způsobu použití proměnné).

Vzhledem k tomu, že k proměnným budeme potřebovat přistupovat, je velmi důležité specifikovat cestu k proměnné. Cestu naleznete ve vlastnostech daného objektu vedle tlačítek uložit a nápověda. Na obr. 3.13 je cesta PmRoot/Data (objekt PmData jsem pojmenoval Data).

V případě, že bych v objektu Data nadefinoval proměnnou napeti, byla by celá cesta k ní PmRoot/Data/napeti.

V případě, že proměnné jsou uvnitř složky (nebo složek) je název složky také součástí cesty.

Chybné mapování proměnných je další častou chybou, na kterou je potřeba si dát pozor.

Aby náš projekt fungoval, potřebujeme zřídit objekty typu PmNumber nebo zřídit jeden nový objekt PmData a v rámci něj pak zřídit proměnné pro naši aplikaci. Oba přístupy jsou funkčně ekvivalentní, ale použití PmData objektu je považováno za modernější přístup, a proto jej v našem projektu použijeme i my.

Zřídíme následující proměnné typu celé číslo (název proměnné kurzívou):

- *napeti* - napětí v síti, jednotka V, počáteční hodnota 230
- *baterie* - stav nabití baterie UPS, jednotka %, počáteční hodnota 100
- *dmn* - dolní mez napětí, jednotka V, počáteční hodnota 200
- *hmn* - horní mez napětí, jednotka V, počáteční hodnota 260

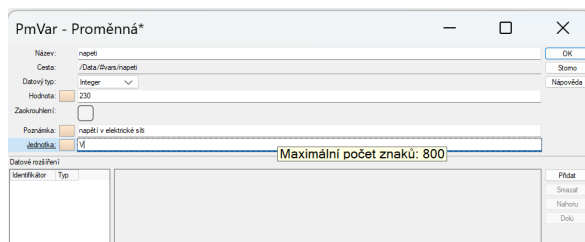
Všimněte si, že výše uvedené proměnné věrně kopírují náš myšlenkový proces a veličiny, se kterými jsme pracovali při vytváření myšlenkového modelu v předchozí podkapitole. Tady vidíme, že prvotní myšlenkový model vycházející z rozboru řešeného problému je klíčový pro úspěšné vyřešení tohoto problému.

V objektu PmData přidáváme proměnné kliknutím na ikonu „+“. Příklad vyplnění pro proměnnou *napeti* je znázorněn na obr. 3.14.

Vlastnosti poznámka a jednotka v rámci Promotic mají pouze dokumentační funkci a systém je jinak nepoužívá. Jejich vyplnění nám ale pomůže s pochopením toho, co který název proměnné vlastně znamená.

PmTimer - časovač

Časovač slouží pro spouštění určitých operací (definovaných skriptem) ve specifikovaném časovém



Obrázek 3.14: Promotic - Vytvoření proměnné celočíselného typu v PmData

intervalu. V praxi se časovače používají pro pravidelnou aktualizaci stavů aplikace na základě hodnot připojených proměnných. K tomuto účelu časovač budeme využívat také.

Pro naše použití ale časovač bude plnit ještě jednu úlohu. Za normálních okolností jsou hodnoty proměnných odečítány z externího zdroje - technologie ať už na přímo nebo zprostředkovaně s data historianu. My ale žádnou technologii připojenou nemáme - takže potřebujeme něco, co nám technologii bude simulovat. Tuto simulaci lze realizovat také pomocí časovače.

Formálně se činnosti, které časovač má vykonávat specifikují pomocí zvoleného programovacího jazyka, v současnosti jsou podporovány jazyky Java Script a Visual Basic Script (VBS). My budeme používat Java Skript.

Problematické oživení časovače budeme věnovat samostatnou podkapitolu, v současnosti jej ale můžeme vytvořit a ve vlastnostech nastavit periodu časovače na 1 sekundu, což je předvolená hodnota.



Programování?

Výše v textu se objevila podivná slova - programovací jazyky (dokonce **dva!**). To jakože znamená, že budeme programovat?

Technicky vzato - ano, přesně to to znamená. Prakticky, ale budeme využívat jen velice úzkou podmnožinu funkcionality poskytované Visual Basicem - takže to nebude tak hrozné :-).

3.3.3 Návrh obrazu aplikace

Základní strukturu projektu máme hotovou, nyní se zaměříme na problematiku vytvoření vizualizace našeho sofistikovaného technologického procesu.

V naší aplikaci nám bude postačovat jediný obraz, proto použijeme hlavní panel (main panel), který nám vytvořil Promotic sám, při prvotním vytváření projektu. Vybereme tento objekt kliknutím a v jeho vlastnostech se přepneme na grafiku, kde budeme navrhovat strukturu vizualizace.

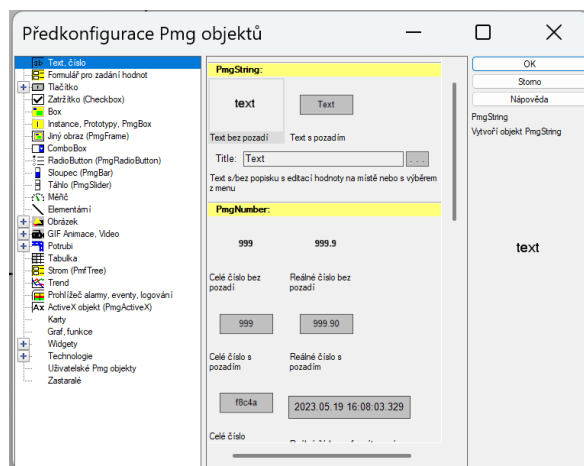
V našem obraze je už objekt typu PmText, ve kterém je předvyplněný název našeho projektu. Ten můžeme, ale také nemusíme v našem projektu ponechat. Já jsem jej ve svém projektu smazal, což uděláme tak, že objekt vybereme (kliknutím) a pak smažeme buďto z kontextového menu nebo stisknutím klávesy delete.

Nové objekty v obraze vytváříme dvojitým kliknutím na prázdné ploše. Tím se nám spustí paleta předkonfigurovaných objektů, které nám Promotic připravil pro jednoduché použití v našich aplikacích, viz obr. 3.15.

Vybíráme vždy takové objekty, které plní naše potřeby v budoucí vizualizaci. Nemáme prostor k tomu, abychom v těchto skriptech rozebrali všechny typy objektů, které Promotic nabízí. Zaměříme se proto na výběr takových objektů, které byste mohli chtít použít a některé z nich použijeme pro naši aplikaci.

Text je určen k zobrazení buďto statického textového řetězce, nebo textového řetězce, který přebíráme z nějaké proměnné uložené např. v objektu PmData. V tomto případě textovým řetězcem rozumíme jakýkoliv alfanumerický řetězec, tzn. že se může jednat i o číslo.

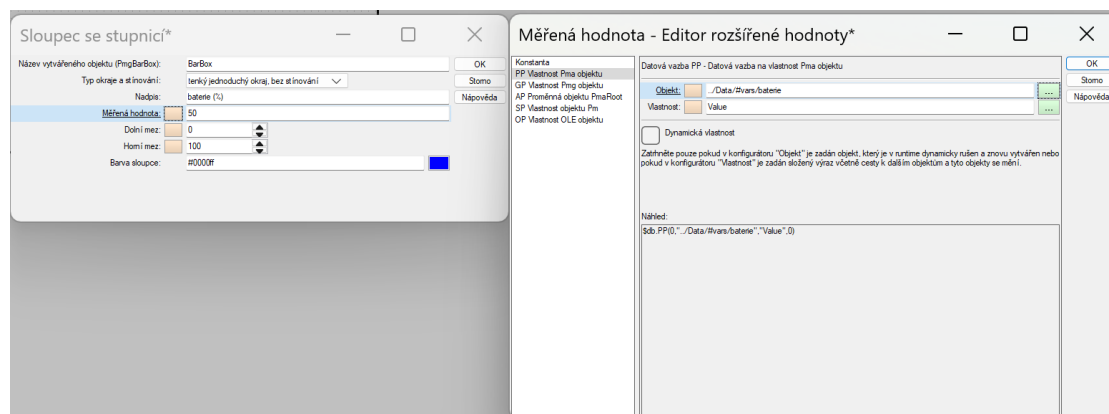
Sloupec umožňuje zadání zobrazení horizontálního nebo vertikálního sloupce. V našem případě, by se tento objekt mohl hodit pro zobrazení stavu nabití baterie, ideálně se škálou od 0 do 100 %. Vybereme proto *sloupec se stupnicí vodorovný, stupnice nahore*. Vybereme buďto kliknutím na daném objektu a potvrzením tlačítkem OK, nebo vybíráme dvojklikem, který objekt vybere a rovnou jej potvrdí.



Obrázek 3.15: Promotic - paleta předkonfigurovaných objektů pro použití v obrazech

Po vybrání se zobrazí jednoduchý průvodce vyplněním, který nám nabízí nejpoužívanější vlastnosti objektu sloupce k vyplnění. Vždy doporučujeme maximum vlastností vyplnit zde. Změnit je sice možné tyto vlastnosti také později, avšak budete je muset najít v záplavě všech vlastností, které lze pro tento objekt nastavit.

Pro baterii nastavíme podobně jako na obr. 3.16. Nastavení je v zásadě přímočaré, pouze pro tzv. měřenou hodnotu je složitější. Měřenou hodnotou rozumíme hodnotu, která jsme naměřili a chceme ji zobrazit. Zde potřebujeme nadefinovat propojení na proměnnou baterie, kterou jsme si vytvořili v objektu PmData. Uděláme to tak, že klikneme na tlačítko vedle měřené hodnoty a v dalším formuláři vybereme vazbu typu PP (vlastnost objektu) a nastavíme objekt kliknutím na „...“, kde se proklikáme k proměnné baterie.



Obrázek 3.16: Promotic - nastavení sloupce pro zobrazení % nabití baterie

Všimněte si, že kurzor se změnil a my máme možnost kliknutím a tažením na ploše obrazu definujeme umístění a velikost objektu.

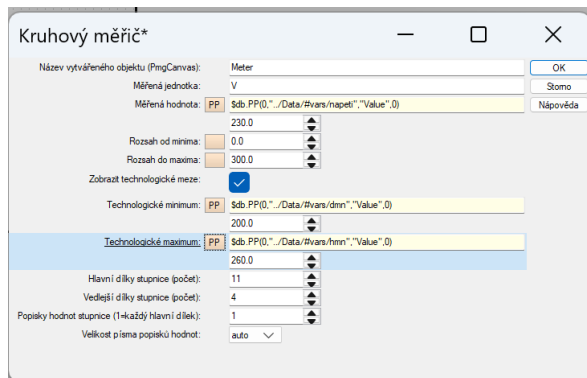
Po vytvoření objektu můžeme kdykoliv jej vybrat (kliknutím), změnit je ho pozici přetažením, nebo jeho velikost - tažením za rohy. Můžeme také kdykoliv změnit jeho vlastnosti. Na paletu vlastností objektu se dostanete dvojitým kliknutím na objekt.

Zejména v počátcích Vaší práce se systémem doporučujeme experimentovat s různými nastaveními vlastností a dopady, které to má na vzhled a funkčnost objektu.

Dalším objektem který vytvoříme bude *měřič*. Tato skupina objektů poskytuje různé budíky, měřadla apod., které slouží pro zobrazení veličiny v nějakém rozsahu. Svým způsobem je objekt podobný sloupci, který také pracuje s určitými rozsahy, byť měřič je vizuálně efektivnější a umožňuje i některé další praktická nastavení, která se nám v naší vizualizaci budou hodit. Z nabídky vybereme hned první z nabízených objektů - *kruhový měřič*.

Měřič použijeme pro vizualizaci napětí v síti a použijeme také přijatelné meze napětí. Objekt nastavíme následovně (graficky pak viz obr. 3.17):

- měřená jednotka: V
- měřená hodnota: 230 a připojíme vazbu PP na proměnnou Promotic: napeti (postup viz předchozí objekt)
- rozsah od minima: 0
- rozsah do maxima: 300
- technologické minimum: 200 a připojíme vazbu PP na proměnnou DMN
- technologické maximum: 260 a připojíme vazbu PM na proměnnou HMN
- ostatní vlastnosti mohou zůstat beze změny



Obrázek 3.17: Promotic - nastavení měřidla typu kruhový měřič

Kruhový měřič má tři barevné části: žlutou pro oblast pod technologickým minimem, červenou naopak nad technologickým maximem a zelenou oblast, kde je hodnota v technologických mezích.

V našem případě jsme technologické meze nastavili jak staticky na hodnoty 200, resp. 260 V, tak dynamicky připojením proměnných Promotic vazbou typu PP. Z hlediska funkčnosti je nezbytná pouze vazba PP. Statické nastavení bude viditelné pouze v editoru obrazů, ale nebude mít vliv na to, jak bude vypadat budík po spuštění vizualizace.

Nař. pro technologické minimum po spuštění vizualizace Promotic připojí proměnnou DMN, ze které načte přednastavenou (počáteční hodnotu), což v našem případě je také hodnota 200. Statickým nastavením tedy zajišťujeme že během vývoje budeme vizuálně vidět objekt nastavený stejně jak bude vypadat při reálné m provozu aplikace. Tedy tento krok není nezbytně nutný, ale pokud uděláte, budete se lépe ve Vámi vyvíjené vizualizaci orientovat.

Dolní a horní mez chceme také nastavovat. Použijeme k tomu objekt typu *táhlo*. to nám umožní meze nastavit tažením v určitém přenastaveném intervalu, popř. hodnotu zadat manuálně přepsáním. Pro dolní i horní mez použijeme stejný typ objektu, proto níže vysvětlím pouze nastavení dolní meze, horní doplníte analogicky.

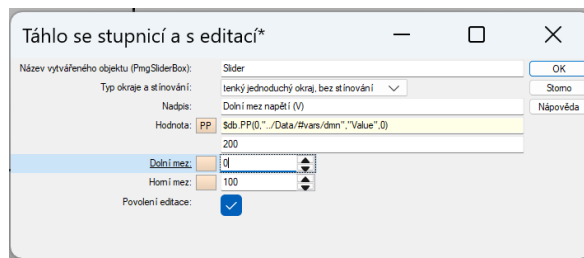
Z nabídky *táholo* (pmSlider) vybereme *Táhlo se stupnicí a s editací vodorovné*. Nastavíme jej následovně (viz také obr. 3.18):

- nadpis: Dolní mez napětí (V)
- hodnota: 200 a připojíme proměnnou DMN pomocí vazby typu PP
- dolní mez: 170
- horní mez: 230

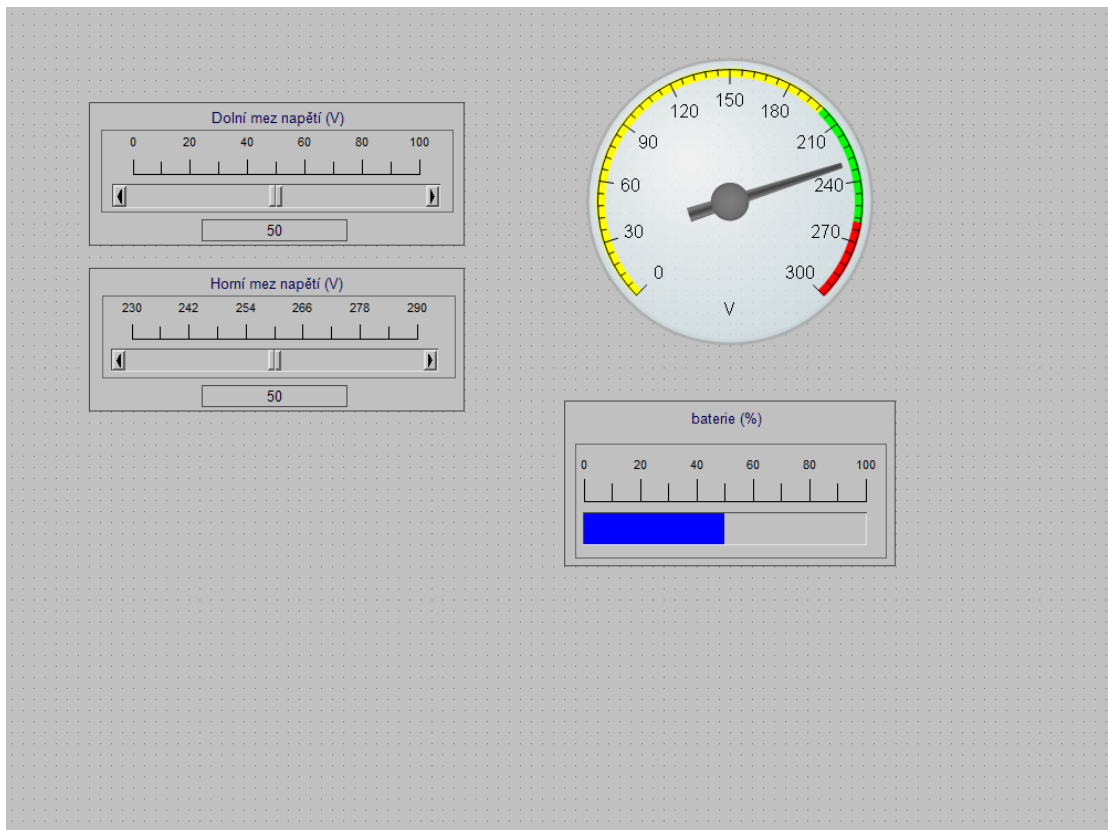
Analogicky k předchozímu postupu nastavte nové táhlo pro horní mez napětí. Toto nastavení by nám mělo umožnit, aby se nastavení dolní a horní meze propagovalo také do měřidla pro nastavení technologických mezí.

Celkově by vizualizace měla vypadat podobně jako na obr. 3.19. Pozice Vašich objektů se samozřejmě může lišit. Umístění objektů nemá vliv na funkčnost.

Vizuální vzhled naší aplikace je tedy hotov. Dokonce ji již můžeme spustit a bude funkční ve smyslu, že táhla můžeme přenastavovat a uvidíme, jak se tyto změny budou propagovat do měřícího přístroje. Spuštění vizualizace provedeme kliknutím na šipku ($\triangleright$) v hlavním panelu nástrojů vlevo nahoře.



Obrázek 3.18: Promotic - nastavení táhla pro dolní mez napětí



Obrázek 3.19: Promotic - celková vizualizace UPS

3.3.4 Oživení aplikace

Ze zkušenosti je pro studenty nejobtížnější oživení aplikace. Provádíme jej totiž nastavení časovače (timeru), což se realizuje pomocí skriptu v programovacím jazyce JavaScript (JS) nebo Visual Basic Script (VBS). Řešení v JS naleznete v této kapitole. V příloze 1 je pak dostupné řešení ve VBS.

Pro oživení použijeme následující kód:

Výpis 3.1: JavaScript: oživení časovače projektu UPS v Promotic 9.0

```

1  var napeti = Pm("/Data/#vars/napeti");
2  var baterie = Pm("/Data/#vars/baterie");
3  var dmn = Pm("/Data/#vars/dmn");
4  var hmn = Pm("/Data/#vars/hmn");
5  var n = Math.random();
6
7  if(n > 0.5){
8      napeti.Value++;
9  }else{
10     if(napeti.Value > 0) napeti.Value--;
11 }
12

```

```

13  if(napeti.Value >= dmn.Value && napeti.Value <= hmn.Value){
14      if(baterie.Value < 100) baterie.Value++;
15  }else{
16      if(baterie.Value > 0) baterie.Value--;
17  }

```

Na řádcích 1 - 4 provádíme deklaraci proměnných timeru pomocí příkazu `var`. Proměnná před svým použitím musí být vždy deklarovaná. V našem případě deklaraci spojujeme s iniciací proměnné, konkrétně do ní připojujeme námi předem definovaný objekt `PmData` nazvaný `Data`. V rámci něj máme definované čtyři proměnné celočíselného datového typu (`napeti`, `baterie`, `dmn` a `hmn`).

Tyto proměnné Promotiku potřebujeme namapovat na proměnné timeru (časovače). To děláme pomocí funkce `Pm`, která jako parametr bere plnou cestu k proměnné, např. `/Data/#vars/napeti`. Kde `/Data` je jméno objektu `PmData`, `/#vars` se odkazuje na jeho proměnné (pokud použijeme objekt `PmData`, vždy budeme odkazovat jeho proměnné proměnné pomocí `/#vars` a konečně `/napeti` je název dané proměnné.

Pokud jste objekt `PmData` nebo proměnnou pojmenovali jinak, musíte použít stejný název jaký jste zvolili pro objekt/proměnnou (anebo je přejmenovat).

Pokud bychom používali pro jednotlivé proměnné samostatné objekty jako třeba `PmNumber`, připojovali bychom je velmi podobně. V případě, že bychom vytvořili v `PmRoot` objekt `PmNumber` a nazvali jej třeba `napeti2`, pak by namapování vypadalo následovně: `var napeti2 = Pm("/napeti2");`

`PmNumber` tedy obsahuje pouze jedinou proměnnou, cesta k objektu je proto jednodušší.

Z předchozích odstavců Vás možná napadne: proměnná Promotic a proměnná časovače ... není to vlastně to samé? NE NENÍ! A je velmi důležité, aby jste pochopili proč, a tedy co se děje v časovači.

Proměnné Promoticu vytváří Promotic pro účely sledování nějaké veličiny v průběhu běhu vizualizace. Tento typ proměnných vytvoříme buďto jednotlivě pomocí objektů jako je `PmNumber` nebo `PmString` a nebo hromadně pomocí proměnných vytvořených uvnitř objektu `PmData`. Tyto proměnné jsou tzv. *persistentní*. To znamená, že zůstávají v paměti po celou dobu běhu aplikace. To je extrémně důležité, protože do těchto proměnných ukládáme „živá“ data z technologie, kterou vizualizujeme.

Oproti tomu proměnná deklarovaná uvnitř časovače (řekněme proměnná `JavaSkriptu` nebo časovače) není *persistentní*. Taková proměnná bude proto fungovat pouze uvnitř tohoto časovače. Prakticky to bude fungovat tak, že časovač se spustí, vytvoří se proměnná a připojí hodnota nebo třeba objekt, se kterým se pak dále ve skriptu časovače pracuje. Skript dojde na konec (vykoná všechny instrukce). Naš skript má nějakých 17 řádků, nebude tedy běžet dlouho. Až skončí všechny proměnné deklarované pomocí `var` se smažou s paměti.

Časovač pracuje tak, že skript spouští v nastavené periodě - v našem případě 1x za sekundu. Tzn. že výše uvedený skript se provede každou sekundu od spuštění aplikace až po její konec. Tím myslíme celý cyklus - vytvoření proměnné, její nastavení, použití a následné uvolnění z paměti. Tyto proměnné tedy NEJSOU *persistentní*.

A to je také důvod proč potřebujeme více typů proměnných a proč je potřebujeme propojovat.

Na řádce 5 vytváříme proměnnou `n`, do které generujeme náhodné číslo v intervalu `<0; 1>`. Toto používáme pro simulaci náhodného kolísání napětí v síti. Všimněte si, že `n` je pouze proměnná JS, není *persistentní*. Na řádcích 7 - 11 provádíme vyhodnocení hodnoty této proměnné. V případě, že `n > 0,5`, pak do hodnoty `napeti` přičtu 1.

Tedy `napeti.Value++`; je ekvivalentem `napeti.Value = napeti.Value + 1`;

Všimněte si, že s proměnnými JS pracujeme jinak, než s připojenými proměnnými Promoticu. Výsledkem připojení objektu Promoticu pomocí funkce `Pm` je objekt, nikoliv jednoduchá proměnná. Proto, když potřebuji pracovat s připojenou proměnnou, potřebuji volat konkrétní vlastnost tohoto objektu. V našem případě nás zajímá hodnota, tedy vlastnost `Value`. `napeti.Value` tedy znamená vlastnost `Value` objektu `napeti`.

Na řádce 10 pak máme dodatečnou podmínku, která naši simulaci kolísání drží trošku na uzdě. Zajišťuje, že hodnota napětí v simulované elektrické síti nemůže poklesnout pod 0.

Na řádcích 13 - 17 je pak hlavní logika, která vyhodnocuje, zda napětí je v limitech stanovených dolní a horní mezí. Pokud ano nabíjíme baterii, pokud ne tak ji postupně vybíjíme.

K výše uvedenému ještě několik upřesnění. JS je senzitivní na velikost písmen, proto `napeti.value` není to samé, co `napeti.Value`.

Dále všimněte si příkazů - obvykle končí znakem středníku. Výjimkou jsou podmínky (a také námi vytvořené funkce a cykly, které ale v našem příkladě nemáme), které ohraničují složenými závorkami a ne středníkem.

Na řádce 13 jsme také použili trošku složitější podmínku. Vyhodnocujeme zda napětí v síti je větší než dolní mez a menší než horní mez. K tomuto účelu používáme logickou spojku `and`. Pro ni JS používá `&&`. Pro úplnost logické `or` v JS je `||`.



Java Skript

Pro začátečníka nemusí být práce s JS úplně příjemná. Dnes ale máte k dispozici nástroje, které Vaši předchůdci k dispozici neměli - AI.

AI můžete použít k vysvětlení funkce skriptu, pokud Vám něco z předchozího výkladu nebylo jasné a při výuce jste se z nějakého důvodu nechtěli zeptat. Může Vám ale pomoci také s návrhem Vašeho skriptu.

Jenom pozor, AI v současnosti, bez ohledu na výrobce, má tendenci k halucinacím. Navržený skript proto nemusí fungovat - může Vás ale třeba nasměrovat správným směrem.



Ladění

Odladění může být problém. Promotic poskytuje test, který můžete spustit kliknutím pravým tlačítkem myši v těle skriptu. V případě, že test selže máte jistotu, že skript nebude fungovat. Zároveň Vám test řekne, kde se ta chyba nachází a v čem spočívá. Tuto informaci můžeme použít k identifikaci a odstranění chyby. Takto ale jsme schopni najít pouze syntaktické chyby, nikoliv chyby v logice programu.

Pro odhalení logických chyb doporučujeme postupovat po malých částech a sledovat, že skript skutečně dělá, co čekáte. Pokud ano, pokračujte dál, pokud ne budete vědět, že problém je v těch řádcích které jste přidali. Problém se Vám tedy bude lépe hledat.



UPS

Implementujte příklad s UPS v systému Promotic, pokud jste tak ještě neučinili.

Pokuste se doplnit některé věci navíc proti původnímu příkladu, např.:

- bitmapu s nějakou UPS
- slovní signalizaci dodávek z elektrické sítě a z baterie
- apod.



Kontrolní otázky

1. Implementujte světelnou signalizaci nabití baterie.
2. doplňte dynamicky měnící se text popisující stav baterie (např. vybitá, téměř vybitá, ...)
3. implementujte skrývání (a objevování se) prvků ve vizualizaci pro signalizaci některých stavů (experimentování s viditelností).
4. navrhnete vlastní řešení zvoleného problému, doplňte jej krátkým textovým komentářem a odevzdejte jej jako semestrální projekt.
5. podle toho jaký programovací jazyk jste zvolili pro oživení časovače - vytvořte variantu projektu využívající druhý programovací jazyk. Zkuste zjistit, který z jazyků Vám lépe vyhovuje. (a toho se pak držte :-)

Kapitola 4

Automatizace - otázky bezpečnosti



Náhled kapitoly

V minulých kapitolách jsme se do určité míry seznámili s funkcí jednotlivých zařízení odpovědných za automatizaci technologií. V této kapitole z těchto znalostí vyjdeme a zaměříme se na bezpečnost těchto zařízení a také možné dopady výpadků nebo ztráty kontroly nad nimi.

Po přečtení této kapitoly budete vědět

1. jaké bezpečnostní problémy jsou typické pro v současnosti používané automatizační systémy
2. jaká řešení, z hlediska architektury sítě, lze zvolit pro minimalizaci rizik spojených s jejich použitím
3. jaké útoky byly prakticky realizovány a jaké měly následky



Čas pro studium

K prostudování této kapitoly budete potřebovat přibližně dvě hodiny.

V předchozích kapitolách jsme se seznámili s různými typy zařízení, které se používají pro automatizaci. Z hlediska bezpečnosti z toho můžeme vypíchnout především následující podstatné momenty:

- automatizace různých procesů v průmyslu (a nejen v něm) je natolik rozšířená, že lze říci že naše civilizace je na nich existenčně závislá - automatizováno je vše od výrobních procesů, systémů kritické infrastruktury až po ovládání kotle popř. klimatu v kancelářských budovách i rodinných domech
- dlouhodobé použití nasazovaných technologií (délka doby užití takových technologií) je obvykle delší než délka podpory ze strany výrobce
- změny v očekáváních o způsobu použití takových technologií pak nutně nemusí odpovídat výrobem předpokládanému užití → *riziko*
- SCADA systém je obvykle provozován na standardním PC s operačním systémem Windows - je proto zranitelný běžnými útoky, stejně jako jiné běžně používané počítače připojené k síti
- typické je dálkové řízení
- některé informace získávané nebo zpracovávané v automatizační části sítě organizace je nutné sdílet i do běžné počítačové sítě organizace.
- ... pravděpodobně bychom dokázali vymyslet také další příklady.

Výše uvedený seznam představuje východiska, pro podrobné rozebrání problémů ale také způsobu ochrany. Z praktických důvodů je problematika ochrany rozdělena do dvou částí popsanými

v samostatných podkapitolách. V první části budou rozebrány některé *případy narušení bezpečnosti* takových systémů a jejich bezpečnostní implikace. V druhé podkapitole budou rozebrány *možnosti realizace izolace automatizační sítě*, jako základního nástroje pro získání jakési úrovně kontroly nad ní.

4.1 Případové studie bezpečnosti automatizace

Naši exkurzi pro známých případech narušení provozu, přerušení výroby, výpadku různého typu zahájíme v poměrně dávno době - v roce 2000. Tehdy se do čističky odpadních vod v Maroochy Shire, Queensland - Austrálie [74] naboural hacker a opakovaně vypustil miliony litrů nezpracovaných odpadních vod do přilehlých místních parků a také hotelu Hyatt Regency - čímž způsobil poměrně velké škody na majetku a především pak životním prostředím.

Motivem byla pomsta za odmítnutí zaměstnání ze strany města. K provedení útoku využil hacker znalostí získaných v rámci svého původního zaměstnání, kde pracoval jako součást týmu pracujícího na instalaci a také údržbě výše uvedené čističky.

Útočník byl dopaden a následně odsouzen ke dvěma letům odnětí svobody.

2000 - čistička odpadních vod - implikace pro bezpečnost

Útočník byl v tomto případě tzv. *insider*. Disponoval tedy podrobnými znalostmi o způsobu fungování napadeného zařízení a vlastnil dokonce kopii software, který umožňoval dálkové připojení do tohoto zařízení a jeho ovládání.

Útoky insiderů jsou obzvláště nebezpečné, protože využívají standardních způsobů používaných pro běžné řízení technologií. Technicky je takový síťový provoz obtížněji detekovatelný než z vnějšku realizovaný pokus o průnik bez takových znalostí a nástrojů.

Problematická je také rychlost provedení útoku - v případě, že se útočníkovi podaří k řízené technologii připojit, může začít přímo ovládat část nebo dokonce celou technologii. Běžný útočník ale postupuje z vnějšku dovnitř. Nejprve musí řešit způsob připojení a následně způsob jakým se zmocní kontroly nad technologií - tato kontrola proto obvykle není úplná, zejména pokud útočníkem je jednotlivec s omezenými zdroji pro realizaci útoku.

Problémem z hlediska zabezpečení byl v tomto případě právě přístup - řízení by mělo být možné realizovat pouze z předem stanovených zabezpečených pracovních stanic. Obecně bychom mohli tento problém označit jako problém **bezpečnosti přípojných bodů**. Útočník by ideálně neměl mít vůbec možnost se připojit do automatizační sítě, natož v ní cokoli dělat.

Jako podpůrný prostředek bylo možné použít metod asymetrického šifrování - konkrétně požadavek na použití ověřeného certifikátu k připojení, vztaheného k osobě operátora. Po odchodu pracovníka lze takový certifikát revokovat, pracovník by pak ztratil možnost se vzdáleně připojit, jelikož by nebyl schopen se do systému úspěšně autentizovat.

Tento způsob má ale také některá úskalí - před revokací certifikátu je potřeba zajistit, aby se k systému mohl připojit alespoň jeden „oprávněný“ operátor - v opačném případě by mohlo dojít je ztrátě kontroly nad řízenou technologií.

Otázkou také zůstává, zda by takové opatření bylo v tomto případě účinné. Ze zdrojů totiž nevyplývá, zda v době realizace útoku byl útočník stále ještě zaměstnancem firmy realizující správu automatizace čističky a tedy zda měl (nebo neměl) z titulu své funkce v době útoku ještě možnost dálkového přístupu. Pokud ano, pak by použití asymetrické kryptografie samo o sobě tomuto útoku nezabránilo.

Zajímavým případem infekce červem **Slammer**¹ [76] v roce 2003. Tento červ napadal MS SQL server, kompromitované servery pak generovaly ohromné množství síťového provozu, které doslova zahltilo počítačovou síť, ve které se napadený server nacházel.

Ačkoliv většina databázových serverů spravovala data běžných informačních systémů byly znamenány i jisté dopady, které rámec běžného IT přesahují. Vyřazena byla např. činnost několika aerolinek [24], což vedlo k nutnosti zrušení řady letů. V tomto případě nebyla narušena bezpečnost leteckého provozu, lety ale nebylo možné odbavovat z důvodu nedostupnosti informací ze systému evidujícího letenky.

¹známý též pod jmény W32.SQLEXP.Worm, SQL Slammer Worm [ISS], DDOS.SQLEXP1434.A [Trend], W32/SQLSlammer [McAfee], Slammer [F-Secure], Sapphire [eEye], W32/SQLSlam-A [Sophos].

V atomové elektrárně Davis-Besse v Ohiu (USA) [68] způsobil Slammer výpadek bezpečnostního monitorovacího systému elektrárny. Je potřeba dodat, že elektrárna byla v to době odstavena z důvodu probíhající zásadní rekonstrukce a tak nedošlo k narušení jaderné bezpečnosti, byť i tak je tento incident hodnocen jako obzvláště závažný.

2003 - Slammer, diskuze následků a ochrana

V obou případech infekci způsobila nedbalost administrátorů MS SQL serveru postižených firem, kteří včas neaplikovali bezpečnostní záplaty, řešící zranitelnosti, které později Slammer zneužil ke kompromitaci serveru.

Tento případ také jasně demonstruje, že řídicí systémy mohou být (a většinou také jsou) zranitelné vůči útokům, které jsou typické pro „běžné IT“, protože běží na stejných technologiích, alespoň co se operačního systému týče. Proto v tomto případě aplikujeme obdobné postupy zabezpečení jako v běžných počítačových sítích.

Ale pozor, stále se zde bavíme o SCADA systémech a dohledových pracovištích, které bývají formálně umístěny v automatizačních/řídicích sítích. Z tohoto důvodu je opatření potřeba nasazovat velmi opatrně tak, aby neohrozila zbývající systémy na síti. Můžeme tento poznatek také obrátit - opatření, která realizujeme v takové síti, by měla umět v takovém prostředí fungovat.

Z opatření, která běžně můžeme nasazovat lze vybrat např.:

- aktualizace operačního systému a instalovaných programů
- minimalizace počtu instalovaných komponent operačního systému a dalších programů instalovaných na daném zařízení (instalace pouze nezbytného minima pro fungování)
- správa uživatelů
- filtrování síťového provozu

V případě červa Slamer byl problém v tom, že administrátoři postižených systémů neaplikovali včas záplatu databázového systému MS SQL Server, ačkoliv v době začátku šíření červa byla k dispozici již několik měsíců a to bezplatně. Administrátoři postižených systémů tedy v tomto případě hrubě podcenili riziko plynoucí ze zranitelnosti opravované záplatou. V důsledku toho zůstal systém zranitelný, čehož pak využil autor červa.

Alternativně bychom mohli říci, že postižené organizace měly chybně zvládnutý *systém managementu rizik*, který jim buď neumožnil identifikovat existující zranitelnost provozovaných systémů, nebo sice umožnil zranitelnost identifikovat, ale vedl k chybnému nastavení priorit stran jejího řešení.

Organizace musí neustále vyhodnocovat v jakém stavu se nachází všechna aktiva, která využívají. Existuje řada služeb, které umožňují takové informace získat z jednoho místa pro různé systémy - viz např. Secunia [73] a řada dalších. Tyto informace je potřeba vyhodnotit a přijmout případně opatření k minimalizaci rizik z něj vyplývajících.

Ve většině případů mají tato opatření charakter buďto aplikace záplaty opravující zjištěnou zranitelnost nebo změnu konfigurace systémů která minimalizuje možnost jejího zneužití, zranitelnost samotná ale zůstává přítomna.

Jako příklad druhé možnosti nám mohou posloužit zranitelnosti Spectre [44] a Meltdown [49] čipů společnosti Intel. Intel vydal sice nové verze tzv. mikrokódu pro své čipy posledních dvou generací, podobně firmy odpovědné za vývoj jednotlivých operačních systémů vydaly pro své systémy záplaty. Tato opatření ale neřeší fyzický problém (konstrukční vadu) způsobující výše uvedené zranitelnosti. Zveřejněné opravy tak pouze znesnadňují jejich zneužití.

Jak vidno, je povaha hrozeb poměrně široká. Pro ochranu musíme zajistit, aby detekční systém byl dostatečně robustní tak, aby byl schopen takové zranitelnosti detekovat a personál zodpovídající za údržbu musí být připraven takto identifikované zranitelnosti řešit ať už aplikací záplat nebo nějakého dočasného opatření, které znesnadňuje možnost zranitelnost zneužít.

V případě jaderné elektrárny Davis-Besse hrál roli také další faktor - *působení třetí strany*. Této problematice se ale budeme věnovat o trochu později v souvislosti s červem Stuxnet.

Necílené infekce malware jsou ve skutečnosti u automatizačních systémů poměrně běžné. V roce 2005 infekce virem Zotob si vynutila hodinovou odstávku v třinácti závodech společnosti Daimler-Chrysler [70]. Škody se v tomto případě odhadují poměrně obtížně, ale práci muselo přerušit okolo 50-ti tisíc zaměstnanců společnosti.

K infekci došlo byť automatizační síť byla za firewalllem. Pravděpodobným vstupním bodem infekce bylo zapojení virem napadeného notebooku do automatizační sítě.

Ačkoliv následky takových útoků jsou citelné, pokud je útok cílený, bývají následky podstatně horší. Např. v květnu 2017 (12. - 19. května) byl zaznamenán útok ransomware WannaCry [79] na Britský **National Health System (NHS)**. Podle šetření **National Audit Office (NAO)** [57] byla ransorwerem WannaCry v nějaké formě postiženo 81 zdravotních zařízení (z celkového počtu 236 zařízení zapojených do systému NHS). Přitom 37 zařízení bylo postiženo přímo, dalších 44 nepřímo tak, že preventivně odstavily část svých systémů z provozu, aby se vyhnuli infekci.

V důsledku útoku muselo být přeloženo okolo 6 900 konzultací, vyšetření nebo zákroků. Některé odhady pak hovoří o tom, že celkový dopad byl ve skutečnosti ještě větší, neboť tyto přeložené úkony by pravděpodobně vyvolaly další - doplňující vyšetření, konzultace specialistů apod.

Následky útoku přitom mohly být ještě podstatně větší, pokud by 19. května 2017 nebyl útok přerušen - útočník použil zabudovaný „kill switch“ pro přerušení útoku.

Tento typ útoků je známý také z prostředí ČR. Např. v roce 2019 mu čelila Benešovská nemocnice, kde si virus vyžádal přerušení činnosti na dva týdny. Postiženy byly také další firmy, jako např. OKD, Olomoucký magistrát a řada dalších. V případě OKD naštěstí následky vzhledem k lépe řešenému modelu bezpečnosti nebyly takové a obnova činnosti po infekci tak byla velmi rychlá.

Infekce i v tomto případě využívala známých zranitelností operačního systému Windows. Situace v tomto případě byla komplikovaná tím, že řada přístrojů používaných ve zdravotnictví vyžaduje ke své práci specifickou verzi operačního systému, pro které již nemusí být nabízeny aktualizace. Bezpečnostní situace ve zdravotnictví je tak velmi podobná situaci např. v průmyslu.

Útok ransomware WannaCry na zdravotní systém Velké Británie vedl ke změnám v požadavcích na zabezpečení, provoz a management IT ve zdravotnictví.

Pro úplnost ještě můžeme zmínit také poslední vlnu opět ransomware WannaCry, která postihla počátkem srpna 2018 výrobní procesy v společnosti **Taiwan Semiconductor Manufacturing Company (TSMC)** [47] a způsobila jejich několikadenní výpadek. Škody v tomto případě jsou odhadovány na přibližně 170 mil. USD.

Odlišná příčina vyřadila řídicí systém jaderné elektrárny Browns – Ferry (USA) [69]. Zde byl zaznamenán v roce 2006 souběžný výpadek dvou oběhových čerpadel (hlavního a záložního). Výpadek způsobil nestandardní síťový provoz v automatizační síti elektrárny. V tomto případě nestandardní provoz ale nebyl způsobem působením viru nebo aktivním útokem hackera, ale prostým provozem celého systému. Řídicí systém oběhových čerpadel se nebyl schopen s touto situací vypořádat a tak reaktor elektrárny bylo nutné odstavit manuálně.

Problém byl následně vyřešen restartem řídicího systému, čímž byla opětovně získána plná kontrola nad systémem.

Přesná příčina nebyla nikdy pro tento případ odhalena - lze se ale dohadovat, že vlivem kombinace nepříznivých událostí se řídicí systém dostal do stavu, se kterým se nebyl schopen vypořádat, a který také zabránil dálkovému zásahu operátora.

Projevila se tak chyba (zranitelnost) v zařízení, která za normálních okolností zůstávala latentní. Obě oběhová čerpadla byla pořízena ve stejnou dobu od stejného výrobce. To je standardní postup, který je vysoce efektivní z hlediska nastavení a obsluhy, má však jeden zásadní problém. Stejná zařízení jsou náchylná ke stejným problémům. Proto nepříznivá kombinace faktorů která zapříčinila výpadek hlavního čerpadla způsobila taktéž výpadek jeho zálohy.

Řešení v tomto případě jsou poměrně problematická. Lze uvažovat o pořízení odlišných záloh (např. od jiného výrobce), aby se zvýšila pravděpodobnost, že záložní systém nebude postižen stejným problémem jako hlavní systém. Realizace této možnosti je ale poměrně obtížná a bude mít negativní dopady na ekonomiku celého řešení a to jak z pohledu pořizovací ceny, tak z pohledu následné údržby takového řešení. Ovládání v takovém případě je nutno řešit samostatně, což prodrazuje celé řešení.

Co by mělo být bezpodmínečně zachováno je alternativní možnost řízení na místě včetně možnosti manuálního řízení. Přítomnost manuálního řízení umožní v případě výpadku dálkového řízení kontrolovat technologii, byť tato kontrola nebude ve stejné kvalitě jako třeba kontrola dálková.

Pro možnost manuálního řízení je potřeba uvažovat s jistou časovou prodlevou od vzniku situace, po dobu identifikace její příčina a okamžikem, kdy se odpovědný pracovník dostane na místo, aby mohl začít manuální řízení využívat. U rozsáhlých sítí pak může tato prodleva být i poměrně dlouhá (např. v hodinách). Sítě tohoto typu naštěstí jsou poměrně odolné proti náhodnému výpadku svých jednotlivých uzlů (ve smyslu vypadne náhodný uzel) a tak takové výpadky obvykle nevedou k rozsáhlejšími výpadkům v síti jako celku.

Teoretické zdůvodnění toho, proč tomu tak je, je možné nalézt např. v Lewis [48]. Bohužel v těchto skriptech není prostor k tomu, abychom se problematice sítí a šíření selhání v nich věnovali větší prostor.

Důležitý je také ještě jeden moment a to samotný charakter síťového provozu v automatizační síti. Právě nestandardní síťový provoz způsobil problém, ale co přesně je nestandardní provoz z hlediska zařízení, která jsou stará třeba 10 nebo více let?

Z tohoto důvodu jsou v automatizačních sítích pouze zřídka instalovány nástroje pro automatické mapování sítě, popř. systémy **Intruder Detection System (IDS)/Intruder Prevention System (IPS)**, které jsou jinak běžně v podnicích využívány pro zlepšení bezpečnosti na síti. Pro velmi stará zařízení může být takový síťový provoz něco, s čím se nebudou schopny vypořádat.

2014 - Stuxnet - první kybernetická zbraň ... možná

Na závěr podkapitoly jsme si ponechali opravdovou lahůdku - červa Stuxnet, který působil mezi léty 2007 - 2010. Tento červ má několik prvenství:

1. jedná se o první zaznamenaný případ, kdy byl nějaký malware zacílen přímo na řídicí systém
2. napadená síť nebyla připojena k Internetu
3. Stuxnet byl schopen fyzicky ničit části napadené technologie
4. Stuxnet byl pravděpodobně vytvořen (nebo jeho vznik byl podporován) nějakou vládou

Stuxnet [31] byl specifický svým zaměřením. Kompromitoval počítače s operačním systémem Windows. K tomu využíval 4 v době útoku v IT komunitě neznámé zranitelnosti operačního systému. Při infekci byl ale červ obzvláště vybíravý - kontroloval lokaci ve které je spouštěn a také technologii, která je v automatizační síti přítomna.

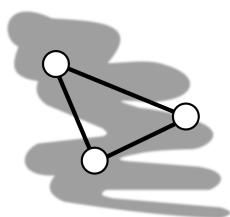
Pozdější analýzou binárního kódu červa a přítomných omezení bylo zjištěno, že hlavním cílem červa byla pravděpodobně zařízení v Natanz v Íránu, které sloužilo k obohacování uranu. Hlavním cílem bylo přitom pravděpodobně napadené PLC automatů v síti ovládající centrifugy na obohacování uranu. Změnou rychlosti rotace centrifug bylo dosahováno velmi rychlého fyzického opotřebení a následného zničení těchto centrifug. Operátorům přitom kompromitovaná technologie hlásila, že vše funguje tak, jak má.

Tímto způsobem bylo zničeno více než 2 000 centrifug a na to, že příčina hromadných selhání by mohla být externí se přišlo až v průběhu kontrolní mise Mezinárodní agentury pro atomovou energii.

Přestože zařízení Natanz bylo pravděpodobně hlavním cílem červa, byla přítomnost Stuxnetu zaznamenána také v dalších zařízeních na blízkém východě, ale také v jihovýchodní Asii.

Atypická struktura červa a také nástroje, které byly použity pro jeho přípravu a také cílení naznačují vysokou profesionalitu tvůrců červa. To jsou pravděpodobně hlavní důvody, proč se obecně předpokládá že jej vytvořil nebo jeho tvorbu sponzoroval nějaký stát nebo státy. Často se přitom spekuluje o USA a Izraeli.

Z pochopitelných důvodů, alespoň v nejbližší době zůstanou podobné úvahy na úrovni spekulací, protože oficiálního potvrzení se pravděpodobně nedočkáme.



Stuxnet - další informace

O Stuxnetu je ale v současnosti k dispozici opravdu velké množství informací. Velmi poutavě popsal celou anabázi analýzy a také dopadů Stuxnetu popsal Kim Zetter ve své knize *Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon* [86].

Na Stuxnet je také možné pohlížet z jiného pohledu - z pohledu platformy, na které byl postaven. Unikátnost návrhu je totiž také v silné modularitě kódu. Bezpečnostní výzkumníky proto napadlo, že by mohli existovat také jiní červi, kteří sdílejí podobné vlastnosti/moduly. Později se takové vzorky skutečně podařilo odhalit.

Stuxnet by pak mohl být považován za útočnou variantu červa, kterou by mohly doplňovat varianty zaměřené např. na špionáž. Z rodiny Stuxnet můžeme zmínit např. červy DuQu, Flame a Gaus, jejich funkčnost je popsána níže.

Červ DuQu [77] byl používán přibližně ve stejné době jako Stuxnet, byl ale zaměřen jiným způsobem. Analýzy antivirových společností prokázaly silné sdílení použitých modulů mezi oběma červy a také stejný (vysoce profesionální) styl, jakým byl napsán také Stuxnet.

Původní DuQu sloužil pro shromažďování informací o průmyslových řídicích systémech. Většina zachycených vzorků byla z oblasti blízkého východu.

DuQu používal poměrně chytrý způsob infekce počítačů prostřednictvím zlovolně upravených běžných dokumentů (např. v PDF, nebo DOCX formátu) a zneužitím existujících zranitelností v jejich prohlížečích.

Jistou vazbu na Stuxnet mohou mít také červi Flame a Gauss. Flame (známý též pod názvem W32.Flamer) [78] otevíral zadní vrátka do systému, umožňující získat vzdálenému útočníkovi plnou kontrolu nad systémem, obsahoval také keylogger, schopnost zachytávat změny na obrazovce a také umožňoval využívat mikrofon a webovou kameru, pokud byly připojeny k počítači pro sledování dění v místnosti.

Malware Gauss [75] se pak zaměřuje primárně na sektor bankovníctví - opět se tedy jedná o špionážní software. Většina zaznamenaných infekcí pak pochází z blízkého východu, především pak Izraele, Sýrie, Libanonu a Palestiny.

Kaspersky se na základě svých analýz výše uvedených malwarů domnívá, že se jedná o jednu rodinu škodlivého kódu od stejných tvůrců. Svá tvrzení pak opírá o použití stejných nebo podobných modulů v červech. Takové závěry ale nejsou všeobecně sdíleny. Např. Symantec argumentuje, že sdílení modulů může naznačovat dočasnou spolupráci různých skupin na vývoji kódů, moduly, kterými se ale červi liší jsou napsány velmi odlišným způsobem. Symantec z toho vyvozuje, že Stuxnet a DuQu byl vyvinut jednou skupinou a Flame a Gauss druhou, nezávislou skupinou.

Existuje také alternativní vysvětlení vycházející z modularity červů. Technicky je možné tyto moduly ze zachyceného vzorku červa vyjmout a použít je společně s nově vyvinutými moduly. Vzniklý červ by tak sice obsahoval některé stejné moduly jako např. Stuxnet, ale vznikl bez kooperace mezi vývojáři těchto červů.



Stuxnet a obdobné kódy - aneb proč bych se to měl učit?

Po přečtení výše uvedených informací o různých škodlivých kódech Vás možná napadne: *proč bych se vůbec touto problematikou měl(-a) zabývat? Vždyť tohle je přece záležitost IT!* Svým způsobem má takový přístup pravdu. Je to ve skutečnosti záležitostí IT. Problémem ale je, že IT prorostlo takovým způsobem do všech běžných činností, že takřka vše co děláme je tak trochu záležitostí IT. Prakticky to znamená, že problematiku bezpečnosti nelze v současnosti od IT zcela oddělit!

To neznamená, že Vaše znalosti z IT by měly být nutně nějak hluboké. Berte prosím v úvahu, že IT specialisté mohou mít v oblasti (obecné) bezpečnosti podobnou úroveň znalostí, jako vy v IT. Aby bylo možné dosáhnout určité úrovně bezpečnosti je nutné se odborně „sejít“ někde uprostřed a intenzivně spolupracovat.

Pro realizaci takové spolupráce je ale nutné, aby si obě skupiny alespoň rámcově rozuměly.

Kybernetická válka

Stuxnet je někdy označován jako první reálně vyvinutá a použitá kybernetická zbraň. V souvislosti s tím, se pak stále častěji hovoří o možnosti vedení tzv. *kybernetické války*, jako dalšího způsobu dosahování cílů v rámci konfrontace s nepřítelem.

Zatímco válka je jednoznačně definovatelná (slovník Merriam-Webster [51]) jako *stav obvykle otevřeného a deklarovaného ozbrojeného nepřátelského konfliktu mezi státy, popř. národy*. V této definici jsou velmi důležitá slova *otevřený* a *deklarovaný* konflikt. Jde o to, že válka je skutečně v určitém smyslu pokračováním diplomacie jinými prostředky, jak prohlásil Carl von Clausewitz.

Jinými slovy všechny strany tohoto konfliktu mají vlastní, často otevřeně deklarované záměry, které se v rámci konfliktu snaží dosáhnout. Jednotlivé strany konfliktu proto ve skutečnosti řeší, zda budou v konfliktu pokračovat nebo druhé straně konfliktu dovolí dosáhnout jejích cílů.

Výše uvedené ale v případě vedení konfliktu na kybernetické úrovni není splněno. Současné zkušenosti tak spíše ukazují, že při nasazování takových prostředků je masivně využívána možnost tzv. *věrohodného popření* původu útoku, jelikož provedení analýz zachycených vzorů obvykle nepřináší

samo o sobě věrohodné důkazy umožňující jednoznačně označit viníka. Alespoň ne na úrovni, kdy by z toho z hlediska mezinárodního práva vyplývaly jakékoliv reálné možnosti.

Z toho důvodu např. Thomas Rid [66] tvrdí, že kybernetická válka je ve skutečnosti protimluv. Argumentuje tím, že použití v současnosti známých nástrojů má spíše charakter špionáže popř. zpravodajských her.

Na druhou stranu samotná existence kódů jako Stuxnet vyvolává jisté otázky o budoucnosti bezpečnosti, zejména pak v oblasti kritické infrastruktury.

Jistá východiska z hlediska možnosti **dlouhodobého vyřazení systémů kritické infrastruktury** navrhuje Peterson [64]. Podle něj je v současnosti velmi jednoduché, alespoň po technické stránce, připravit škodlivý kód schopný kompromitovat a následně vyřadit určitý systém. Pro dlouhodobé působení takového útoku je ale zapotřebí aby kompromitovány nebyly jednotlivé systémy, ale systémy celého sektoru KI nebo alespoň jeho podstatné části.

K tomuto účelu by bylo nutné kompromitovat velké množství technicky odlišných systémů, což je opět technicky možné, ale finančně je to natolik náročné, že taková možnost je reálně dostupná pouze pro státy.

Přípravná fáze takového útoku by tak pravděpodobně musela probíhat minimálně měsíce, spíše však roky předem. Před spuštěním samotného útoku by tím ale vznikl prostor pro objevení již kompromitovaných systémů a jejich oprava. Také to znamená, že pokud stát uvažuje o reálné možnosti takto rozsáhlý útok realizovat musí zahájit snahy o kompromitaci takových systémů svého protivníka, byť reálně by útok - ve smyslu vyřazení KI nakonec třeba ani nikdy nebyl realizován.

Nejblíže k plnému, dlouhodobému vyřazení části kritické infrastruktury měl útok na rozvodnou síť Ukrajiny z Vánoc 2015, který způsobil výpadek elektřiny pro přibližně 230 000 lidí na dobu přibližně 6-ti hodin. Některé pozdější studie kódu naznačují, že Ukrajina měla ve skutečnosti štěstí, protože škodlivý kód obsahoval část, která měla fyzicky poškodit rozvodný systém, při jeho manuální obnově činnosti. Vinou chyby v kódu viru se ale tato část nespustila.

Výpadek tak byl řádově v hodinách. V případě fyzického poškození nebo dokonce zničení transformátorů by obnova trvala týdny až měsíce, jelikož tato zařízení jsou vyráběna na míru a jsou zde poměrně dlouhé čekací lhůty.

Výše uvedené má také značné implikace z pohledu „ochrany KI“. Jednotlivé prvky KI jsou totiž obvykle v soukromém vlastnictví. Vlastník pak dle ekonomické teorie s infrastrukturami podniká za účelem maximalizace prospěchu svých vlastníků (stakeholderů). Tento prospěch se obvykle vyjadřuje finančně ve smyslu zisku, vyplacené dividendy, popř. ceny akcií.

Proti obránci ale může stát útočník, který podobnou motivaci nutně sdílet nemusí - pokud útok připravuje stát, pak motiv bude politický. Pro realizaci útoku přitom může stát uvolnit libovolné množství peněz, obránce je však limitován svými disponibilními prostředky.

K této problematice a možnostem organizace ochranných opatření je věnována kapitola 5.

4.2 Způsoby izolace automatizační sítě

Jedním ze základních mechanismů ochrany řídicích systémů je jejich izolace. Tradičně byly takové systémy izolovány zcela - tedy řídicí systém byl zapojen ve zcela samostatné síti bez jakýchkoliv spojení na další síť včetně Internetu. Tento způsob je možno považovat za nejbezpečnější. V angličtině se pak označuje jako tzv. *air gap*. (Tedy že mezi sítěmi neexistuje fyzické spojení.)

Pro pokus o získání kontroly nad takovou sítí pak útočník musí získat fyzický přístup k síti - útok, tedy alespoň v určité fázi musí být proveden fyzicky a na místě.

Zkušenosti s červem Stuxnet ale ukazují, že prostá totální izolace sítě není sama o sobě dostatečná pro zajištění její ochrany. Pro kompromitaci sítě v Natanz byl pravděpodobně použit kompromitovaný notebook nebo médium (např. flash disk) některého z kontraktorů, kteří pracovali v tomto zařízení při jeho výstavbě.

Alternativní scénář kompromitace takových izolovaných sítí vyžaduje fyzicky přistoupit k síti a buďto přímo realizovat útok, nebo do sítě doplnit další zařízení - modem, radiostanice apod., které umožní útočníkovi později do sítě dálkově přistoupit a realizovat zamýšlený útok.

Nevýhodou takového přístupu je fakt, že propojením sítí mohou být dosaženy některé jinak nedostupné synergické efekty, kdy informace z technologie mohou být dále využívány pro efektivnější řízení organizace. Taková síť je pak také jednodušeji vzdáleně spravovatelná. To jsou také důvody proč je úplná izolace takových sítí v současné době spíše minoritní záležitostí.

Ochrana pak musí být realizována dalšími vrstvami obrany sloužícími především k filtraci síťového provozu.

Pro organizace s automatizační sítí oddělenou od zbývajících sítí a Internetu nutnost fyzického přístupu k získání/převzetí kontroly nad technologií znamená, že praktická ochrana musí být založena na zajištění ochrany *vnějšího perimetru organizace*. Vzhledem k tomu, že fyzický přístup k zařízení představuje zároveň také černý scénář i pro systémy a sítě připojené k Internetu, nelze ochranu vnějšího perimetru podcenit ani v takovém případě.

Mezi opatření, kterými takovou ochranu lze realizovat můžeme zařadit např.:

- kontrola vstupu (do střeženého perimetru mohou vstupovat pouze osoby které jsou k tomu oprávněny),
- kontrola činnosti návštěv, kontraktorů apod. v průběhu jejich pobytu v areálu
- kontrola narušení perimetru - kamerové systémy, čidla pohybu, senzory monitorující otevření např. dveří, pochůzková činnost v areálu apod.
- management manipulace se zařízeními - kontrola toho jaká zařízení (ve smyslu např. notebooků) se v areálu používají
- management médií, zejména těch, která jsou vnášena/vynášena z areálu.

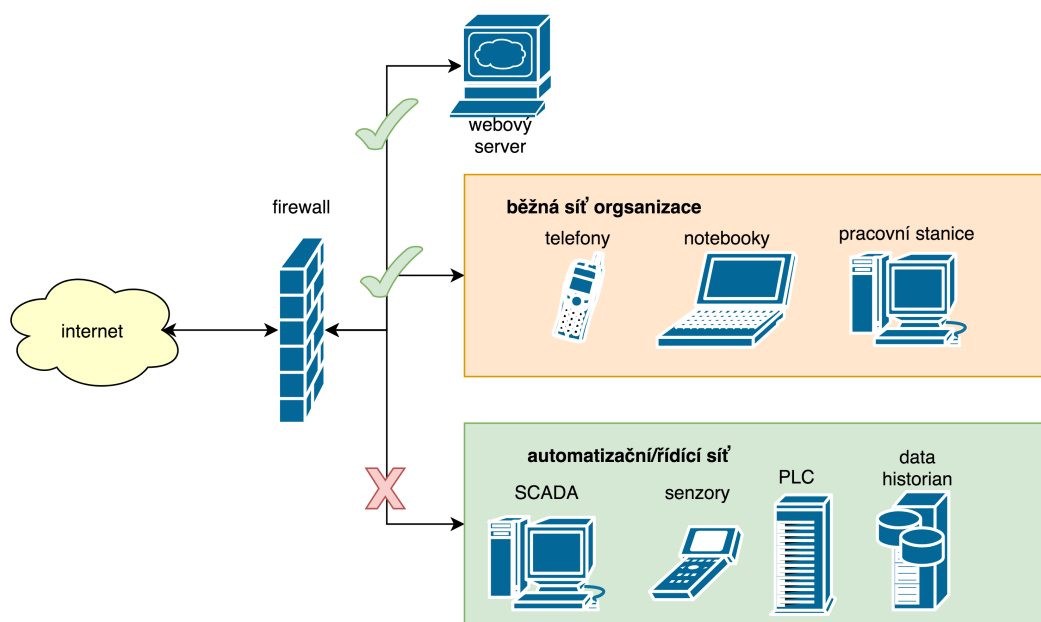
Výše uvedená opatření mají za cíl zvýšit náročnost provedení útoku, popř. prodloužit čas nutný pro jeho realizaci a tak zvýšit šanci, že dojde k jeho odhalení a následně přerušení. Cílem tedy není provedení útoku zcela zamezit - neboť úplné zabezpečení není technicky možné nikdy realizovat.

Z praktických důvodů často nelze ani realizovat úplnou izolaci řídicího systému. V následujících odstavcích se proto zaměříme na různé scénáře realizace *softwarově* řešené izolace řídicí a běžných sítí. Pracovat budeme s následujícími scénáři:

1. ochrana na vnějším perimetru sítě obvykle na bázi firewallu
2. omezení společných částí sítí
 - dual home computer (vybrané stanice mají přístup do různých sítí)
 - dual home server (vybrané servery mají přístup do různých sítí)
3. oddělení sítí směrováním komunikace (pomocí routeru)
4. oddělení sítí pomocí firewallu
5. vytvoření demilitarizované zóny a oddělení sítí dvojicí firewallů

Ochrana na vnějším perimetru sítě obvykle na bázi firewallu

Tuto formu ochrany bychom si mohli představit podobně jako na obr. 4.1.



Obrázek 4.1: Bezpečnost řešená na vnějším perimetru sítě firewalllem

Vnitřní síť organizace je v tomto případě oddělena od internetu firewalllem. Firewall pracuje na bázi protokolu TCP/IP. Umožňuje nastavením pravidel specifikovat systémy (IP adresami, popř. jejich

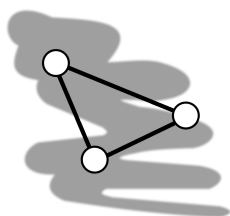
rozsahy) a služby (porty), jejichž komunikace bude firewallem propuštěna. Nepovolenou komunikaci pak firewall zastaví.

Na obr. 4.1 tak na Internetu mohou komunikovat všechna zařízení v běžné síti a také webový server, oproti tomu komunikaci automatizační (řídící) sítě je zabráněno.

Všimněte si, že výše uvedený scénář řeší pouze vnější hranici sítě. Komunikaci mezi běžnou počítačovou sítí a sítí automatizační tak nic nebrání. Útočníkovi tak může stačit kompromitovat jakékoliv zařízení v „otevřené“ části sítě a pak může pokračovat neomezeně v útoku do zbytku sítě i té, kde byla komunikace s vnějším světem na firewallu zakázána. V této fázi je útoku, je tak ochrana poskytovaná firewallem již překonána.

Úroveň ochrany, kterou toto řešení poskytuje, tak nelze přeceňovat. Z toho důvodu je tento scénář ochrany volen samostatně pouze zřídka a to zejména v případech kdy oddělení sítí nemá vzhledem k její velikosti smysl (např. mikro firmy nebo malé firmy).

Zároveň ale tento způsob ochrany poskytuje dobrý základ, který lze doplnit dalšími opatřeními vedoucími k lepší kontrole síťového provozu mezi jednotlivými částmi sítě. **Další scénáře proto předpokládají existenci tohoto řešení - tedy, že je nějakým způsobem řešena bezpečnost na vnějším perimetru sítě.**



Firewally nové generace

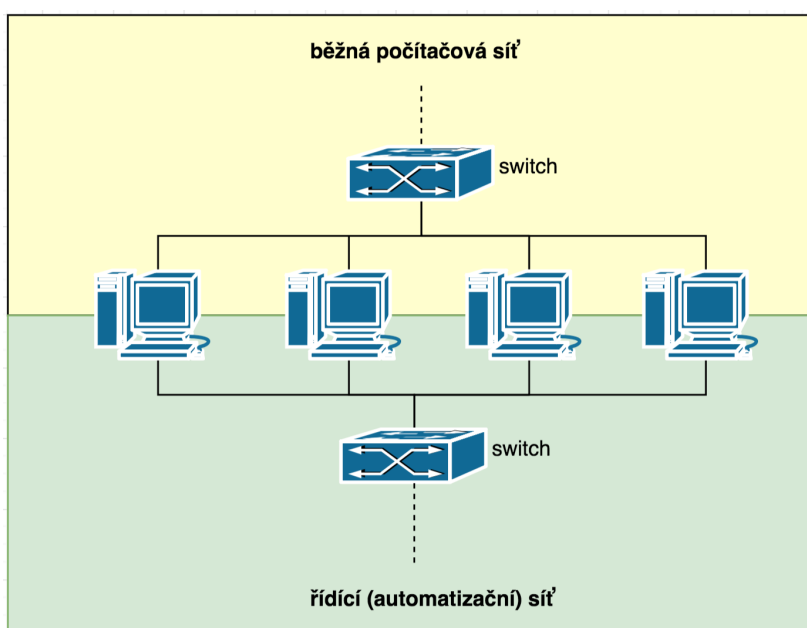
V posledních letech získávají na popularitě tzv. *firewally nové generace* (NG). Tento typ firewallů kromě schopnosti filtrovat síťový provoz umožňuje také provádět tzv. *hloubkovou analýzu paketů*, které přes něj procházejí. Firewall tak má k dispozici nástroje podobné systémům **IDS/IPS**, což mu umožňuje lépe identifikovat náznaky průniků do sítě organizace, popř. se takovým útokům přímo bránit.

Pokud má být filtrován síťový provoz zařízení v automatizační síti, musí firewall mít schopnost pracovat s protokoly, které se na takové síti používají. Podpora takových protokolů není přítomna ve všech firewallech tohoto typu.

Dual home computers

Tímto ne úplně dobře do češtiny přeložitelným označením rozumíme pracovní stanice, které jsou připojeny zároveň do řídící sítě a také běžné sítě. Připojení do každé ze sítí je přitom realizováno samostatnou fyzickou kabeláží. Tyto počítače tak musí být vybaveny dvěma síťovými kartami.

Schématicky si lze o způsobu realizace vytvořit představu z obr. 4.2.



Obrázek 4.2: Dual home computers

K realizaci je používána běžná hvězdicová architektura sítě. Na obr. 4.2 je to naznačeno připojením počítačů do switchů v jednotlivých sítích.

Toto řešení není příliš praktické. Jeho takřka jedinou výhodou je, že formálně definuje rozhraní mezi běžnou počítačovou sítí a sítí automatizační. Toto rozhraní je tvořeno právě počítači, které jsou umístěny v obou těchto sítích. Z hlediska bezpečnosti je pak klíčové udržení kontroly právě nad těmito počítači.

S realizací tohoto scénáře oddělení sítí je však spojena také řada nevýhod. Toto řešení je poměrně rigidní - ve smyslu jeho změna není úplně snadná. Pokud mají být počítače umístěny ve dvou počítačových sítích, je nutné aby v budově byla vedena dvojí kabeláž. To ale také znamená, že vedené kabeláže nebudou univerzální a umístění těchto počítačů tak nebude v budoucnu možné měnit, aniž by byla taktéž přeložena kabeláž.

Podobná situace je s možností škálování řešení - např. přidávání dalších počítačů s touto konektivitou.

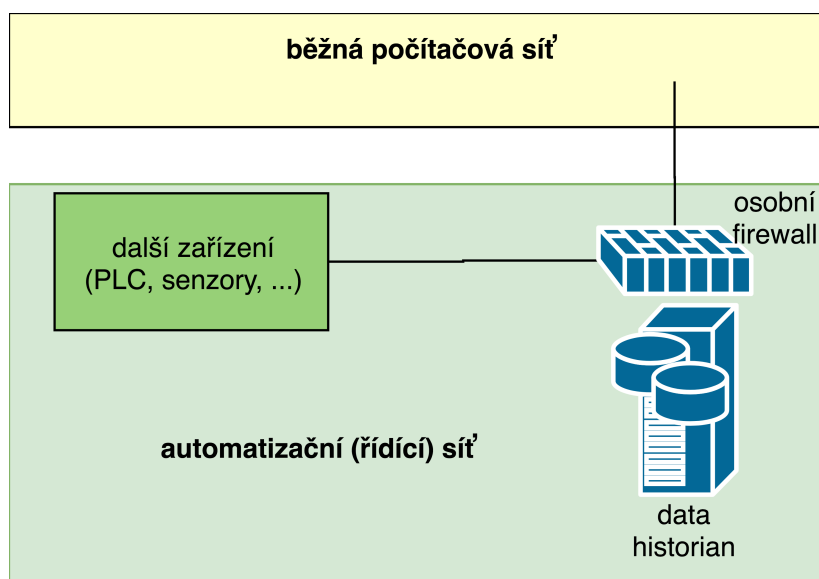
Z hlediska bezpečnosti není přínos tohoto řešení až tak výrazný. Je sice definováno rozhraní mezi sítěmi, ale stále se jedná o běžné počítače, které navíc jsou přímo přístupné z běžné počítačové sítě nebo dokonce Internetu (podle konfigurace firewallu na vnějším perimetru sítě). Tyto počítače tak mohou být samy přímo cílem útoku.

Pokud pak dojde k jejich kompromitaci, otevírá se útočníkovi cesta do chráněné řídicí/automatizační sítě organizace.

Dual home server

Alternativním řešením je neumisťovat do obou sítí jednotlivé počítače ale infrastrukturu, která má být v obou sítích používána. Z automatizační sítě jsou pro řízení organizace cenné informace o stavu technologií. Z předchozích kapitol již víme, že existuje jedno místo, kam se takové údaje často ukládají - *data historian*.

Tento typ řešení by mohl být realizován podobně jako na obr. 4.3.



Obrázek 4.3: Dual home server

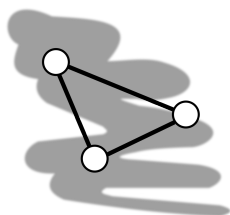
Toto řešení je ve srovnání s umisťováním jednotlivých počítačů do dvou sítí lépe škálovatelné. Data-bázový server bude nejspíše umístěn v serverovně, u které se dá očekávat, že její přesun nebude probíhat právě často. Dále se jedná pouze o jedno místo, jeho zasíťování není proto až tak problematické.

Umístění v serverovně s sebou obvykle nese také zvýšení úrovně fyzické bezpečnosti. Jelikož serverovna má obvykle zavedenu nějakou formu řízeného vstupu. Počet pracovníků s fyzickým přístupem k serveru je tak omezený a proto lépe kontrolovatelný.

Na obr. 4.3 je taktéž naznačena možnost použití osobního firewallu pro řízení síťového provozu serveru. V tomto scénáři se zatím nebere v úvahu možnost použití dedikovaného firewallu (tímto scénářem se budeme zabývat později).

Osobní firewall se od dedikovaného liší tím, že běží přímo na daném zařízení jako program (služba). Funkce firewallu je tak pouze doplňující vlastností systémové vrstvy zařízení. V současnosti většina operačních systémů má přímo v nějaké formě implementovaný osobní firewall, existují ale také samostatné softwarové balíky s pokročilejšími funkcemi, které lze doinstalovat.

Osobní firewall v sobě často obsahuje také některé další funkce zaměřené na zvýšení bezpečnosti použití zařízení. Může se jednat o moduly **Host Intruder Prevention System (HIPS)** sloužící pro identifikaci některých typů průniků do systému. V těchto řešeních jsou často implementovány také možnosti tzv. whitelistingu aplikací (specifikace aplikací, které mohou být spouštěny, popř. mohou komunikovat po síti), sandbox aplikací (omezení možnosti aplikací ovlivnit zbytek systému) a řada dalších.



Osobní firewall pokusy

Některé z osobních firewallů jsou dostupné také bezplatně. Např. *Comodo Firewall* [25] je hodnocen poměrně příznivě a je dostupný bezplatně buď samostatně nebo v balíku s antivirovým programem.

Pokud jsou schopnosti osobního firewallu tak velké, dosahujeme výrazně vyšší úrovně bezpečnosti, ne? Ve skutečnosti tomu tak být nemusí. S použitím osobních firewallů na serverech se pojí řada problémů. Asi největším problémem je, že osobní firewally jsou určeny pro „osobní použití“ - tedy předpokládají, že u chráněného zařízení bude fyzicky sedět osoba, která v případě potřeby zasáhne do funkce firewallu.

Osobní firewally se totiž obvykle „učí“, jaký síťový provoz je možné považovat za bezpečný, které aplikace (a jejich verze) jsou bezpečné apod. V případě, že se firewall setká se situací, pro kterou nemá vytvořeno pravidlo za normálních okolností se uživateli objeví výzva na rozhodnutí, co má firewall v této situaci dělat - např. povolit dané aplikaci možnost komunikovat po síti. Na základě interakce s uživatelem jsou pro činnost firewallu doplňována pravidla - firewall se tedy postupně adaptuje na síťový provoz zařízení, na kterém je instalován.

Představitelná může být také opačná situace, kdy si uživatel všimne, že určitý síťový provoz, který měl přes firewall projít jím neprošel. To jej může motivovat k zásahu do pravidel filtrace síťového provozu, pokud na provedení takového zásahu má dostatečně vysoká uživatelská práva.

U serveru ale za normálních okolností není fyzická obsluha přítomna - většina přístupů a to i administrátorských zásahů je realizována dálkově prostřednictvím na serveru provozovaných služeb. Navíc tato interakce je nárazová a je spojena obvykle s realizací údržby, servisních zásahů apod. Služby poskytované serverem jsou konzumovány klienty pomocí specializovaných aplikací nebo poskytovány přes webové rozhraní. Hlášení, které se zobrazí ve Windows tak pravděpodobně nikdo neuvidí - firewall tak ztrácí možnost se adaptovat.

I osobní firewall lze nakonfigurovat tak, aby se na nic neptal. Sada použitých pravidel je ale v takovém případě obvykle omezenější. Plnohodnotné použití všech (zejména těch pokročilejších funkcí jako je sandbox nebo HIPS) pak často není možné.

Osobní firewally také obvykle nemají implementovanou podporu automatizačních protokolů. Proto je pokročilá funkcionality tohoto typu firewallů v automatizačních sítích téměř nepoužitelná.

Další nevýhodou je to, že k filtraci síťového provozu dochází přímo na zařízení - zařízení tedy přijme packet a komponenta firewallu provede jeho vyhodnocení. Firewall ale může sám obsahovat zranitelnosti, které mohou umožnit útočníkovi tuto vrstvu ochrany (za určitých specifických podmínek) obejít nebo je ji možné dokonce zneužít k tomu, aby byl takový systém přehlcen požadavky a nebyl schopen vyřizovat oprávněné požadavky.

Útoky typu **Denial of Services (DoS)** jsou bohužel těmi nejčastějšími, kterým jsou sítě organizací vystaveny a tak předchozí scénář není nepravděpodobný. Omezení schopnosti data historianu udržet aktuální informace o stavu technologie jsou pak klíčovými z pohledu naší schopnosti udržet kontrolu nad řízeným procesem, jak již víme z předchozích kapitol.

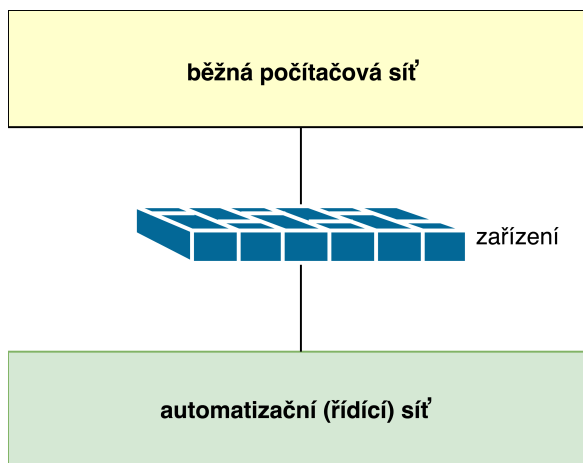
Poslední poznámka - z hlediska funkce tohoto řešení je v zásadě jedno, zda dual home server je umístěn v řídicí síti (jak je tomu na obr. 4.3) nebo zda je umístěn v běžné síti. Řešení (i proces zneužití) by fungovalo analogicky - pouze v obráceném gardu než je znázorněno na obrázku.

Oddělení sítí pomocí ... zařízení

V této části rozebereme zbývající scénáře oddělení sítí, konkrétně scénáře 3 - 5. Všechny tyto scénáře jsou založeny na použití dedikovaných zařízení umožňujících filtrovat provoz mezi jednotlivými sítěmi. Pro řešení oddělení lze použít v zásadě:

- router,
- switch s funkcí správy,
- **firewall** (dedikovaný).

Schématicky je situace znázorněna na obr. 4.4. Na obr. je použit generický název *zařízení*. Umístění zařízení ve všech výše uvedených případech bude stejné, řešení se však bude lišit svou funkcí podle typu zařízení, které se v řešení použije.



Obrázek 4.4: Oddělení sítí pomocí zařízení různých typů

Jednotlivá zařízení se zcela zásadně liší podle svých schopností - ty jsou definovány primárním účelem těchto zařízení.

Hlavní funkcí *routeru* je směrování síťového provozu a to primárně mezi jednotlivými sítěmi nebo segmenty sítě. Z tohoto pohledu se proto router nehodí pro řízení přístupu k jednotlivým zařízením. To samozřejmě neznamená, že router není schopen při směrování provozu jít až na úroveň jednotlivých zařízení, pouze že pro takovou činnost není optimalizován a pokud se používá tímto způsobem je potřeba architekturu sítě přizpůsobit. Takové řešení pak obvykle není nijak elegantní, špatně se udržuje a ještě hůře škáluje.

Podobná situace je se zařízeními typu *switch* - ty se starají primárně o funkci tzv. přepínání (switching) síťové komunikace. Zjednodušeně řešeno přeposílá síťovou komunikaci do cílových portů. V případě tzv. „spravovatelných“ switchů, s jejichž použitím v tomto scénáři počítáme, lze zasáhnout do konfigurace zařízení a ovlivnit způsob, jakým se toto přeposílání děje. V našem případě tedy specifikujeme, která zařízení spolu mohou komunikovat.

Z hlediska použitelnosti switchů, jsou k tomuto účelu používány tzv. L3 switche, které fungují o úroveň výše, než switche běžné. To umožňuje realizovat management spojení na základě IP adres místo MAC adres, které se využívají u L2 switchů.

L2 a L3 v tomto případě odkazují na 2. a 3. úroveň sítě v architektuře ISO/OSI.

K výše uvedenému je potřeba dodat, že ačkoliv jsou L3 switche z hlediska filtračních/routovacích schopností výrazně pokročilejší než L2 switche, jedná se stále o switche, které k tomuto účelu nejsou primárně určeny. Je tak sice možné nastavit, jaká zařízení mohou komunikovat, ale nikoliv služby na kterých mohou komunikovat.

Granulita pravidel řízení síťového provozu na switchích je proto poměrně malá.

Router, ani switch tedy nejsou určeny primárně k filtraci síťového provozu - tento typ použití je tak pouze sekundární a lze jej použít tam, kde se nevyplatí (z různých důvodů) použití firewallu. Firewall oproti tomu je přímo určen k filtraci síťového provozu a proto práce s ním je v tomto směru podstatně jednodušší.

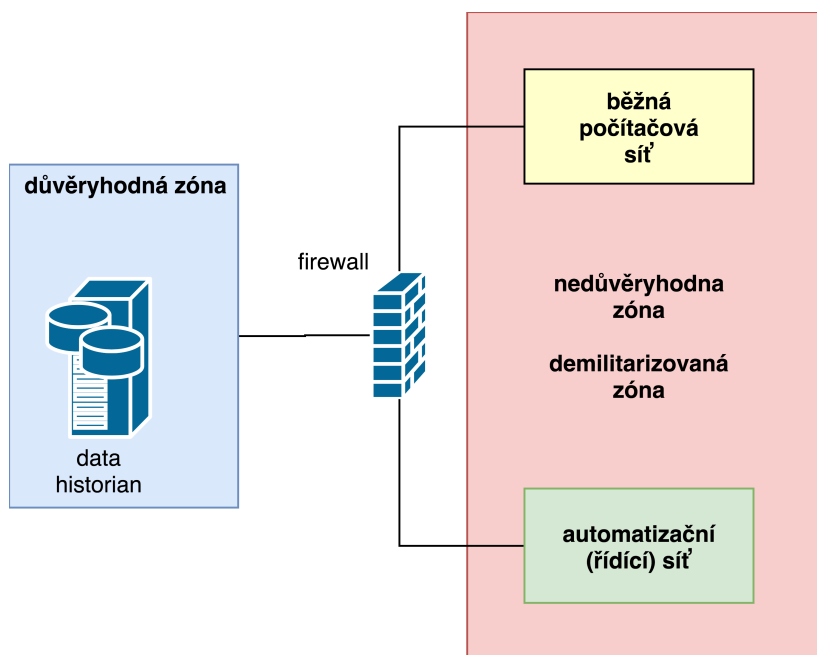
Filtraci je možné provést na úrovni jednotlivých zařízení (identifikovaných IP adresou) nebo rozsahu IP adres. Ovlivnit lze také síťové služby, které mohou v síti komunikovat (prostřednictvím specifikace síťových portů). Firewall je tak ze všech řešení, která jsem zmiňovali v této podkapitole prvním,

které umožňuje zajistit kvalitně izolaci řídicí sítě.

Při použití firewallu nové generace je potřeba vybírat takový, který podporuje v organizaci používané protokoly, viz vysvětlující box.

Podstatná otázka, kterou je potřeba dořešit je kam přesně umístit „sdílená zařízení“. Obr. 4.4 naznačuje dvě možnosti - buďto do sítě automatizační nebo běžné. V obou případech by řešení bylo podobné. Na firewallu by bylo nutné nastavit pravidla pro komunikaci s druhou sítí, zařízení by bylo přístupné pak z vybraných (na firewallu povolených) zařízeních druhé sítě a všech zařízeních sítě, ve které je sdílené zařízení umístěno.

Zvažme ale řešení představené na obr. 4.5. V rámci tohoto řešení vyčleňujeme sdílená zařízení do *důvěryhodné zóny* a na firewallu řešíme komunikaci s oběma sítěmi. Tímto způsobem lze získat mnohem lepší kontrolu nad síťovým provozem adresovaným těmto zařízením a řídit jej.



Obrázek 4.5: Důvěryhodná vs demilitarizovaná zóna

Řešení z obr. 4.5 předpokládá, že žádné ze sítí není možné bez výhrady věřit a z toho důvodu je žádoucí klíčová zařízení vyčlenit do důvěryhodné zóny a přesně stanovit nastavením pravidel firewallu, která zařízení s použitím kterých služeb mohou s nimi komunikovat. Zbývající zařízení uvnitř sítě jsou tedy považována za nedůvěryhodná (potenciálně kompromitovaná). Někdy je zóna nedůvěryhodných zařízení označována jako **demilitarizovaná zóna (DMZ)**.

Úvahy o DMZ nám umožňují jednak technicky oddělit obzvláště kritická zařízení od zbytku sítě, jednak mění způsob, kterým uvažujeme o zařízeních na síti. Ideově pracujeme s možností, že může dojít ke kompromitaci zařízení na síti a řešíme způsob, jak minimalizovat škodlivé následky takového průniku.

U realizace oddělení sítí a zón jediným firewallem je potřeba mít na paměti, že toto řešení je náchylné k selhání člověka při tvorbě pravidel. V ideálním případě by v situaci popsané na obr. 4.5 měla smysl komunikace pouze data historian $\longleftrightarrow$ vybraná zařízení nebo data historian $\longleftrightarrow$ vybraná zařízení v automatizační síti. Naopak smysl nemá přímá komunikace mezi běžnou a autentizační sítí.

Zkušenosti bohužel ukazují, že chybu je často poměrně jednoduché udělat, avšak její zpětné odhalení a odstranění je obtížné. Z tohoto důvodu můžeme uvažovat o alternativním řešení realizace důvěryhodné zóny realizované ne jedním ale dvěma firewally, viz obr. 4.6.

V řešení představeném na obr. 4.6 je pro každou síť použit samostatný firewall. Pravidla komunikace mezi sítěmi zůstávají stejná ve smyslu kdo s kým může komunikovat. Hledání chybných pravidel je ale v tomto případě podstatně snazší. Při řešením jedním firewallem existovala pravidla pracující s oběma sítěmi. V případě použití dvou firewallů ale taková pravidla existovat nemohou. Lze tak jednoduše (třeba i automatizovaně) testovat pravidla na přítomnost takových chyb.

Použitím dvou firewallů na druhou stranu přinášíme do systému určitý nový stupeň komplexity



Zero trust network

Koncept DMZ položil základy odlišnému přístupu k bezpečnosti. Tradiční architektura bezpečnosti pracuje s perimetrem sítě. Vše, co je mimo tento perimetr je považováno za nebezpečné (nedůvěryhodné), vše co je uvnitř je pak považováno za důvěryhodné. Dlouhodobé zkušenosti ale ukazují, že tento přístup nezaručuje ve skutečnosti žádnou reálnou bezpečnost.

V posledních letech se proto prosazuje koncept zero trust network (popř. architecture), která je založena na tom, že zařízením na síti implicitně nevěříme - *zero trust* = *nulová důvěra*. Filozoficky proto se zařízeními pracujeme, jako by už z pohledu bezpečnosti byla kompromitována. Bezpečnostní opatření jsou pak navrhována tak, aby ani kompromitace takových zařízení nevedla k ohrožení důvěryhodnosti a dostupnosti informací, které jsou na síti zpracovávány.

Přesný popis principů tohoto přístupu ale svým rozsahem přesahuje možnosti tohoto textu. Podrobnosti o tomto přístupu můžete nalézt např. v knize Gilman a Barth [34].

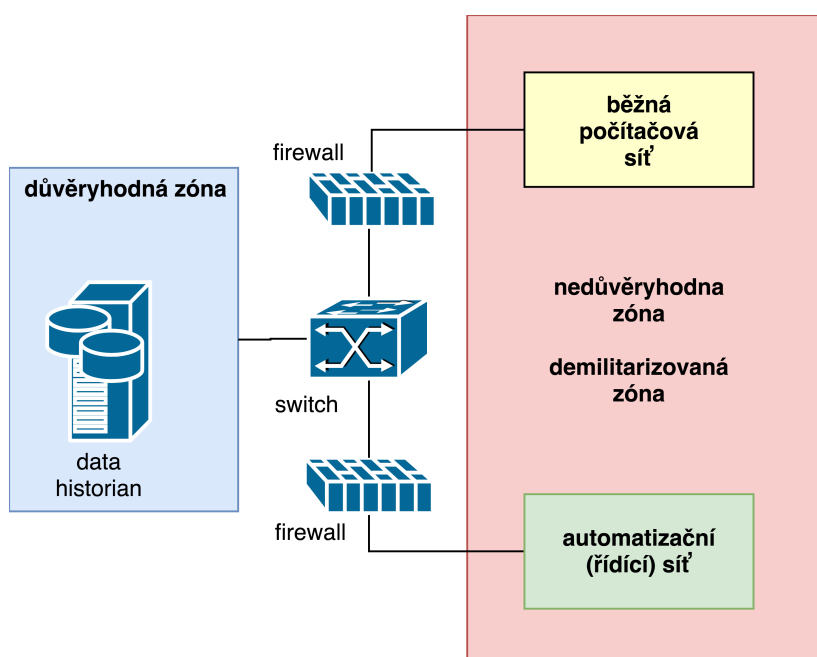
- místo jednoho firewallu musíme zpravovat dva, musíme to zaplatit ... získáme však řešení které je ale také poměrně odolné vůči selhání člověka při konfiguraci těchto zařízení. Aby síťová komunikace prošla mezi běžnou a automatizační sítí nestačí totiž udělat chybu při konfiguraci jednoho firewallu, ale obou firewallů - taková chyba je však mnohem méně pravděpodobná.

Tímto způsobem nebudou odhaleny všechny chyby v konfiguraci firewallů - lze ale takto detekovat chyby, které jsou z pohledu bezpečnosti nejzávažnější a jejich porušení poskytuje útočníkovi přímý vektor k napadení druhé sítě.

Bezpečnost **Industrial Control System (ICS)** pro systémy kritické infrastruktury je založena na variantě tohoto postupu, konkrétně na modelu zón a vodičů (zones and conduits). Komunikace v tomto modelu je omezena pouze na ta zařízení, aplikace a uživatele, u kterých je prokázáno, že mají používat danou funkcionalitu.

Tento přístup je standardizován pomocí ISA 62 443 a v posledních letech nabývá na popularitě všude tam, kde otázky bezpečnosti, resp. řešení bezpečnosti je považováno za prioritu.

Nedá se říci, že by některé z výše uvedených řešení (snad s výjimkou použití firewallu na vnějším perimetru sítě) bylo skutečně univerzálně použitelné. Spíše platí, že čím rozsáhlejší, popř. cennější systém je potřeba chránit, tím sofistikovanější řešení se pro ochranu používají.



Obrázek 4.6: Důvěryhodná vs demilitarizovaná zóna - řešení s použitím dvojice firewallů



Shrnutí

Zajištění bezpečnosti automatizační sítě je poměrně složitá otázka. Zkušenosti z minulosti ukázaly, že ke kompromitaci těchto sítí může docházet náhodně vinou malware, který je primárně zaměřen na běžné počítače a sítě nebo projevem chyby v nastavení nebo v systému zařízení, byly však zaznamenány také příklady, kdy ke kompromitaci takové sítě došlo úmyslně s velkými finančními dopady na provozovatele napadeného systému.

Z hlediska ochrany je potřeba řešit zejména problematiku perimetru (okraje) organizace, resp. její počítačové sítě a to často včetně řešení fyzické bezpečnosti síťových uzlů zneužitelných pro neautorizované připojení k síti. Mezi řešení problémy patří zejména kontrola vstupu, management zařízení a médií vnášených a vynášených z areálu, kontrola pohybu a působení zaměstnanců třetích stran apod.

Velké části problémů lze také předcházet vhodným návrhem a realizací počítačové sítě. V popředí zájmu by především mělo být oddělení síťového provozu sítě určené pro řízení technologií. Způsobů jak takové oddělení realizovat je celá řada, avšak použití **DMZ** je považováno jako jedno z nejbezpečnějších.

Způsob oddělení je obvykle odvozen z velikosti/ceny a očekávání stran bezpečnosti, které jsou na celý systém kladeny.



Kontrolní otázky

1. Co je Stuxnet?
2. Co je DMZ a jak přispívá k bezpečnosti?
3. Co rozumíme managementem médií?
4. Jaký je rozdíl mezi osobní a dedikovaným firewalllem?
5. Co je dual home computer?



Správné odpovědi

1. Stuxnet je škodlivý kód napadající automatizační systémy společnosti Siemens (první takto navržený červ, fyzické zničení centrifug, ...)
2. Demilitarizovaná zóna - nedůvěryhodná zóna. V této zóně je provozována většina IT. Umožňuje nám lépe uvažovat o možných scénářích kompromitace sítě a minimalizace jejich následků na kritická zařízení umístěná v důvěryhodné zóně.
3. Rozumíme jím soubor opatření zaměřených na pečlivou kontrolu všech médií vstupujících a odnášených z areálu s cílem zamezit vnesení škodlivého kódu a vynesení citlivých informací.
4. Dedikovaný firewall umožňuje filtrovat síťový provoz na bázi protokolu TCP/IP, osobní k této funkcionalitě může obsahovat další komponenty jako např. HIPS, sandbox a další, ke své plnohodnotné funkci však potřebuje často interakci uživatele.
5. Jedná se o počítač s fyzicky realizovaným samostatným spojením do běžné a automatizační sítě - možno realizovat pro počítače které mají zpracovávat data udržována na těchto sítích.

Kapitola 5

Ochrana ICS



Náhled kapitoly

V této kapitole se zaměříme na způsob ochrany **ICS** ze strany státu. Konkrétně se budeme zabývat v současnosti platnou legislativou, zejména pak zákonem o *Kybernetické bezpečnosti* a také funkcí různých typů CSIRT týmů.

Po přečtení této kapitoly budete vědět

1. jaká je funkce **Computer Emergency Response Team (CERT)** a **Computer Security Incident Response Team (CSIRT)**
2. co je to stav kybernetického nebezpečí
3. jaké jsou mezinárodní souvislosti řešení kybernetické bezpečnosti



Čas pro studium

Pro prostudování této kapitoly budete potřebovat přibližně dvě hodiny.

V minulé kapitole jsem zejména s některými zaznamenanými incidenty v oblasti kritické infrastruktury pracovali s možností, že za realizací alespoň některých z těchto útoků mohou být buď přímo státy nebo mohou být státy sponzorované. Vzhledem k tomu, že infrastruktura samotná je většinou v soukromých rukou, vzniká nepoměr v dostupných zdrojích z hlediska možnosti financovat útok na infrastrukturu na straně jedné a na druhé straně pak možnosti obrany proti takovým útokům.

Z praktických důvodů nemohou ponechat státy systémy kritické infrastruktury zcela, alespoň z hlediska ochrany, na libovůli vlastníka, popř. provozovatele. Různé státy ale podpůrné ochranné aktivity organizují různě. V následujících podkapitolách se proto zaměříme na některá používaná řešení, jejich možnosti, výhody a nevýhody.

5.1 Typy CERT a CSIRT

CERT je označení týmů určených k reakci na počítačové hrozby. První CERT tým vznikl v roce 1988 na univerzitě Carnegie Mellon pod názvem CERT-CC (CERT Coordination Center). Jeho zaměření bylo poskytování pomoci při problémech s IT.

IT v roce 1988 bylo ale velmi odlišné od IT, se kterým pracujeme dnes. Většina počítačů bylo lokálních často bez připojení do sítě, Internet neexistoval, malware existoval, ale nebyl tak rozšířen jako dnes, zařízení jako chytré telefony ovládané pomocí dotykového rozhraní byly futuristickou vizí která se objevovala ve sci-fi filmech a seriálech (např. Star Trek: Nová generace a další).

Hlavní aktivitou tohoto týmu tak byla obecná podpora uživatelům. Podobný přístup a také označení takových týmů bylo později přejímáno také jinými organizacemi. Univerzita Carnegie Mellon si název CERT zaregistrovala jako ochrannou známku, takže použití názvu CERT podléhá schválení této univerzity. Ta sice schvaluje možnost použít název bezplatně, ale ne úplně všem zájemcům. Z tohoto důvodu se v praxi pro tento typ týmů začal používat také jiný název **CSIRT**, který podobným omezením nepodléhá.

CSIRT znamená česky Tým pro reakce na počítačové bezpečnostní incidenty. Takové označení je podstatně specifičtější než CERT - indikuje zaměření týmu nikoliv na problémy běžné podpory uživatelů IT, ale na řešení problémů spojených s bezpečností.

K této transformaci došlo zejména v souvislosti s masifikací použití prostředků IT. Dnes jsou počítače běžnou každodenní součástí života většiny obyvatelstva a to jak v rovině pracovní, tak v rovině osobní a také změnám povahy hrozeb.

Rozvoj Internetu jako univerzální, celosvětové platformy pro sdílení informací sice na jedné straně otevřel firmám i jednotlivcům netušené možnosti, na straně druhé obdobné možnosti přinesl také lidem, kteří by takové možnosti chtěli zneužít pro vlastní prospěch - hackerům samotářům, nebo zločineckým syndikátům.

Dobře je situace viditelná na příkladu malware. Ještě v 90. letech bylo hlavní motivací tvůrců buď vlastní pobavení nebo demonstrace vlastní technologické nadřazenosti. S postupem času se ale situace změnila a proto dnes je hlavním cílem tvůrců vlastní prospěch ať už formou finanční (např. ransomware), nepřímé finanční (kompromitace počítačů za účelem realizace útoků **DoS**, rozesílání spamu za úplatu), zcizení osobních nebo firemních informací, které je možno zpeněžit na černém trhu.

Většina společností soudí, že specializovaný tým (CSIRT) by se tak měl zaměřovat především na oblast bezpečnosti - kde jsou vyžadovány specifické znalosti, popř. nástroje, které je výhodné soustředit na jedno místo - do týmu CSIRT. Běžná podpora uživatelů by pak měla být doménou oddělení IT.



Bezpečnostní incident vs mimořádná událost

Všimněte si používané terminologie. V souvislosti s bezpečností IT používáme pojem *bezpečnostní incident*, nikoliv pojem *mimořádná událost*, se kterým jste se dosud v průběhu studia setkávali. Rozdíl je poměrně důležitý - mimořádnou událost definuje zákon o integrovaném záchranném systému [7] jako *škodlivé působení sil a jevů vyvolaných činností člověka, přírodními vlivy, a také havárie, které ohrožují život, zdraví, majetek nebo životní prostředí a vyžadují provedení záchranných a likvidačních prací*.

Bezpečnostní incident je ze své povahy jiný - jedná se o jakékoliv z hlediska organizace významné narušení činnosti v její IT infrastruktuře. Může se tak jednat o kompromitaci počítačů malware, únik citlivých informací, masivní útok na infrastrukturu organizace apod. V žádném z výše uvedených scénářů však není vyžadováno provádění záchranných a likvidačních prací ze strany jednotlivých složek IZS.

Za určitých okolností může ale bezpečnostní incident přerůst do mimořádné události - např. ve formě dálkového útoku na technologii vedoucí k uvolnění nebezpečné látky do prostoru ...

Z hlediska fungování lze takové týmy organizovat různě. Např. výše uvedený CERT-CC je typickým představitelem podnikových CERT/CSIRT týmů. Tyto týmy působí v organizaci a řeší problémy dané organizace. To však není jediná úroveň, na které mohou tyto týmy pracovat.

Rada států realizuje tzv. národní a vládní CERT týmy. V případě České republiky plní funkci národního CERT týmu tým CSIRT.CZ [26], funkci vládního CERT týmu plní pak GovCERT.CZ [63].

Tým CSIRT.CZ byl zřízen v roce 2010 na základě memoranda mezi Ministerstvem vnitra ČR a sdružení CZ.NIC, jako provozovatelem týmu. V roce 2011 pak byla podepsána smlouva mezi CZ.NIC a Národním bezpečnostním úřadem, který se stal gestorem problematiky kybernetické bezpečnosti.

Funkce národního CERT týmu se soutěží. Poslední soutěž proběhla v roce 2015. Smlouva na provoz národního CERT týmu byla uzavřena na dobu neurčitou.

Pro úplnost v rámci národního týmu v současnosti působí 8 zaměstnanců. Podrobněji se k této problematice vrátíme za chvíli.

Poznámka:

Všimněte si používání pojmů CERT a CSIRT. Ve výše uvedeném textu se může zdát, že jsou zaměňovány a v praxi se tomu tak skutečně děje. Např. v obecné rovině se často hovoří o vrcholových CSIRT týmech, popř. národních a vládních CSIRT týmech.

Při probírání této problematiky v podmínkách ČR je však v souvislosti s provozem národního a vládního týmu potřeba respektovat dikci zákona o kybernetické bezpečnosti [6], který explicitně hovoří o národním CERT týmu a vládním CERT týmu.

Hlavní cíle CSIRT.CZ jsou následující [26]:

- Udržování zahraničních vztahů - se světovou komunitou CERT/CSIRT týmů a organizacemi, které tuto komunitu podporují.
- Spolupráce se subjekty v rámci ČR - ISP, poskytovateli obsahu, bankami, bezpečnostními složkami, akademickým sektorem, úřady státní správy a dalšími institucemi.
- Poskytování služeb v oblasti bezpečnosti
 - Řešení a koordinace řešení bezpečnostních incidentů
 - Osvětová a školicí činnost
 - Proaktivní služby v oblasti bezpečnosti

K těmto činnostem je tým vybaven poměrně dobře také díky svému provozovateli - sdružení CZ.NIC, které je správcem domény CZ. CZ.NIC tedy provozuje registr domén, registrovaných pod doménou CZ a zabezpečuje provoz domény nejvyšší úrovně .CZ. Má tedy k dispozici potřebné informace o sítích (doménách) provozovaných v ČR stejně jako mechanismy kontaktu jejich správců.

Právě zprostředkování kontaktů provozovatelů sítí je jedním z nejúčinnějších nástrojů jak řešit např. útoky. Např. pokud na síť organizace je veden útok z jiné sítě, je možné často síť, ze které byl útok veden identifikovat. Mezi provozovateli jednotlivých sítí ale obvykle není realizován přímý kontakt, který by umožnil problémy řešit operativně.

Národní CERT tak slouží jako ověřený/důvěryhodný kontaktní bod, který potřebné informace může zprostředkovat.

Vzhledem k tomu, že útoky nemusí přicházet pouze ze sítí provozovaných v ČR, je nutná také mezinárodní spolupráce. Důvěryhodnost je v tomto případě zaručena členstvím (akreditací) v mezinárodních uskupeních zaměřených právě na provoz CERT/CSIRT týmů.

Konkrétně se jedná o sdružení **Trusted Introducer (TI)** [36] a **FIRST** [32]. TI bylo zřízeno v roce 2000 evropskou komunitou CERT a CSIRT týmů [36]: *za účelem řešení společných potřeb a budování infrastruktury, která by poskytovala důležitou podporu všem bezpečnostním týmům, zaměřeným na řešení a řízení bezpečnostních incidentů.*

Jednotlivé týmy se do TI akreditují. Po akreditaci pak mají k dispozici některé služby především pak [36]:

- seznam týmů s potřebnými dodatečnými informacemi (např. kontaktními)
- zabezpečný mailing list
- možnost účastnit se porad, konferencí, školení organizovaných TI
- a některé další.

Forum of Incident Response and Security Teams (FIRST) sdružuje důvěryhodné CERT a CSIRT týmy spolupracující na řešení bezpečnostních incidentů a propagující programy jejich prevence. Z hlavních činností [32]:

- členové vyvíjejí a sdílejí technické informace, nástroje, metodologie, procesy a nejlepší postupy
- podporuje vývoj kvalitních produktů v oblasti bezpečnosti, politik a služeb
- vyvíjí a zveřejňuje nejlepší postupy v oblasti počítačové bezpečnosti
- podporuje vznik a rozvoj týmů reakce na incidenty a členství organizací ze všech koutů světa
- členové používají své kombinované znalosti, schopnosti a zkušenosti k podpoře bezpečnějšího globálního elektronického prostředí

Zvláštní postavení v této oblasti má také **European Union Agency for Network and Information Security (ENISA)** [29], česky Evropská agentura pro bezpečnost sítí a informací. Jedná se o agenturu Evropské unie zřízeného na základě Nařízení Evropského parlamentu a Rady (ES) č. 460/2004 o zřízení Evropské agentury pro bezpečnost sítí a informací. V současnosti je provoz této agentury regulován Nařízením Evropského parlamentu a Rady (ES) č. 526/2013 o Agentuře Evropské unie pro bezpečnost sítí a informací (ENISA) a o zrušení nařízení (ES) č. 460/2004.

Nařízení [30] definuje úkoly ENISy následovně:

1. podporuje rozvoj politiky a práva Unie (poradenství, analýzy a přípravné práce v oblasti bezpečnosti sítí a informací)
2. podporuje rozvoj schopností (podporuje dobrovolnou spolupráci mezi státy, zdokonalování prevence, odhalování a analýzy problémů a incidentů v oblasti bezpečnosti sítí a informací a schopnost na ně reagovat)
3. zajišťuje fungování skupiny pro reakci na počítačové hrozby (CERT) ... na úrovni Evropské unie.
4. zajišťuje fungování skupiny pro reakci na počítačové hrozby (CERT), organizuje cvičení
5. podporuje rozvíjení mechanismu včasného varování Unie
6. podporuje vývoj a výzkum a normalizaci evropských a mezinárodních norem pro řízení rizik a pro bezpečnost elektronických produktů, sítí a služeb
7. výměna know-how a poskytování poradenství s orgány, institucemi a jinými subjekty Unie, včetně těch, které se zabývají počítačovou kriminalitou a ochranou soukromí a osobních údajů, s cílem řešit otázky společného zájmu

Z hlediska fungování je zajímavé, že existence agentury byla původně časově omezená. Plnohodnotným orgánem Evropské komise se, nyní již bez časového omezení, ENISA stala až Nařízením Evropského parlamentu a rady 2019/881 [16].

Nařízení stanovilo ENISu jako tzv. *vrcholový CERT tým*, který na úrovni Evropské unie zastřešuje jednotlivé národní týmy. Nově má také specifikovány některé další úlohy (nad rámec úloh specifikovaných ve výše uvedeném seznamu), především směrem k:

- rozvíjení a aktualizaci strategií pro bezpečnost sítí a IS na úrovni EU a členských států, pokud o to požádají,
- podněcování spolupráce mezi veřejným a soukromým sektorem a v rámci soukromého sektoru, zejména za účelem podpory ochrany kritických infrastruktur,
- agregaci a analýze dobrovolně sdílených vnitrostátních zpráv od týmů CSIRT a interinstitucionálních týmů pro reakci na počítačové hrozby pro orgány, instituce a jiné subjekty EU.
- V případě rozsáhlých přeshraničních incidentů a krizí v oblasti kybernetické bezpečnosti by agentura ENISA měla přispět k reakci na úrovni.
- a řada dalších.

Nejdůležitějším novým úkolem, který není specifikován v seznamu výše je vytvoření a udržování *certifikačního rámce kyberbezpečnosti EU*. Tyto certifikace by měly osvědčit, že produkty a služby certifikované podle takových schémat splňují specifikované požadavky na kybernetickou bezpečnost.

V současnosti je totiž tato problematika řešena pomocí mezinárodní dohody SOG-IS, jejíž poslední verze je z roku 2010 [11]. Tato dohoda si kladla za cíl poskytnout vysokou úroveň důvěry ve spolehlivost a také konzistence hodnocení certifikací probíhajících podle standardů *Information Technology Security Evaluation Criteria (ITSEC)* a *Common Criteria (CC)*. Nevýhodou tohoto standardu bylo, že byl schválen pouze 14-ti členskými státy EU a Norska. Přitom např. Česká republika není signatářem této dohody (pro zajímavost ale Slovensko signatářem je).

Sjednotit přístup k certifikaci by měl certifikační schéma *European Cybersecurity Certification Scheme (EUCC)*. Toto schéma již nepracuje s rychle zastarávajícím standardem ITSEC, místo toho se exkluzivně věnuje CC (odtud také název). Schéma je již koncipováno jako standard, nikoliv jako mezinárodní dohoda, což by mělo zjednodušit certifikaci a možnost nasazování produktů takovou certifikací vyžadující v rámci jednotného trhu EU. Zvolená forma schématu pak umožňuje pokrýt celý trh najednou, což v minulosti nebylo možné.

V současnosti je schéma jako takové od února 2025 dostupné pro dodavatele. Je tedy již prakticky nasazováno.

GovCERT.CZ má za sebou poměrně krátkou, avšak pestrá minulost. Původně GovCERT.CZ provozoval *Národní bezpečnostní úřad (NBÚ)* v rámci *Národního centra kybernetické bezpečnosti (NCKB)* (od roku 2011). V roce 2017, ale došlo k transformaci NCKB do formy samostatného správního úřadu *Národní úřad pro kybernetickou a informační bezpečnost (NÚKIB)*.

Původní NCKB vznikl na základě usnesení vlády č. 781 o ustavení Národního bezpečnostního úřadu gestorem problematiky kybernetické bezpečnosti (říjen 2011). V té době však neexistoval právní rámec, který by centru dával pravomoci. Tyto byly doplněny až v roce 2014, kdy vešel v platnost zákon o kybernetické bezpečnosti, který už formálně definoval postavení vládního CERT týmu a svěřoval mu některé pravomoci.

K hlavní náplni práce NÚKIB patří [58]:

- provozovat Vládní CERT České republiky (GovCERT.CZ)
- spolupráce s ostatními národními a mezinárodními CERT a CSIRT týmy
- příprava bezpečnostních standardů pro informační systémy Kritické informační infrastruktury (KII) a Veřejných informačních systémů (VIS)
- osvěta a podpora vzdělávání v oblasti kybernetické bezpečnosti
- výzkum a vývoj v oblasti kybernetické bezpečnosti
- ochrana utajovaných informací v oblasti informačních komunikačních systémů
- kryptografická ochrana

Úkoly NÚKIB jsou tedy poměrně rozsáhlé, jaké úlohy však plní GovCERT.CZ? Odpověď lze nalézt na webových stránkách organizace [59]:

- koordinační činnost a pomoc při řešení incidentů
- zprostředkování kontaktů
- informační hub pro zveřejňování informací, analýz, článků apod. týkajících se aktuálních hrozeb
- sdílení dat o potenciálně infikovaných strojích v ČR
- nasazování honeypotů
- penetrační testování (v současnosti primárně zaměřeno na bezpečnost webových stránek)
- vzdělávací a výzkumná činnost
- forenzní laboratoř a SCADA laboratoř

Jak je vidno z přehledu výše jsou některé zajišťované činnosti totožné z činnostmi které provádí národní CERT tým - jedná se zejména o koordinační činnosti, zprostředkování kontaktů a zveřejňování bezpečnostně orientovaných informací především na webových stránkách instituce.

Vládní CERT tým, resp. NÚKIB jako takový disponuje ale mnohem většími zdroji, proto má kapacity v případě potřeby poskytnout také aktivní pomoc. Tato pomoc však je primárně určena především povinným osobám podle zákona o kybernetické bezpečnosti.

Zajímavé schopnosti představují informace zachycené z C&C (Command and Control) serverů různých botnetů, které obsahují seznamy kompromitovaných zařízení na síti, které do daného botnetu byly přiřazeny. Na základě takto získaných IP adres lze identifikovat sítě, ve kterých se taková zařízení nachází a kontaktovat jejich provozovatele.

Provozovatelé by se jinak k takovým informacím sami nedostali.

Dalším zdroj informací představují tzv. *honeypoty*. Jedná se o většinou servery, které jsou nakonfigurovány jako běžné servery (např. databázové, ale i jiné), které však neobsahují reálná data. Jedná se pouze o uměle vytvořený cíl pro případné útočníky. Vzhledem k tomu, že v tomto případě není důvod, aby se k serveru někdo reálně připojoval, budou všechna připojení spojena s různými útoky. Tak lze jednoduše identifikovat problémový síťový provoz a studovat mechaniku provedeného útoku.

Informace tohoto typu jsou cenné pro návrh a testování ochranných opatření.

Velmi důležitý je provoz forenzních laboratoří a laboratoře SCADA. Forenzní laboratoře umožňují analyzovat kompromitovaná zařízení v kontrolovaných podmínkách. Takto získaná zjištění jsou pak použitelná i v případném vyšetřování Policie ČR apod.

SCADA laboratoře jsou v podmínkách ČR nové. Jejich význam ale nesporný, umožňují totiž v kontrolovaných podmínkách testovat vlastnosti kontrolních systémů, což je nesmírně důležité např. pro ochranu kritické infrastruktury. Obdobné laboratoře budují dnes prakticky všechny rozvinuté státy.

Prostřednictvím ENISA je pak uvažováno o vytvoření obdobné laboratoře na úrovni EU jako celku.

5.2 Legislativa kybernetické bezpečnosti

Náš vhled do legislativy rozdělíme do dvou podkapitol. Samostatně se budeme věnovat Zákonu o kybernetické bezpečnosti, který tvoří jádro legislativy v této oblasti. V druhé podkapitole pak shrneme některé další pojmy, které se v souvislosti s kybernetickou bezpečností a kritickou infrastrukturou objevují v jiných zákonech.

5.2.1 Směrnice NIS2 a Zákon o kybernetické bezpečnosti

První pokus o regulaci kybernetické bezpečnosti v ČR byl realizován zákonem 181/2014 Sb. [6] o kybernetické bezpečnosti. Zákon byl v roce 2017 významně novelizován v souvislosti se vstupem do platnosti Směrnice EU 2016/1148 o opatřeních k zajištění vysoké společné úrovně bezpečnosti sítí a informačních systémů v Unii [13] (tzv. Směrnice NIS).

Primárním účelem směrnice NIS bylo sjednotit úroveň bezpečnosti sítí v EU, která byla dosud realizována různými státy na velmi odlišné úrovni. Většina opatření vyžadovaných směrnicí NIS byla již obsažena v původním zákonu o kybernetické bezpečnosti. Novela proto pouze rozšířila okruh povinných osob a také řešila některé další věci, které ale nebyly směrnicí vyžadovány, např. vznik správního úřadu NÚKIB.

V souvislosti s přijetím nové směrnice NIS2 [4] pak bylo potřeba přijmout novou právní úpravu této problematiky. ČR směrnicí implementovala zákonem 264/2025 Sb. o kybernetické bezpečnosti [87]. Ačkoliv se zákon jmenuje stejně jako zákon z roku 2014, jedná se o zcela novou právní úpravu této problematiky. Zákon je tedy velmi odlišný.

Z hlediska doby psaní těchto skript (IX 2025), se jedná o velmi čerstvou záležitost. Zákon vchází v platnost až 1. 11. 2025, tedy pravděpodobně až po vydání těchto skript.

Od 1.11. 2025 začíná také běžet pro povinné osoby také jednoleté přechodné období, v průběhu kterého by se měli jednak přihlásit k plnění tohoto zákona, jednak by měli implementovat opatření tak aby v dostatečné míře získali kontrolu nad svými systémy (z pohledu kybernetické bezpečnosti) a tím pádem splnili požadavky zákona.

Zákon se na organizaci může vztahovat ze tří důvodů:

1. organizace působí v jednom z regulovaných sektorů
2. poskytuje regulovanou službu
3. splňuje podmínku významnosti, zejména měřená velikostí organizace

Regulovaná odvětví odpovídají přibližně sektorům kritické infrastruktury, *regulovaným službám* pak odpovídají služby, které tyto sektory pro společnost¹ zajišťují.

Přesnou definici sektorů a služeb v budoucnu nalezneme ve vyhlášce o regulovaných odvětvích. Tato vyhláška ale v době psaní těchto skript (IX 2025) teprve dokončila sběr v rámci připomínkového řízení. Finální verze vyhlášky tak dosud není hotová.

Návrh obsahuje následující seznam sektorů:

- veřejná správa
- vodní hospodářství
- poštovní služby
- vesmírný průmysl
- zdravotnictví
- energetika (s podsektory: vodík, ropa a ropné produkty, teplárenství, elektřina, zemní plyn)
- potravinářský průmysl
- věda, výzkum a vzdělávání
- chemický průmysl
- výrobní průmysl
- digitální infrastruktura a služby
- drážní doprava
- odpadové hospodářství
- vodní doprava
- finanční trh
- obranný průmysl
- letecká doprava
- silniční doprava

Pokud porovnáme výše uvedený seznam regulovaných sektorů s odvětvími kritické infrastruktury, tak jak je najdeme např. na příloze zákona 266/2025 Sb. o kritických subjektech [88].

Zákon odvětví KI vyjmenovává následovně:

- energetika (pododvětví elektřina, dálkové vytápění a chlazení, ropa, zemní plyn, vodík)
- doprava (pododvětví letecká, železniční, vodní, silniční doprava a veřejná přeprava)

¹společnost se v tomto kontextu rozumí komunita občanů, nikoliv tedy organizace nebo podnik

- bankovníctví
- infrastruktura finanční trhů
- zdravotnictví
- pitná voda
- odpadní voda
- digitální infrastruktura (pododvětví služby vytvářející důvěru a elektronická identifikace, veřejné komunikační sítě a veřejně dostupné služby elektronických komunikací, ostatní digitální služby a infrastruktura)
- veřejná správa
- vesmír
- výroba, zpracování a distribuce potravin (pododvětví výroba potravin, skladování a distribuce potravin)
- bezpečnost (pododvětví požární ochrana a ochrana obyvatelstva, vnitřní pořádek, státní hmotné rezervy, Hydrometeorologická výstražná služba)

Pro rozdíly existují důvody. Např. bankovníctví je odvětvím KI ale ne regulovaným sektorem. Důvodem v tomto případě je, že bankovníctví z pohledu kybernetické bezpečnosti není regulované směrnicí NIS2, ale nařízením DORA (2022/2554).

Nemáme prostor, abychom rozebírali rozdíly, proto se spokojíme tím, že zde existuje významná souvislost, ale zároveň tento přesah není z řady důvodů úplný.

Nový zákon o kybernetické bezpečnosti v definici regulovaných oblastí výrazně širší než zákon původní. Původní zákon se dotkl přibližně 600 organizací. Pro nový zákon se v současnosti odhaduje, že se bude týkat 6 - 10 000 subjektů, tedy řádově více.

Přesný počet není subjektů v současnosti není znám a bude hodně záviset na výkladu ustanovení, zejména o kvalifikačních kritériích. Další komplikací je také to, že kromě regulovaných subjektů (subjektů v regulovaných odvětvích, poskytujících regulované služby) se zprostředkovaně do určité míry vztahuje také na jejich dodavatele. Dosud ale není zcela jasné, jak bude toto ustanovení vykládáno v praxi.

Z hlediska povinností zavádí zákon dvojkolejný systém povinností:

- *vyšší režim povinností* - určený pro velké podniky nebo organizace s klíčovým dopadem na společnost
- *nižší režim povinností* se vztahuje na všechny ostatní povinné osoby

Organizace spadající pod zákon v obou případech zavádějí povinně bezpečnostní opatření, liší se ale rozsah s jakým tak musí činit.

Také hlavní povinnosti jsou pro obě skupiny stejné:

- ohlášení regulované služby
- hlášení kontaktních údajů
- stanovení rozsahu řízení kybernetické bezpečnosti
- zavádění bezpečnostních opatření
- hlášení kybernetických incidentů
- informování zákazníků o incidentech a hrozbách
- realizace protiopatření daných NÚKIB
- kontrola bezpečnosti dodavatelského řetězce
- zajištění dostupnosti strategicky významné služby

V nižším režimu povinností jsou nižší požadavky na odpovědné osoby, které zajišťují kybernetickou bezpečnost v organizaci a také významně nižší požadavky na bezpečnostní dokumentaci a procesy v organizaci.

Organizace v režimu vyšších povinností musí zavést v podstatě plnohodnotný systém **systém řízení bezpečnosti informací (SRBI)/Information Security Management System (ISMS)** kompatibilní s požadavky základních norem ISO 27000, především ISO 27001, 27002 a 27005.

Zákon ale nevyžaduje explicitně, že SRBI musí být realizován podle těchto norem, nebo že by dokonce musel být certifikovaný. Lze ale říci, že pokud se organizace rozhodne zavést systém SRBI podle těchto norem a dotáhne jej až k certifikaci, bude se jí později jednodušeji prokazovat, že tento systém je v souladu s požadavky zákona.

Všimněte si také v povinnostech požadavků na řízení bezpečnosti dodavatelských řetězců. Tento požadavek reaguje na v IT poměrně běžné problémy, kdy zranitelnost systému nevyplývá ze špatné konfigurace, pravidel provozu nebo obsluhy, ale je vnesen do systému některou z použitých komponent.

Dobrým příkladem z poslední doby (září 2025) je útok na dodavatelský řetězec NPM [38]. NPM je jedna z nejpobulárnějších technologií, která umožňuje řídit závislosti v knihovnách používaných různými informačními systémy. Útočník kompromitoval jednu z pobulárních knihoven, která byla využívána dalšími knihovnami, které byly využívány různými informačními systémy. Pozměněný kód přeměroval platby v kryptoměnách do peněženek útočníka.

Útok tohoto typu tedy necílí na informační systém jako takový, protože tento je obvykle dobře chráněný, ale některou z jeho komponent, které tento systém pouze přejímá od svých dodavatelů. Ve velkých informačních systémech není neobvyklé, že takový dodavatelský řetězec je velmi dlouhý. Informační systém tak může používat desítky nebo dokonce stovky externích knihoven, nad kterými nemá přímou kontrolu.

Opatření k získání kontroly na bezpečnosti dodavatelského řetězce tedy vede k tomu, že poskytovatel regulované služby tlačí na své dodavatele, aby měli zaveden systém řízení bezpečnosti, který zajistí, že k takovým problémům pokud možno nebude docházet.

Dodavatelé v tomto případě tedy nejsou přímo regulováni zákonem, ale přesto jeho efekty na ně dopadají.

V souvislosti s dodavatelskými řetězci je v zákoně také ještě jedna možnost a to zakázat určitého dodavatele. Zakaz by se měl týkat dodavatelů technologií v oblasti telekomunikací, datových center apod. Důvodem je v podstatě politické riziko.

V současnosti v této souvislosti hovoříme zejména o dodavatelích jako je společnost Huawei nebo ZTE z Číny. Zdůvodnění v tomto případě je, že právní řád Číny obsahuje zákon, který nutí Čínské společnosti na vyžádání Čínských orgánů poskytnout součinnost. Obava je pak taková, že by tito dodavatelé mohli být donuceni k vývoji zadních vrátek do systému, nebo donuceni dokonce k odstavení technologie jako takové.

V systémech kritické infrastruktury pak použití takových technologií může být vysoce rizikové. Za provoz KI za normálních okolností zodpovídá jejich provozovatel. Ten také obvykle je zodpovědný za management rizik, které se této infrastruktury týkají. Pro běžná rizika takový systém funguje dobře. Problém je, že pro riziko úmyslného narušení bezpečnosti dodavatelem provozovatel (subjekt) KI nemá dost informací, protože není příjemcem zpravodajských informací.

V současnosti v tomto ohledu postupujeme stále ještě podle původního zákona, který umožňuje NÚKIB vydat varování - v tomto kontextu před použitím určitých technologií. Těchto varování byla vydána celá řada - před společností Huawei a ZTE pro dodávky mobilních sítí 5G [60], použití sociální sítě TikTok [61], z poslední doby pak varování před použitím AI DeepSeek [62], všechny se silou Čínskou stopou.

Zákaz dodávek bude fungovat jinak. Bude se jednat o skutečný zákaz používání, nikoliv pouze varování před používáním. Takový zákaz dosud nebyl nikdy vydán. Podle nového zákona o kybernetické bezpečnosti může zákaz vydat pouze Vláda ČR.

Je zajímavé, že tato možnost nevyplývá z požadavků směrnice NIS2, ale byla doplněna do zákona navíc. ČR byla jedním z mála států EU, který dosud možnost takového zákazu neměl.

Velmi významnou změnou prošel také *stav kybernetického nebezpečí (SKN)*, který zákon definuje jako stav, který lze vyhlásit v případě, kdy je značně ohrožena, nebo narušena bezpečnost informací v kybernetickém prostoru, což může mít za následek negativní dopady na poskytování regulovaných služeb nebo může dojít k ohrožení bezpečnosti ČR, vnitřního pořádku, života a zdraví, majetkových hodnot nebo životního prostředí.

Původní definice v předchozím zákoně byla koncipována obecněji směrem k chráněným zájmům ČR, které jsou definovány v zákonu o utajovaných skutečnostech. Tento odkaz byl ale poměrně často chybně interpretován, že stav kybernetického nebezpečí je spojován právě s ochranou utajovaných skutečností. Nová definice je tak z hlediska interpretace jednodušší.

Upravena byla také délka tohoto stavu. Původně mohl být tento stav vyhlášen maximálně na 7 dní a pak prodlužován až na maximální délku 30 dní. Nová právní úprava umožňuje vyhlásit tento stav na 30 dní s možností prodloužení o dalších 30 dní. V případě, že ani tato prodloužená doba není dostatečná, žádá NÚKIB vládu o vyhlášení nouzového stavu, v rámci kterého se stav kybernetického nebezpečí prodlouží.

Výše uvedená změna reaguje na čerstvé zkušenosti z konfliktu na Ukrajině, kde vojenské operace jsou doprovázeny tvrdým bojem v kybernetickém prostoru. Tato situace trvá v podstatě nezměněné intenzitě od roku 2022 až do současnosti.

Původní právní úprava nebyla na takové situace připravena. Stav kybernetického nebezpečí byl kalibrován spíše na hacktivistické útoky, nebo náhodné útoky malware než dlouhodobý nebo dokonce trvalý útok na kritické systémy státu.

Nová pravidla sice také úplně neřeší situace, jakým čelí denně Ukrajina, ale delší časový interval přesto dává lepší možnosti reakce a umožňuje jasně eskalovat stav kybernetického nebezpečí pomocí vyhlášení nouzového stavu.

Změnila se práva, která má NÚKIB po vyhlášení SKN. Původní právní úprava rozšiřovala okruh osob, kterým může NÚKIB uložit vykonat opatření, ale všechny tyto osoby byly vyjmenovány v zákoně. Toto se v novém zákonu změnilo. Vyhlášením SKN získává NÚKIB možnost uložit opatření v podstatě komukoliv, tedy i takovým organizacím, kterých se zákon jinak netýká.

Další klíčovou povinností je *hlášení kybernetických bezpečnostních incidentů*. Z hlediska systému SŘBI je hlášení vlastně až jedním z posledních kroků, které organizace v případě *kybernetického bezpečnostního incidentu* dělá. Pro správné pochopení celého procesu bychom měli začít spíše na jeho začátku a to *bezpečnostní událostí* a schopností organizace ji detekovat, popř. na ni rovnou reagovat.

Bezpečnostní událostí rozumíme jakoukoliv událost, která má potenciál narušit některý z aspektů bezpečnosti chráněných informací. Tyto aspekty se obvykle skrývají pod zkratkou **Confidentiality, Integrity, Availability (CIA)**, tedy důvěrnost, integrita a dostupnost informací.

Bezpečnostní události se v systémech vyskytují poměrně často. Systém by tak měl být dobře připraven se vypořádat s takovými událostmi, aniž by došlo k reálnému dopadu na bezpečnost.

Typickými představiteli takových událostí je např. skenování portů počítače, neúspěšné pokusy o přihlášení, pokus o načtení neexistující URL na webovém serveru. Ve všech těchto případech se může jednat o průvodní znaky, nebo počáteční fáze, nějakého útoku.

Tyto události proto potřebujeme detekovat, obvykle záznam o detekci ukládáme do logu a logy pravidelně vyhodnocujeme tak, abychom zajistili, že všechny relevantní bezpečnostní události jsou skutečně pod kontrolou a systém na ně reaguje tak, jak má.

V některých případech, ale systém nezareaguje správně. Bezpečnostní událost tak postupně eskaluje až do stavu, kdy bude mít reálné dopady na bezpečnost. V tomto okamžiku se z bezpečnostní události stává *bezpečnostní incident*. Organizace v takovém případě musí:

- podniknout kroky vedoucí k přerušení incidentu a obnově narušeného systému
- hlásit takový incident příslušnému CERT/CSIRT týmu

Pokud se bavíme o bezpečnostních incidentech v kontextu zákona, pak uvažujeme takové, kde už dochází k narušení bezpečnosti regulované služby. Předpokládáme tedy dopad, který se netýká pouze postižené organizace ale je pociťován i mimo ni např. odběrateli služeb. Zákon pracuje s pojmem *významný dopad* a definuje jej jako takový dopad, který způsobil nebo mohl způsobit závažné narušení provozu regulované služby, finanční ztráty nebo může jiným osobám způsobit značnou újmu.

Logicky z toho vyplývá, že třeba nebudeme hlásit rutinní kompromitaci účtu, kterou jsme vyřešili resetem hesla.

Většina bezpečnostních incidentů je relativně málo významných. V praxi se setkáváme ale také s velmi významnými bezpečnostními incidenty, kterou mohou narušit schopnost poskytovat služby třeba na týdny. Hlášení příslušnému CSIRT týmu v takovém může být jedním z prvních kroků jak získat externí pomoc. CSIRT tým za určitých okolností může poskytnout podporu pro zvládnutí incidentu a obnově provozu a to poskytnutí rad až po nasazení pracovníků na místě.

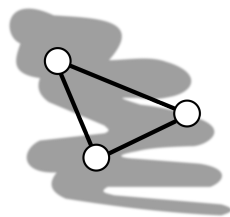
Hodnocení významu bude tedy vždy na organizaci, která incidentu čelí. Způsob hodnocení bude určen prováděcí vyhláškou zákona, která ale v době psaní těchto skript (IX 2025) nebyla k dispozici.

Zákon o kybernetické bezpečnosti se zaměřuje na základní regulaci prostředí v organizacích. Neřeší tedy ze své podstaty implementační detaily toho, jak tato regulace bude aplikována v realu. V souvislosti se zákonem tak existuje řada otázek, které bude postupně potřeba řešit řadou prováděcích vyhlášek.

Jelikož je zákon 264/2025 Sb o kybernetické bezpečnosti nový, tak tyto vyhlášky zatím k dispozici nejsou. Jsme tedy v přechodném období, kdy částečně postupujeme dle starého zákona, který je plně vybaven prováděcími vyhláškami. a můžeme se dohadovat o tom, co bude, popř. nebude použito v nové právní úpravě.

Bezpečným odhadem je, že v SŘBI budou požadovány některé bezpečnostní role, které organizace musí mít obsazený. Tento odhad je založen na tom, že požadavky zákona vycházejí z norem a ty

takové role požadují. Zároveň jsou tyto požadavky obsaženy již v předchozí legislativě, především pak Vyhlášky č. 82/2018 Sb. o kybernetické bezpečnosti.



ISO 27000

Kodex norem ISO 27000 obsahuje v současnosti několik desítek norem, základ ale tvoří tři řešící:

- ISO 27001 [2] - systém ISMS
- ISO 27002 [3] - nejlepších praktik z oblasti bezpečnosti informací
- ISO 27005 [8] - rizika

V tomto předmětu není prostor se touto problematikou zabývat ve větší míře, proto v případném studiu můžete pokračovat buďto samostatně nebo v rámci předmětu *Bezpečnosti informačních systémů*.

Klíčovými bezpečnostními rolami jsou:

- *manažer kybernetické bezpečnosti*, který má odpovědnost za systém SRBI, koordinuje řízení rizik, incidentů, aktiv a bezpečnostní dokumentaci
- *architekt kybernetické bezpečnosti* navrhuje bezpečnostní architekturu služeb, definuje bezpečnostní požadavky při vývoji a změnách systémů, provádí testování a revize bezpečnostních opatření
- *auditor kybernetické bezpečnosti* provádí nezávislé audity, poskytuje zpětnou vazbu vedení
- *garant aktiva* odpovídá za bezpečnost konkrétního aktiva, stanovuje požadavky na důvěrnost, dostupnost a integritu

Základní organizační strukturou je pak *výbor kybernetické bezpečnosti*. Jedná se o strategický orgán, který řídí kybernetickou bezpečnost na té nejvyšší úrovni. Stanovuje cíle, priority a opatření, která jsou následně implementována jednotlivými bezpečnostními rolami a následně kontrolu jejich plnění.

Z hlediska opatření pracuje zákon se dvěma základními typy opatření, které implementují organizace pro zajištění své bezpečnosti:

- *organizační* se týkají řízení dokumentace, školení, procesů apod.
- *technická* se týkají konkrétních technologií a jejich konfigurace.

Tyto dva základní typy opatření jsou doplňovány *speciálními protiopatřeními*:

- výstraha
- varování
- reaktivní protiopatření

Výstrahou úřad informuje veřejnost o kybernetickém bezpečnostním incidentu nebo porušování povinností vyplývajících ze zákona. Alternativně může úřad uložit provozovateli regulované služby aby výstrahu zveřejnila sama.

Varování vydává úřad v případě, že se dozví o závažné hrozbě nebo zranitelnosti v oblasti kybernetické bezpečnosti.

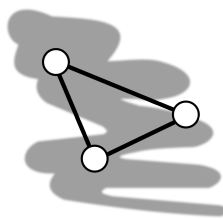
Konečně *reaktivní protiopatření* ukládá úřad provozovateli regulované služby. Opatření vždy řeší hrozící nebo probíhající kybernetický bezpečnostní incident. Opatření může být zaměřeno na zabezpečení aktiva proti takovému incidentu. V tomto případě má opatření za cíl preventivně zabránit vzniku takového incidentu. Nebo se může jednat o opatření, které vychází z analýzy již ukončeného bezpečnostního incidentu. V takovém případě je cílem opatření zabránit opakování incidentu.

Za normálních okolností může opatření ukládat úřad pouze organizacím poskytujících regulované služby, pokud je ale vyhlášen SKN, pak toto omezení padá a úřad může opatření ukládat v podstatě komukoliv.

5.2.2 Ostatní legislativní souvislosti

V právním řádu ČR a EU existuje celá řada souvislostí, které stojí za zmínku. Výše jsme už poměrně podrobně zmiňovali souvislost s ochranou kritické infrastruktury. Pro úplnost uvádíme, že tato oblast je regulovaná směrnicí CER o odolnosti kritických subjektů (2022/2557), kterou do právního řádu transponoval zákon 266/2025 Sb. o kritické infrastruktuře.

Tak jako jsou směrnice NIS2 a CER provázané, jsou poměrně výrazně provázané také zákony o kritické infrastruktuře a o kybernetické bezpečnosti. V obou případech se jedná o zcela nové právní úpravy, řešené ve větší míře detailů než v minulosti.



Kritická infrastruktura

Pokud jste studenty studijního programu Bezpečnostní plánování, budete se problematikou kritické infrastruktury podrobněji zabývat v předmětu *Ochrana kritické infrastruktury*.

Až budete studovat předmět, vzpomeňte si, že ochrana kritické infrastruktury má svůj IT rozměr a něco málo jste se o něm dozvěděli v tomto předmětu :-).

S problematikou KI do jisté míry souvisí také *služby elektronických komunikací*. Tyto definuje zákon o elektronických komunikacích [9] (§2 písm. n) jako služby obvykle poskytovaná za úplatu, která spočívá zcela nebo převážně v přenosu signálů po sítích elektronických komunikací, včetně telekomunikačních služeb a přenosových služeb v sítích používaných pro rozhlasové a televizní vysílání a v sítích kabelové televize, s výjimkou služeb, které nabízejí obsah prostřednictvím sítí a služeb elektronických komunikací nebo vykonávají redakční dohled nad obsahem přenášeným sítěmi a poskytovaným službami elektronických komunikací; nezahrnuje služby informační společnosti, které nespočívají zcela nebo převážně v přenosu signálů po sítích elektronických komunikací.

Taková definice pokrývá celou škálu provozovatelů různých sítí - od televizních a rozhlasových sítí, až po sítě např. operátorů mobilních sítí.

Pojmem *významná síť* se rozumí síť zajišťující přímé zahraniční propojení do veřejných komunikačních sítí nebo zajišťující přímé připojení ke kritické informační infrastruktuře. Do této skupiny tak spadají tzv. páteřní sítě, ale také samostatné sítě zajišťující řízení kritické infrastruktury. Takové sítě jsou typické např. pro odvětví energetiky budující rozsáhlé sítě produktovodů (zemní plyn, ropa) apod.

Vyhláška o významných informačních systémech a jejich určujících kritériích

Vyhláška 317/2014 Sb. řeší kritéria, podle kterých se určují kritéria významné informační systémy. Takové systémy musí jejich provozovatel ze zákona lépe chránit. Určující kritéria se člení na *dopadová* a *oblastní* kritéria.

Svou dík, Vám možná výše uvedený odstavec něco připomene - třeba průřezová a odvětvová kritéria pro určování prvků kritické infrastruktury dle nařízení vlády 432/2010 [12].

Dopadová kritéria hodnotí dopady úplné nebo částečné nefunkčnosti systému způsobené narušením bezpečnosti. Hodnotí se přitom dopady na provozovatele, případně poskytování služby nebo informací orgány veřejné moci nebo další dopady mimo samotný systém. Spektrum takovým dopadů je poměrně různorodé - může se jednat o narušení prvku kritické infrastruktury, počítat lze také se ztrátami na životech, zdraví, majetku nebo obecně veřejných zájmů.

Oblastní určující kritéria se rozlišují podle provozovatele systému a to orgány veřejné moci a pak také orgány veřejné moci - kraje v přenesené působnosti. Jedná se především o schopnost vést správné řízení, vykovávat státní dozor, hospodařit, spravovat osobní údaje a další.

Úplný přehled oblastních kritérií je specifikován v příloze 2 vyhlášky.

V určování prvků KI je potřeba dodat, že nová právní úprava se zaměřuje na subjekty poskytující základní služby, což je výrazný posun proti původnímu zaměření právní úpravy, která se zaměřovala na prvky a jejich ochranu. Lze proto předpokládat, že bude přijato nové nařízení vlády, které tento nový přístup bude aplikovat.

Zatím není jasné, zda nařízení k určování prvků KI zůstane i nadále v platnosti nebo jej nové nařízení úplně nahradí. Argumentovat by šlo oběma směry.

Dalším pojmem jsou *významné informační systémy*. S těmi pracoval původní zákon o kybernetické bezpečnosti. Klasifikace a jmenný seznam takových systémů byl obsažen ve vyhlášce č. 317/2014 Sb. Nový zákon, ale pracuje jinak, což povede k odlišné klasifikaci informačních systémů.

Prováděcí vyhláška zákona, ale dosud nebyla přijata (IX 2025), proto nelze s jistotou říci, jak bude tato problematika ve finále přesně upravena.

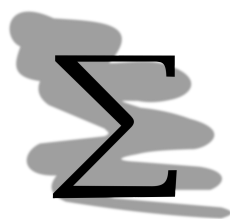


Významné informační systémy vs informační systémy veřejné správy

Prosím neplést tyto pojmy. **Informační Systém Veřejné Správy (ISVS)** jsou souborem informačních systémů, které slouží pro výkon veřejné správy. Jejich pořizování a provoz se řídí zákonem 365/2000 Sb. o ISVS [5] a jeho prováděcími vyhláškami. V ČR je v současnosti provozováno dle Informačního systému o informačních systémech veřejné správy 6 633 [56] takových systémů, přičemž nejvíce je jich provozováno městy a obcemi v samostatné působnosti (1 818), přenesené působnosti (1 384) a ostatní (520).

Problematicke **ISVS** byl věnován prostor v předmětu *Bezpečnostní informatika*^a.

^aod roku 2018, do roku 2018 byla problematika **ISVS** probírána na FBI v rámci předmětu *Bezpečnostní informatika 2*.



Shrnutí kapitoly

Základní právní úpravou problematiky kybernetické bezpečnosti představuje zákon o kybernetické bezpečnosti. Tento zákon definuje právní postavení, pravomoci a povinnosti národního a vládního CERT týmu. Národní tým CERT.CZ je primárně určen pro zprostředkování kontaktů mezi provozovateli sítí pro řešení zaznamenaných narušení kybernetické bezpečnosti.

Vládní tým GovCERT.CZ provozuje **NÚKIB**. Ten je určen primárně provozovatelům významných sítí a systémů a také systémů **Kritická informační infrastruktura (KII)**.

Provozovatelům výše uvedených systémů vznikají dle zákona některé povinnosti, především pak vést bezpečnostní dokumentaci, monitorovat kybernetické bezpečnostní události a hlásit bezpečnostní incidenty.

V případě, že hrozí vážné narušení provozu významných sítí a systémů, popř. systémů KII, kde rozsah narušení by mohl ohrozit základní zájmy ČR, může ředitel **NÚKIB** vyhlásit stav kybernetického nebezpečí. Behem trvání tohoto stavu se Úřadu rozšiřuje okruh osob, kterým může ukládat úřad opatření. Za normálních okolností přitom úřad může ukládat taková opatření pouze provozovatelům významných systémů a KII.



Kontrolní otázky

1. Co je reaktivní opatření?
2. Jaký je rozdíl mezi národním a vládním CERT týmem?
3. Co je poskytovatel regulované služby, uveďte příklad.
4. Jaký je rozdíl mezi událostí a incidentem (kybernetická bezpečnost)?
5. Jak se liší režim vyšších a nižších povinností dle zákona o kybernetické bezpečnosti?



Správné odpovědi

1. Opatření řešící hrozící/probíhající bezpečnostní incident.
2. Národní je provozovaný soukromou org. a je určen primárně ke koordinaci mezi provozovateli běžných sítí, vládní CERT proti tomu primárně řeší kybernetickou bezp. ve spolupráci s provozovateli významných IS a KII a je provozován správním úřadem (**NÚKIB**).
3. Rozumí se jím provozovatelé klíčových služeb pro fungování společnosti. Patří zde kritická infrastruktura a další systémy nutné pro plynulé fungování společnosti.
4. Událost je obecnější, incident už představuje reálné narušení kybernetické bezpečnosti.
5. V režimu vyšších povinností jsou vyšší nároky na organizaci informační bezpečnosti, dokumentaci a personální zabezpečení této problematiky.

Kapitola 6

Data mining



Náhled kapitoly

V této kapitole se přesuneme od problematiky systémů řízení k problematice data miningu. Tato kapitola slouží jako úvod do problematiky. Proto si v ní řekneme co to vůbec data mining je, ale také jaké problémy a pomocí jakých metod řeší. Další kapitoly se pak zaměří na jednotlivé nástroje a metody, které pro dolování informací můžeme použít. Tato kapitola tak poslouží jako teoretické východisko pro další výklad jednotlivých metod.

Po přečtení této kapitoly budete vědět

- co je to data mining
- jaké metody lze pro dolování informací použít
- a co od nich můžeme očekávat

znát

- obecný postup řešení data miningových projektů



Čas pro studium

Pro prostudování této kapitoly budete potřebovat přibližně dvě hodiny.

Data mining je pojem, který souhrnně označuje postupy schopné identifikovat v datech zajímavé souvislosti nebo skryté informace. Pro data mining je typické zpracování velkých objemů dat, které jsou pouze obtížně zpracovatelné pomocí běžných tabulkových procesorů, jako je např. MS Excel. Bavíme se tak o rozsahu dat od desítek tisíců řádek výše, s desítkami, stovkami nebo více sloupci s tím, že horní limit pro data mining není určen.

Pro dolování dat je také typické, že data samotná se nachází ve fragmentované podobě, umístěna ve velkém množství datových zdrojů. Jedním ze základních problémů, které data mining řeší je tak integrace dat z těchto různorodých zdrojů tak, aby proces dolování mohl vůbec započít.

V české literatuře se někdy používá doslovný překlad dolování dat. Objevují se také jiné názvy jako dobývání dat, v angličtině se pak někdy používá také **Knowledge Discovery in Databases (KDD)**. *Data mining* je ale jednoznačně nejpoužívanější název a to i v češtině.

Původně se data mining používal především v pojišťovnách, statistických úřadech, bankách, investičních fondech a výzkumu. Motivací byla na jedné straně existence ohromných objemů dat a na straně druhé relativní dostupnost finančních zdrojů nutných pro investici do nástrojů, které by data mining realizovaly.

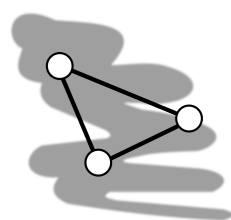
Představiteli takových nástrojů jsou např. SAS [72] od stejnojmenné společnosti a SPSS [41] od společnosti IBM. Obě tyto společnosti stály u zrodu data miningu a dokonce vyvinuly vlastní doporučené postupy realizace data miningu.

S postupem doby a rozvojem výpočetní techniky, se ale začaly objevovat také alternativní, cenově dostupné nástroje, které data mining zpřístupnili také malým a středním firmám nebo dokonce jednotlivcům. V současnosti nejpoužívanější analytický nástroj celosvětově je R [15], který je open source. To znamená, že je dostupný bezplatně a to i ke komerčnímu použití.

Otevřená podstata R přispěla k rozšíření v akademickém prostředí, které pak přispívá v vývoji jak prostředí samotného, tak rozšiřujících knihoven poskytující nové funkce a postupy, které pokrývají prakticky všechny oblasti matematiky, statistiky, strojového učení apod. Dostupnost širokého spektra funkcí pak vedla k rychlé adopci R pro realizaci cenově dostupných analýz dat v organizacích různého zaměření i velikosti.

Na rozdíl od nástrojů SAS nebo SPSS nedisponuje R jednotným grafickým rozhraním, které by umožňovalo parametry datových analýz „naklikat“. Proto je s použitím R spojena nutnost zvládnout vestavěný programovací jazyk a také způsobu, jakým jsou implementovány jednotlivé toolkity.

Pro řešení problémů dataminingu v R je pak nutné zvládnutí zvolené knihovny implementující analytické metody, které jsme vybrali pro řešení problému.



R zapojení do výuky

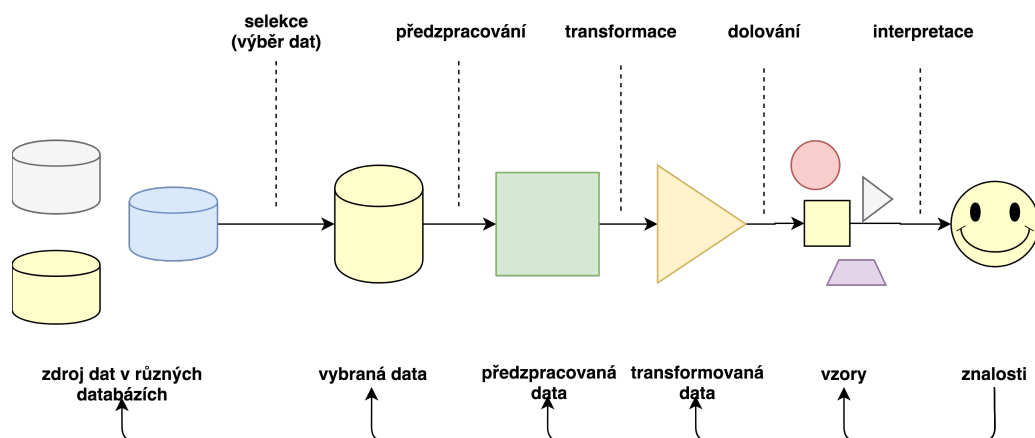
Dostupnost knihoven různého druhu, včetně těch, které jsou zaměřeny na řešení různých problémů včetně implementace různých metod použitelných pro data mining, byla jedním z faktorů, které přispěly v volbě tohoto nástroje pro výuku. V následující kapitole se seznámíme se základy použití R. V dalších kapitolách se zaměříme na jednotlivé metody (a knihovny, které je implementují), praktické ukázky pak budou demonstrovány právě v R.

6.1 Postup data miningu

Bez ohledu na zvolený analytický nástroj je základní postup data miningu stejný. Můžeme se na něj dívat technicky nebo manažersky. Při *technickém* pohledu vnímáme data mining jako na posloupnost na sebe navazujících činností realizovaných v analytickém nástroji. Oproti tomu při *manažerském* pohledu je pro nás primární spíše problém, který se pomocí data miningu snažíme řešit. Data mining je tak vnímán jako projekt, který řešíme sestavením řešitelského týmu, přidělením zdrojů apod.

Tyto přístupy si vzájemně neodporují - prostě se na problematiku data miningu dívají z odlišného úhlu pohledu. Graficky jsou přístupy znázorněny na obr. 6.1 a 6.2.

Oba přístupy jsou vzájemně kompatibilní, lze dokonce říci, že se vzájemně doplňují.



Obrázek 6.1: Technický pohled na postup data miningu (adaptováno z [20])

Data použitelná v data miningu se obvykle nacházejí v různých informačních systémech popř. jiných zdrojích, které organizace používá pro vedení svých agend. Data mohou být také získána z

měřicí techniky, technologie, ale také externích zdrojů (od externích dodavatelů). Před jejich použitím pro účely data miningu je ale potřeba rozhodnout, která data jsou pro řešení zvoleného problému vůbec relevantní.

Vybíráme tak relevantní data (*selekce*) pro další zpracování. Relevantností přitom rozumíme předpokládanou relevantnost. Vybíráme tedy taková data, o kterých se domníváme, že mají potenciál (umožní nám) vyřešit zadaný problém. Nemáme ale jistotu, že naše data skutečně tuto schopnost mají. Jistotu získáme až později, resp. vlastně na konci, celého procesu.

Na začátku procesu dolování data zůstávají ještě ve svých původních umístěních. Jejich fyzický výběr a příprava pro data mining se děje až v kroku *předzpracování dat*, v rámci kterého jsou data extrahována ze svého původního umístění a integrována v úložišti zvoleného data miningového nástroje.

Proces předzpracování je často časově velmi náročný, často náročnější než samotná aplikace data miningových metod pro získání znalostí. Operace selekce a předzpracování je přitom v rámci data miningového projektu často nutné opakovat, protože zejména na začátku projektu není vždy zcela jasné, jaká data jsou pro dosažení cíle vlastně potřeba.

Konečně z data miningu vyplývá, že hledáme znalosti které jsou **skryté** v datech. Jak vyplývá ze zpětnovazebních šipek na obr. 6.1 je možné se v rámci data miningu vrátit zpět a opakovat jeden nebo více kroků např. se změnou vstupních parametrů apod.

V kroku *transformace* připravujeme data na jejich zpracování pomocí zvolených data miningových metod. Tento krok je nutné provést, jelikož různé metody očekávají odlišné vstupy. V rámci transformace řešíme často následující problémy, ale nejen je:

- chybějící data,
- standardizace dat do jednotného (požadovaného formátu),
- diskretizace dat (převod spojitých dat do intervalů),
- apod.

Z výše uvedeného vyplývá, že pro správné provedení transformace dat je již nutné mít vybrané metody, které chceme pro samotné dolování dat použít. Po provedení transformace již bez problémů můžeme na připravená data tyto metody použít a získat *vzory*.

Vzory jsou tedy přímým výstupem zvolených metod. Tedy nestačí aplikovat metodu, napsat výsledek a prohlásit náš úkol za vyřešený - výsledky je nutné interpretovat. Jinými slovy je potřeba zjistit, co přesně takové výsledky znamenají. Jsou užitečné? Jaká je chyba v řešení, ke kterému jsme došli? Je možno kvalitu řešení zlepšit, např. doplněním dalších vstupních dat nebo volbou jiné metody, popř. změnou parametrů metody stávající? Podobných otázek pak lze vymyslet celou řadu.

Proces, kterým kontrolujeme nakolik námi vyvinuté modely jsou přesné nazýváme *validace*. V rámci validace porovnáváme výsledky modelu s očekávanými a vypočítáváme chybu. To nám umožní rozhodnout, zda byl celý proces úspěšný, nebo jej bude potřeba opakovat, popř. nějakým způsobem modifikovat, např. úpravou dat, použitím jiných dat, nebo jiných analytických metod.

Odpovědi na tyto otázky nám pak spolu s výsledky metod tvoří znalosti, které jsou aplikovatelné pro řešení problémů.

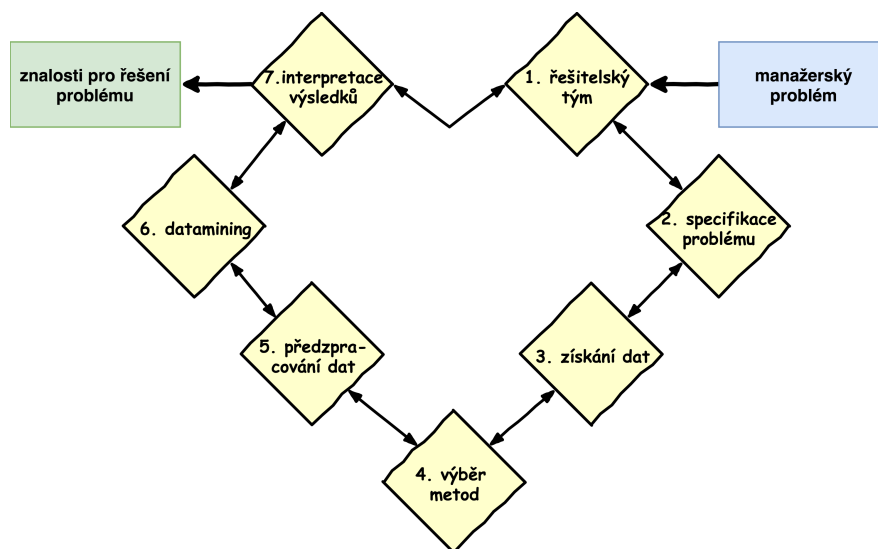
Manažerský pohled je trošičku jiný - nesoustředí se totiž na technickou stránku, ale spíše na organizaci celého řešení. Z pohledu managementu je na začátku problém, který je potřeba vyřešit. S problémem je spojeno určité zadání specifikující, v čem problém spočívá a také jaké jsou očekávané výstupy - např. z pohledu přesnosti, rychlosti, popř. jiných nároků, které na budoucí řešení máme.

Určí se leader projektu - může to být sám manager, ale mnohem častěji se jedná buďto o odborníka v doméně problému (odborník na problém) nebo odborníka na data mining. V rámci data miningu jsou často zpracovávány údaje, které jsou oborově velmi různorodé. K jejich správnému pochopení a zpracování je tak často potřeba sestavovat celé řešitelské týmy s interdisciplinárním oborovým zájmem.

Tým rozpracovává zadání tak aby získal podrobnou *specifikaci* jak problému samotného tak metrik, kterými má být poměřováno jeho řešení. Specifikace se často dělá ve spolupráci s managementem - v některých případech jsou v tomto kroku objeveny nejasnosti, které je potřeba již na začátku data miningového projektu odstranit.

Specifikace problému může také ukázat, že tým není odborně složen správně - mohou např. chybět některé klíčové odbornosti. Na druhou stranu se může ukázat, že některé odbornosti naopak nebudou potřeba.

Pro dobře definovaný problém se získávají data z různých zdrojů, o kterých se řešitelský tým domnívá, že by se v nich mohlo skrývat řešení. Následně jsou *vybírány metody*, které splňují požadavky



Obrázek 6.2: Manažerský pohled na postup data miningu (adaptováno z [20])

zadání z hlediska očekávaných výstupů a také naděje na úspěch.

Řadu metod např. lze vyloučit předem z důvodu nedostupnosti dat, nebo přílišné složitosti přípravy dat.

Pro zvolené metody se dat *předzpracují*. Na takto připravená data již lze aplikovat zvolené metody. Výsledky je pak potřeba podrobit *interpretaci*, abychom zjistili, zda odpovídají našim představám o řešení nebo ne. V případě že ne lze některé kroky celého procesu opakovat, nebo opakovat proces celý ovšem s tím, že suma znalostí o problému je již větší - včetně znalosti o tom, proč naše případné předchozí pokusy selhaly.

Tedy klíčovým prvkem data miningu je *problém*. Jaké typy problémů vlastně lze řešit. V data miningu jsou obvykle rozlišovány tři typy úloh [20]:

- klasifikace a predikce,
- deskripce,
- hledání nugetů.

Klasifikace je pravděpodobně nejčastější úlohou, kterou v rámci dataminingu řešíme. V rámci klasifikace na základě dříve zaklasifikovaných příkladů klasifikujeme příklady nové.

Takový postup má celou řadu aplikací. Např. většina emailových klientů má implementovanou podporu detekce spamu. To není nic jiného než řešení klasifikačního problému zařazení zprávy do jedné ze dvou cílových tříd spam nebo ham (tedy běžná zpráva). Pro text můžeme dělat např. analýzu sentimentu s přiřazením do jedné z cílových tříd pozitivní, neutrální nebo negativní. Tímto způsobem se rutinně analyzují příspěvky na sociálních sítích, recenze produktů nebo služeb apod.

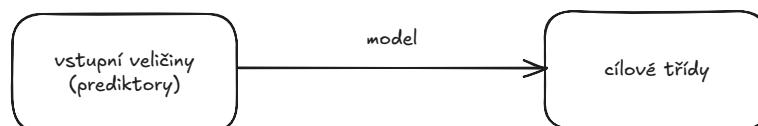
Za řešení klasifikačního problému můžeme brát také **Named Entity Recognition (NER)**, tedy rozpoznání pojmenovaných entit. I tento typ analýzy provádíme na textu, kde analyzujeme jednotlivá slova nebo několik vedle sebe stojících slov s cílem přiřadit je k některé pojmenované entitě. Právě tyto entity jsou cílovými třídami, do kterých zaklasifikováujeme. NER má obvykle schopnost v textu identifikovat objekty, místa, data, organizace, osoby a události.

V obecné rovině klasifikace znamená, že hledáme vazby mezi vstupními veličinami (mluvou statistiky prediktory) a cílovými třídami. V rámci dataminingu tedy řešení klasifikačního problému znamená formulaci modelu, který formálně popíše tuto transformaci. Graficky je zjednodušeně postup načrtnut na obr. 6.3.

Predikci pak můžeme chápat jako klasifikaci zaměřenou na budoucnost. Abychom zůstali v mantinelech předchozích příkladů může příkladem predikce být odhad reakce pacienta na léčbu (bude se uzdravovat? bude to trvat dlouho?).

Z algoritmického hlediska ale obvykle mezi klasifikací a predikcí rozdíl není, proto dále v textu budeme pro zjednodušení používat pojem klasifikace pro klasifikaci i predikci.

Z algoritmů, které bychom pro řešení klasifikačních problémů mohli použít lze zmínit např.:



Obrázek 6.3: Zjednodušený postup dataminingu

- statistika (např. regresní modely)
- pravidlově orientované metody jako rozhodovací stromy nebo pravidla
- metody strojového učení (**Machine Learning (ML)**) jako např. neuronové sítě, rozhodovací stromy
- apod.

Všimněte si, že do metod ML jsme zařadili také rozhodovací stromy, ačkoliv jsme tuto metodu už zařadili mezi pravidlově orientované metody. Je tím myšleno, že jedna metoda svými vlastnostmi může spadat do několika skupin, což je právě případ rozhodovacího stromu.

V obecné rovině můžeme *ML* definovat jako algoritmy a techniky, které umožňují počítači učit se bez explicitního naprogramování konkrétních pravidel. To je zásadní rozdíl oproti modelům produkováným klasickou statistikou.

Příkladem statistického modelu může být např. *lineární regrese*¹. Pro výpočet regresního modelu musí statistik formulovat základní strukturu modelu. Regresí se pak pouze pro tuto strukturu odvodí regresní koeficienty, které nejlépe odpovídají datům.

V ML sice obvykle také nastavujeme parametry, ale ty se týkají nastavení třeba omezujících podmínek, nikoliv však struktury modelu - tu odvozuje stroj sám.

Pro řešení klasifikačních problémů je typické, že máme k dispozici vstupy, na základě kterých se snažíme odvodit výstup. Deskripce a hledání nugetů, ale podobným způsobem nefungují.

Deskripce řeší problém popisu analyzovaných dat. Deskripce nám pomáhá data pochopit a to v souvislostech. Řeší se tím problém kdy data mohou být pochopena v kontextu svých původních informačních systémů. Výběr a spojování dat do nových datových sad vyžaduje podrobnější průzkum. Právě toto řeší deskripce.

Vzhledem k tomu, že na základě výše uvedený popis deskripce jste si nejspíše neudělali úplně jasnou představu o tom, co deskripce vlastně je, zkusme se podívat na nějaké nástroje a postupy, na kterých bychom si to mohli ukázat:

- vizualizace dat pomocí grafů (např. vzájemný vztah veličin),
- analýzy distribuce veličin,
- shluková analýza,
- apod.

Konečně *dolování nugetů* se zaměřuje na identifikaci nových zajímavých, možná překvapivých informací v datech. Tam kde se tedy deskripce zaměřovala na získávání informací popisujících používaný datový soubor jako celek, dolování nugetů se zaměřuje spíše na minoritní část dat, která z nějakého důvodu vybočuje.

Umožňuje proto nacházet např. limity určitých postupů, např. ve smyslu hranice za kterou už neplatí model používaný pro většinu dat. Informace tohoto typu jsou často vedlejším produktem analýz řešících třeba klasifikační problémy nebo deskripce dat.

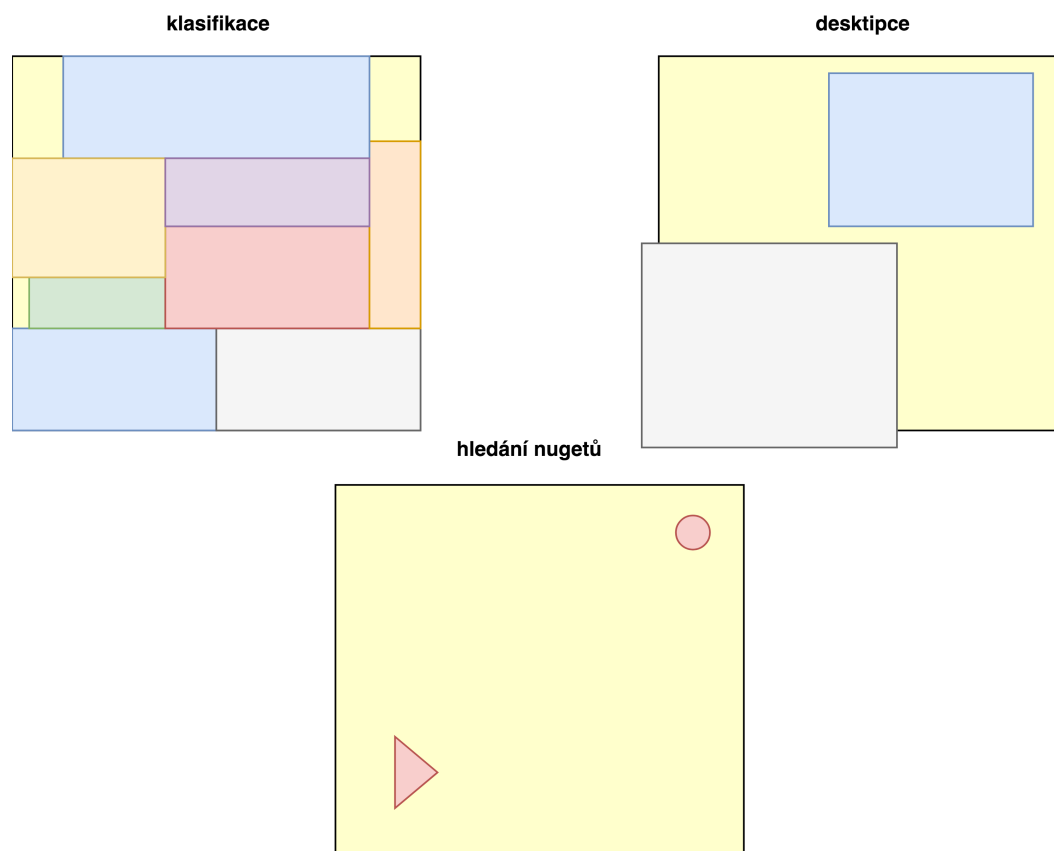
Předem se přitom dá pouze těžko odhadnout, zda takové nugety budou v datech nalezeny a jaké bude jejich podstata. Logicky můžeme dovodit, že pokud bychom na tyto otázky měli odpovědi už předem, nemuseli bychom vůbec datamining realizovat.

Problematickou hledání nugetů se v tomto textu nebudeme příliš zabývat. Z metod, které by k tomuto účelu bylo možné použít lze ale zmínit např. asociativní pravidla.

Graficky si lze představit tyto tři typy úloh jako na obr. 6.4. Uvažujme na chvíli o těchto problémech jako o stěně, kterou máme postavit. Prostor, který má stěna pokryt označíme jako koncept. Jednotlivé stavební prvky stěny nám budou sloužit pro pokrytí tohoto konceptu. Koncept je reprezentován na obr. žlutou barvou.

V případě klasifikace je požadováno, aby koncept byl pokrytý co možná nejlépe. Jednotlivé „stavební bloky“ by se neměly pokud možno překrývat. Nevýplněná místa, popř. překryvy totiž vedou ke snížení

¹problematika lineární regrese (a řada dalších věcí) byla probírána v předmětu *Statistika*



Obrázek 6.4: Úlohy data miningu - pokrytí konceptu (adaptováno z [20])

použitelnosti klasifikace. Příklady, které se nacházejí v nepokryté části konceptu nemohou být zaklasifikovány. Překryvy mezi množinami pak způsobují nejasnosti v tom, kam přesně příklad zaklasifikovat.

V reálných datech se takřka vždy objevují nepřesnosti, které způsobují neostrosti mezi množinami. Získat takovou množinu dat, která skutečně úplně pokrývá hledaný koncept je také netriviálním problémem.

To je také důvodem toho, proč je příprava dat v procesu dataminingu tak náročná. Uvádí se, že až 80 % práce v celém procesu je právě s daty a jejich přípravou. Z výše uvedeného nám také vyplývá potřeba *validace*, tedy formální ověření, že to co jsme odvodili věrně (tedy s dostatečnou přesností) reprezentuje hledaný koncept.

Pro maximální pokrytí konceptu proto preferujeme větší množství množin - preferujeme tedy přesnost před jednoduchostí.

Oproti tomu deskripce preferuje identifikaci skutečně nosných vlastností konceptu. Deskripce tak nevyžaduje, aby koncept byl pokrytý opravdu celý. Přípustné mohou být také určité překryvy mezi množinami, popř. situace kdy identifikovaná množina přesahuje hranice hledaného konceptu.

Jelikož deskripce nám primárně popisuje data. Validace zde až na výjimky nemá smysl. Výjimku by mohly představovat třeba shlukové analýzy, které formálně řadíme do metod deskripce, ale umožňují také řešení klasifikačních problémů. V případě použití pro klasifikaci pak musíme validaci provést.

Hledání nugetů přichází s novými, překvapivými zjištěními. Ta jsou z hlediska validace specifická. Jelikož se jedná obvykle o minoritní koncepty, jsou závěry vytvářeny na proporcionálně menším objemu dat než je typické pro třeba klasifikaci. To činí validaci obtížnou. Jednou z možností, kterou máme je získat další, nezávislou, datovou sadu a ověřit, že tento minoritní koncept se vyskytuje i v těchto datech.

Tímto způsobem se bráníme problému, že v datech se vždy objeví nějaké detekovatelné struktury, popř. závislosti pouze vlivem náhody. Pokud se tak stane, pak by taková struktura/závislost neměla být detekována v nezávislé sadě dat.

6.2 Metodiky dolování dat

Metodikami rozumíme postupy, které implementují jednotlivé nástroje používané pro data mining. Postupy v předchozí podkapitole jsou tedy univerzální, avšak různé nástroje je implementují různým způsobem.

V tomto výkladu se zaměříme na tři takové metodiky:

1. 5A - implementovanou původně v SPSS,
2. SEMMA - implementovanou v SAS a
3. CRISP-DM - nezávislý standard.

5A metodologie

5A je zkratka složená z počátečních písmen Assess, Access, Analyze, Act a Automate popisující kroky metodologie.

Assess (zhodnot) znamená potřeb data miningového projektu. Hovoříme specifikaci problému, možných zdrojů dat a také našich očekávání stran budoucího řešení.

V rámci *Access (přístup [myšleno k potřebným datům])* přistupujeme k datům z různých zdrojů, vybíráme ta, která považujeme ze relevantní a integrujeme je do ucelených datových sad pro potřeby budoucích analýz.

Analyze (analyzuj) představujeme krok, ve kterém aplikujeme zvolené analytické metody na shromážděná data a získáváme výsledky.

Název dalšího kroku metodiky 5A *Act* se do češtiny překládá v použitém kontextu poměrně obtížně. Doslovný překlad je *jednej*, ale ten úplně neodpovídá tomu, jak je tento krok myšlen. 5A totiž rozlišuje mezi tzv. znalostmi a akčními znalostmi.

Toto si lze představit tak, že vybraná data podrobíme analýzám zvolenými metodami a dostaneme výsledky (v předchozích krocích 5A) a tyto výsledky interpretujeme. Všechny konzistentní interpretace nám představují znalosti. Jsou ale všechny znalosti použitelné pro řešení našeho problému? Act proto znamená identifikaci podmnožiny znalostí, která jsou akční - přímo použitelné pro řešení problému.

Konečně *Automate (automatizuj)* představuje finální krok 5A zaměřený na zautomatizování akčních znalostí pro řešení problému.

Představme si příklad problému klasifikace klientů banky na bonitní/nebonitní z pohledu přidělování úvěrů. Na základě informací o stávajících klientech banky a jejich schopnosti splácet úvěry odvodíme pravidla přidělování úvěrů a implementujeme do informačního systému banky (zautomatizujeme jej).

5A metodika používá specifickou terminologii pro *znalosti*. Pokud se podíváme na obr. 6.1, znalosti nám vznikají až na samotném konci celého procesu dataminingu interpretací výsledků modelu. Metodika 5A pracuje s pojmem znalosti jinak. Chápe je ve smyslu modelu, který odvodíme z dat. Interpretací a automatizací pak ze znalostí děláme tzv. *akční znalosti*. U těch pak předpokládáme, že je budeme prakticky nasazovat pro řešení daného problému, tedy že se jedná už o odladěné řešení daného problému.

Pro náš příklad akční znalosti vznikají tedy až v okamžiku kdy je model implementován do IS banky a používá se pro automatizované hodnocení klientů.

Je potřeba podotknout, že samotné IBM ve svém nástroji SPSS Modeler postupně od 5A upustilo a přešlo na vlastní rozšířenou verzi standardu CRISP-DM nazvanou **Analytics Solutions Unified Method for Data Mining/Predictive Analytics (ASUM-DM)**.

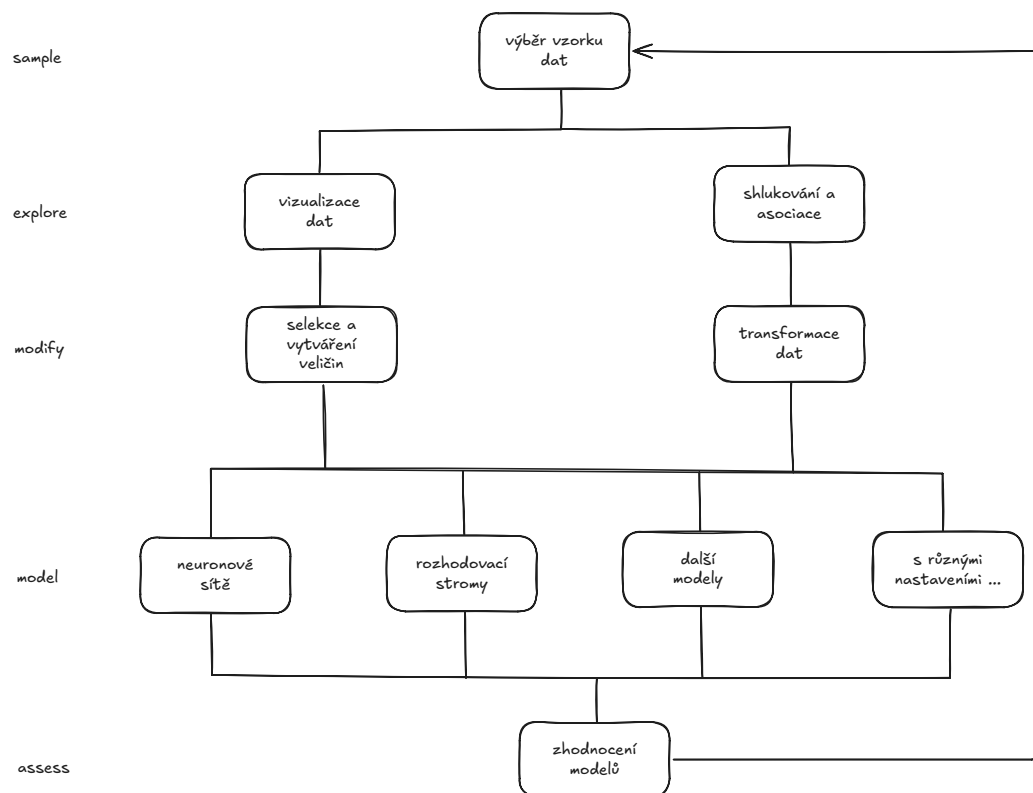
SEMMA

I SEMMA je zkratkou - konkrétně pro Sample, Eplore, Modify, Model a Assess, která nám charakterizují proces dataminingu tak, jak je analyzován analytickým nástrojem společnosti SAS, především SAS Enterprise Miner.

Pro představu je základní struktura procesu znázorněna na obr. 6.5.

Sample (výběr) - rozumíme jím výběr ve smyslu statistiky. Výběr je soubor, který obsahuje tu část prvků ze základního souboru, jejíž vlastnosti skutečně pozorujeme nebo měříme. Patří zde výběr dat z jejich původních umístění a jejich integrace do datasetů pro pozdější zpracování.

Explore (průzkum) je krok, v rámci kterého se snažíme pochopit data, se kterými pracujeme. Účelem je identifikovat možné vztahy mezi daty, možnosti případně omezení, která by mohla být relevantní pro provádění analýz. K tomuto účelu se často používají vizualizace dat, zkoumání jejich distribuce apod.



Obrázek 6.5: Proces metodiky SEMMA (adaptováno z [20])

V terminologii, se kterou jsme dosud pracovali se tento krok nejvíce blíží deskripci.

Modify (úprava) - připravuje data pro provedení analýz. Formálně odpovídá kroku transformace dat z předchozí podkapitoly. Výstupem jsou data připravená pro aplikaci zvolených analytických metod.

Zvolené metody jsou aplikovány v kroku *Model (modeluj)*. Tady je použití metodiky SEMMA poměrně specifické, úzce svázané s nástrojem SAS Enterprise Miner, pro který byla metodika vyvinuta. Tento nástroj má některé specifické vlastnosti v oblasti modelování a také hodnocení kvality modelů (*Assess*).

Nástroj funguje tak že uživatel v grafickém uživatelském rozhraní specifikuje data, která mají být zpracována a řadu různých analytických metod (ať už skutečně různých nebo pouze se změněnými parametry), které mají být na datech použity.

Následně se na základě kontrolního vzorku dat provede automatické vyhodnocení kvality použitých modelů a doporučí se ten nejlepší.

Určitou představu o způsobu fungování si lze udělat z obr. 6.6.

Všimněte si, že takový postup je jednoduše možné použít pouze v SAS Enterprise Miner. Praktická použitelnost takového postupu je tak omezená. Všimněte si také, že SEMMA se zaměřuje takřka výhradně na problematiku modelování.

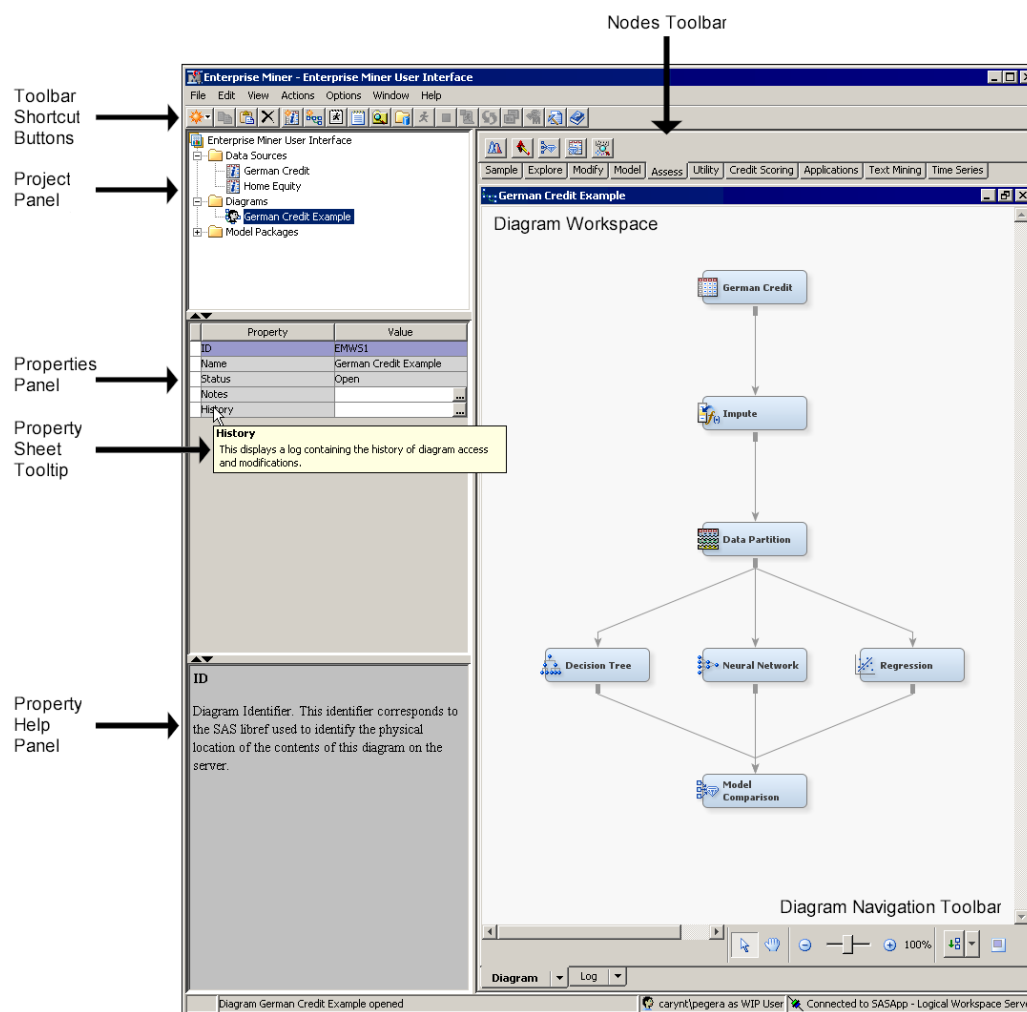
Pokud to srovnáme s např. 5A (nebo CRISP-DM dále) neobsahuje úvahy o tom, jak se data mining začleňuje do celkového fungování organizace, jak se použijí výsledky apod. Takové otázky je proto nutné řešit samostatně (nebo použít jinou metodologii práce s daty).

CRISP-DM

Metodologie CRISP-DM vznikla v roce 1996 v rámci projektu EU ESPRIT. Účelem projektu bylo vyvinout nezávislý, otevřený standard pro realizaci data miningu. CRISP-DM je zkratka pro **Cross-industry standard process for data mining (CRISP-DM)**, tedy česky mezioborový standard procesu pro data mining.

Graficky je možno postup znázornit jako na obr. 6.7.

Podle standardu je data mining realizován v šesti základních krocích. V prvním kroku pracujeme na *pochopení problému*. Problém je obvykle úzce spojen s fungováním organizace jako takové. Není tedy řešen izolovaně od fungování zbytku organizace.



Obrázek 6.6: Proces data miningu v SAS Enterprise Miner (převzato z [71])

V druhém kroku se snažíme *pochopit data* - jaká máme k dispozici a co nám říkají, jaká v současnosti k dispozici nemáme, ale z hlediska řešení by byla přínosná. Podstatnou informací je taktéž, kde přesně se data nachází a v jakém formátu.

Následně potřebná data připravíme k analýze. V CRISP-DM se do této fáze dává jak fyzický výběr dat z jejich původního umístění, tak jejich transformace do použitelné podoby.

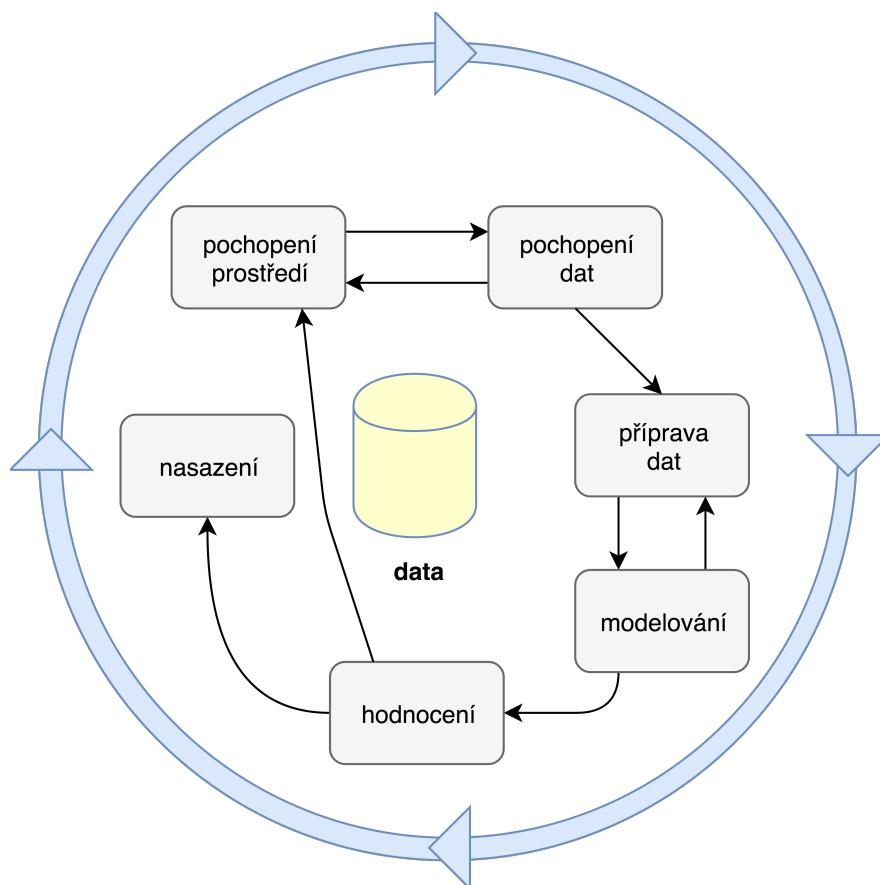
V *modelování* jsou na připravená data aplikovány zvolené analytické metody a získáváme výsledky. Tyto výsledky jsou našim vodítkem pro *hodnocení* kvality.

Byl splněn účel data miningu? Pokud ano, získali jsme dostatek informací k tomu, abychom problém vyřešili. Automatizované *nasazení* je pak logickým zakončením celého procesu.

Pokud ale nejsou výsledky dostatečné pro ukončení data miningového projektu, je potřeba identifikovat příčinu neúspěchu. Chyba mohla nastat buďto ve zvolených metodách - použili jsme nevhodné metody a nebo jsme špatně nastavili jejich parametry. Chyba ale může být přímo v datech. Data např. nemusí být dostatečně reprezentativní, aby nám vůbec umožnila problém vyřešit. V takovém případě pak řešíme problém, jak získat lepší data.

Problém mohl ale nastat v samotném našem chápání problému. To může např. znamenat, že se data miningem snažíme řešit jiný problém, než ve skutečnosti máme.

I v případě neúspěchu jsme v procesu získali lepší pochopení situace a to nám může otevřít nové cesty pro řešení problému.



Obrázek 6.7: Životní cyklus data miningu (převzato z [40])



Srovnání metodologií a postupů

Porovnejte vlastnosti metodologií 5A, SEMMA a CRISP-DM. Podívejte se jakým způsobem korespondují (implementují) kroky řešení data miningu zobrazené na obr. 6.1.



Metodika vs metoda

Rozlišujte prosím mezi pojmy *metodika* a *metoda*. V této podkapitole (a v obecně rovině i v té předchozí) jsme se zabývali metodikami. Jedná se o standardizované postupy (posloupnost činností), které vedou k určitému cíli - v našem případě k získání znalostí. Metodika obsahuje informace o způsobu použití metod a také metodické informace, např. ve smyslu jak dosažené výsledky metod interpretovat.

Takže SEMMA, 5A, CRISP-DM jsou metodiky a např. lineární regrese, neuronové sítě, baysovský klasifikátor jsou metodami. **Prosím neplést!**

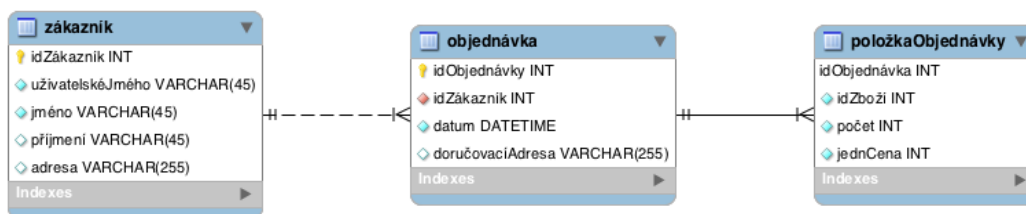
6.3 Zdroje dataminingu

Jak jsme již rozebrali výše, datamining bere údaje z jejich původních zdrojů a pak s nimi dále pracuje. Jak jsou ale taková data organizována?

V běžných provozních systémech jsou používány obvykle relační databáze. V takových databázích se vytvářejí 2D datové struktury exaktně definovaných tabulek a relací (vztahů) mezi nimi. Data samotná odpovídají jednotlivým datovým transakcím popisujícím určitý děj nebo stav reálného světa.

My jsme se problematiky databází již trochu dotkli v souvislosti s problematikou tzv. *realtimových databází* a získané poznatky nám dobře poslouží i nyní.

Jako příklad bychom si mohli uvést prodej výrobku e-shopu. V určitém časovém okamžiku dojde k potvrzení objednávky určitého množství výrobků za určitou cenu. Takové datové struktury bychom mohli zjednodušeně popsat pomocí **Entity Relationship Diagram (ERD)** na obr. 6.8.

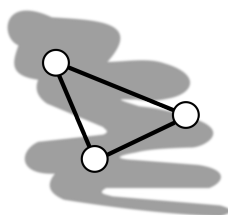


Obrázek 6.8: Příklad struktury zjednodušené objednávky

Příklad na obr. 6.8 je potřeba vnímat pouze jako ilustrativní. Pro praktickou realizaci e-shopu bychom potřebovali podstatně více dat organizovaných ve větším množství tabulek.

Provedení analýz ale vyžaduje, aby jako vstup byla předložena pouze jedna tabulka. Proto jsme v předchozích podkapitolách věnovali tolik času identifikaci potřebných dat, jejich předzpracování a transformaci.

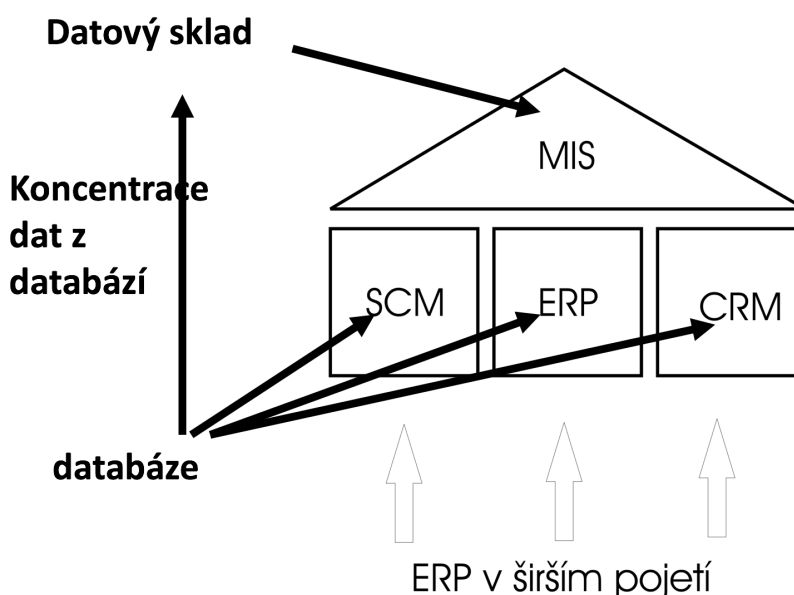
Data miningový projekt je také komplikován tím, že podobných bází dat (systémů) je potřeba jako zdroj použít často více. V takovém případě, ale vazba mezi daty z různých systémů nemusí být úplně jasná. Datový analytik v takovém případě musí pochopit dostupná data a manuálně specifikovat vazby mezi daty.



Databázová teorie

Základní informace o relačních databázích, způsobu návrhu báze dat je možno nalézt třeba ve skriptech předmětu *Bezpečnostní informatika* [91].

Ne všechny údaje jsou ale organizovány tímto způsobem. Na obr. 6.9 je znázorněna celková (zjednodušená) struktura informačních systémů středních/velkých společností.



Obrázek 6.9: Struktura informačních systémů podniku

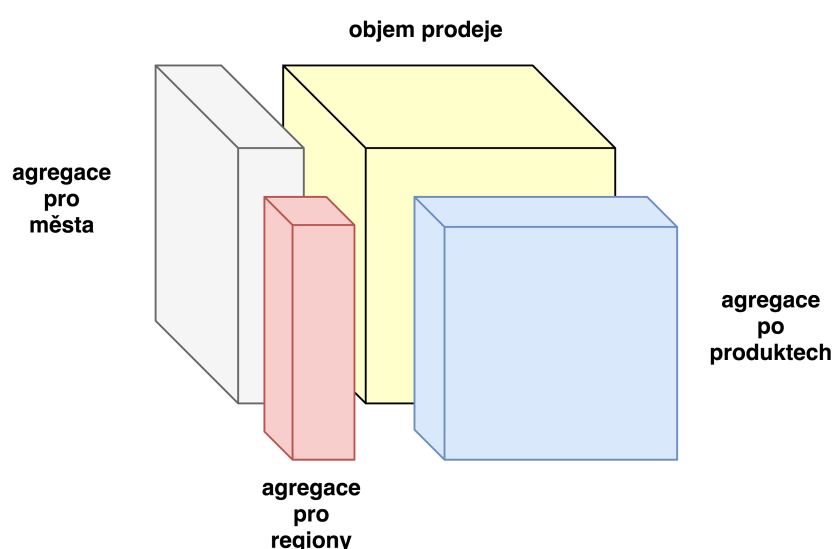
Jako základ fungování podniku jsou na obr. 6.9 ERP systémy sloužící po pokrytí běžných agend, které mají všechny organizace bez ohledu na předmět jejich činnosti. jedná se o:

- **ERP** - plánování zdrojů podniku (financování apod.)
- **Supply Chain Management (SCM)** - řízení odběratelsko - dodavatelského řetězce.
- **Customer Relationship Management (CRM)** - řízení vztahu se zákazníky

V ERP systémech jsou ukládány transakční informace. Tyto však nejsou přímo použitelné např. pro dlouhodobé řízení. Pro tyto účely jsou z ERP integrovány do **Management Information System (MIS)**. Pro tyto účely je potřeba data agregovat napříč transakcemi a získat tak pohled např. regionálně, popř. za určité období (měsíc, čtvrtletí) apod.

Odlišný způsob práce s daty je pak často výhodné organizovat odlišně než pomocí relačních databází. Takovou novou organizaci dat může poskytnout *datový sklad*. Vnitřně je datový sklad organizován jako **Online Analytical Processing (OLAP)**.

OLAP je multidimenzionálním konceptem uložení a manipulace s daty. Data jsou organizována do „krychlí“, které umožňují rychlé a pohodlné agregace. Informace prodejších by pomocí krychle bylo možné znázornit podobně jako na obr. 6.10.



Obrázek 6.10: Datová krychle v OLAP (adaptováno z [20])

Hlavní tělo krychle je na obr. znázorněno žlutou barvou. Krychli v rámci práce OLAP lze dle potřeb natáčet data agregovat podle zvolených kritérií apod. Základními operacemi OLAP proto jsou:

- natáčení (pivot),
- řezy (slice),
- výběr části krychle (dice),
- zobrazování agregovaných hodnot.

OLAP lze implementovat různým způsobem. Lze tvořit jedinou krychli, do které bude integrované všechno - takové řešení je někdy označováno jako *hyperkrychle* (hypercube). Řešení používající více krychlí jsou pak označovány jako *multikrychle* (multicube).

Hyperkrychle je náročnější z hlediska prvotní integrace dat, následná manipulace s nimi je ale pak jednodušší. Multikrychle je pak naopak jednodušší z hlediska integrace dat, ale následné propojování dat mezi krychlemi více zatěžuje OLAP server.

Rozdíl může být také ve způsobu uložení dat. **Multidimensional Online Analytical Processing (MOLAP)** je založen na řešení pomocí datové krychle nebo krychlí. Lze tedy říci, že se jedná o „čisté“ OLAP řešení.

Relational Online Analytical Processing (ROLAP) ponechává data v relační databázi. OLAP server pak pracuje tak, že vybírá data příkazy jazyka **SQL** a vytváří „virtuální krychle“. Uživatel OLAP tedy používá běžné operace manipulující s datovou krychlí. Mezi ním a daty je však navíc ještě jedna mezivrstva převádějící příkazy OLAP do SQL jazyka manipulujícího s relační databází.

Řešení různých výrobců pak mohou přicházet s dalšími variantami práce s daty. Microsoft např. pracuje ještě s tzv. **Hybrid Online Analytical Processing (HOLAP)** [53]. Většinou se však jedná o speciální případy MOLAP nebo ROLAP řešení.

Ostatní zdroje data miningu

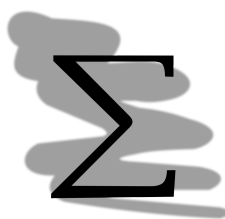
Rozvoj výpočetní techniky směrem ke zpracování velkých objemů dat se posunul natolik do předu, že v současnosti je možno zpracovávat takové objemy dat, které pomocí běžných databází nejsou zpracovatelné. Dobrým příkladem mohou být data ze sociálních sítí, informace o webových stránkách na Internetu apod.

Pro zpracování takových objemů dat byly vyvinuty nové databáze a postupy, které označujeme jako *NoSQL* databáze. Takové neorganizují data do tabulek propojených relacemi, jak je obvyklé u relačních databází. Data zůstávají „na jedné hromadě“ a jsou k nim doplňovány komplexní anotace, popisující vazby mezi daty, identifikující různé možné použití apod.

My v předmětu Bezpečnostní informatika 3 (Informatika v bezpečnosti) takové objemy dat zpracovávat nebudeme, proto berte výše uvedenou informaci jako doplňkovou.

Posledním možným zdrojem pro data mining jsou výsledky data miningu. Proces dolování tedy může být více stupňový. Můžeme použít modely pro doplnění některých dat, nebo jejich jiné zpracování. Výstupy těchto metod pak mohou otevírat nové cesty pro odvozování znalostí z dat.

Při aplikaci metod je ale potřeba mít na paměti, že data samotná i výsledky metod jsou vždy zatíženy určitou chybou. Jinými slovy nelze si vymyslet data a pak předpokládat, že výsledek data miningu bude mít jakoukoliv vypovídající hodnotu.



Shrnutí

Data mining umožňuje analyzovat velké množství různorodých dat a nacházet v nich nové, užitečné znalosti. Obvykle se postupuje tak, že se shromáždí data z různých zdrojů, připraví se do potřebného formátu vyžadovaného zvolenými analytickými metodami, které se na připravená data aplikují. Výsledky metod jsou pak interpretovány ve světle problému, který se snažíme data miningem vyřešit.

Data mining provádíme obvykle pomocí specializovaných analytických nástrojů. Většina z nich implementuje některou z metodik data miningu. V současnosti nejrozšířenější metodikou je CRISP-DM umožňující zasadit data mining do celkového kontextu fungování organizace, pro řešení jejich „business problémů“.

Vstupy pro data mining mohou být organizovány různě. Největší část dat je obvykle dostupná v různých provozních informačních systémech organizace. Tyto systémy obvykle využívají jako databázový backend relační databázi strukturující data do tabulek a vazeb mezi nimi (relací).

Při větších objemech dat, vyžadujících provádění periodických a složitých agregací je možné budovat tzv. datové sklady. Ty jsou často organizovány do podoby datové krychle umožňující použití **OLAP** systémů pro realizaci analytických operací nad nimi.



Otázky

1. Vyjmenujte základní kroky data miningu.
2. Jaký je rozdíl mezi metodikou a metodou?
3. Jaký je rozdíl mezi relační databází a OLAP?
4. Jaké má výhody použití OLAP?
5. Jaký je nejčastější problém řešení data miningem a definujte jej.



Odpovědi

1. Selekcce, předzpracování, transformace dat, dolování (modelování), interpretace výsledků.
2. viz vykřičník mezi kapitolami 6.2 a 6.3.
3. Relační databáze organizují data do tabulek (a spojení mezi nimi), OLAP pracuje s datovými krychlemi.
4. OLAP umožňuje jednodušeji provádět nad daty agregace např. geograficky, objektově nebo časově.
5. Klasifikace. Řeší přiřazení hodnoceného případu do jedné z cílových tříd na základě vstupních údajů.

Kapitola 7

Úvod do R



Náhled kapitoly

Pozor - tato kapitola je zaměřena prakticky. Proto předtím, než vůbec začnete studovat R jako nástroj jej nainstalujte na svůj počítač a jednotlivé probírané kroky rovnou zkoušejte na počítači. Zjistíte, že to, co ve skriptech není příliš jasné, se náhle vyjasní při praktické aplikaci.

Nebojte se experimentovat. Skripta jsou vždy do určité míry omezená a mohou tak obsahovat vždy pouze základy. Ilustrační příklady ale lze jednoduše dále rozšiřovat. Experimentováním jednak utužíte znalosti, jednak lépe proniknete do probírané problematiky.

Po prostudování této kapitoly budete mít

- k dispozici špičkové analytické prostředí
- základní znalosti o filozofii použití R

umět

- řešit některé základní problémy v R (načítat data, manipulovat s nimi, vykreslovat grafy)



Čas pro studium

Prostudování této kapitoly může proběhnout buďto velmi rychle nebo také velmi pomalu. Jako autor skript Vám doporučuji, postupovat spíše pomalu a seznámit se s fungováním R. Čas, který do nástroje investujete se Vám bohatě vrátí při zpracování dat pro diplomovou práci a v praxi při zpracování libovolných dat.

7.1 Instalace prostředí

Před započítím práce bude potřeba připravit si pracovní prostředí. Během výuky budeme používat prostředí R a také editor RStudio. Nejprve provedeme instalaci samotného R. Prostor je dostupné prakticky pro všechny používané operační systémy na <https://www.r-project.org/>. Stažení se provádí z CRAN. CRAN je síť serverů (zrcadel) sloužících pro obsluhu základní instalace R a také jeho toolkitů.

Jednotlivé servery jsou rozmístěny po celém světě, čímž je zajištěno, že stahování probíhá z místa, které je nejbližší uživateli (a také je tím řešena vzájemná zastupitelnost v případě výpadku).

Základní instalace prostředí je dostupná Windows, Linux a Mac.

MS Windows

Přímý odkaz je <https://cran.r-project.org/bin/windows/base/>. Pro instalaci je potřeba stáhnout a spustit instalátor zvolené verze. Pro instalaci doporučujeme vždy poslední stabilní dostupnou verzi (v době psaní těchto skript to byla verze 4.0.2).

Po dokončení průvodce, budete mít k dispozici základní instalaci R, včetně jednoduchého editoru. Prostředí bude mít schopnost instalovat rozšiřující balíky, ale pouze v případě, že jsou dostupné v binární podobě. Bohužel velká část toolkitů je dostupná pouze v podobě zdrojového kódu, který je před použitím potřeba zkompileovat.

R tuto činnost provádí automatizovaně (z pohledu uživatele je většinou jediným rozdílem trochu delší doba instalace toolkitu) - potřebuje ale mít nainstalovaná navíc RTools - ty jsou dostupné ke stažení z <https://cran.r-project.org/bin/windows/Rtools/>.

Po dokončení instalace RTools je možno ještě pokračovat dále instalací MikTeX, InnoSetup a Perl - tyto balíky ale nejsou povinné a pro naši práci je potřebovat nebudeme.

Ze stránek <https://posit.co/download/rstudio-desktop/> stáhneme a nainstalujeme *RStudio Desktop* - z dostupných edicí zvolíme Open Source Licenci, která je dostupná bezplatně.

Dostupné jsou i další zpoplatněné verze, které doplňují podporu a určitou pokročilou funkcionalitu pro práci v týmech, tvorby větších projektů apod., které pro náš způsob použití nebudeme potřebovat.

Po dokončení instalace RStudia máme k dispozici vše potřebné pro zahájení práce.

Mac (OS X)

Instalace pro Mac probíhá ve skutečnosti velmi podobně jako instalace pro Windows. Základní prostředí R instalujeme z balíku (.pkg souboru) staženého z <https://cran.r-project.org/bin/macosx/> a RStudio z <https://www.rstudio.com/>.

Pro podporu kompilace je potřeba ale doinstalovat podporu pro Clang a Fortran, dostupnou z <https://cran.r-project.org/bin/macosx/tools/>. Podporu Tcl/Tk není potřeba moderních instalací R řešit (je vestavěná). I tyto balíky stáhneme a nainstalujeme.

Linux

Verze R a RStudia je dostupná v repozitářích řady distribucí Linux. Tato verze ale nutně není nejaktuálnější. Pro naše účely by ale měla stačit.

Pro Ubuntu (a jiné distribuce založené na Debian) by mohla instalace z příkazové řádky vypadat následovně.

Výpis 7.1: Příklad instalačního skriptu pro Bash instalujícího nástroje R a RStudio pro distribuce OS Linux na bázi Debian Linux

```
1 sudo apt update
2 sudo apt-get install r-base
3 sudo apt-get install gdebi-core
4 cd ~/Downloads
5 wget https://download1.rstudio.org/rstudio-xenial-1.1.456-amd64.deb
6 sudo gdebi rstudio-xenial-1.1.456-amd64.deb
```

Prosím berte v úvahu, že Vaše instalace se může drobně lišit např. podle verze RStudia, kterou stáhnete. Místo instalace z repozitářů Vaší distribuce Linux, lze také přidat oficiální repozitář R podle návodu [65] a R instalovat z něj.

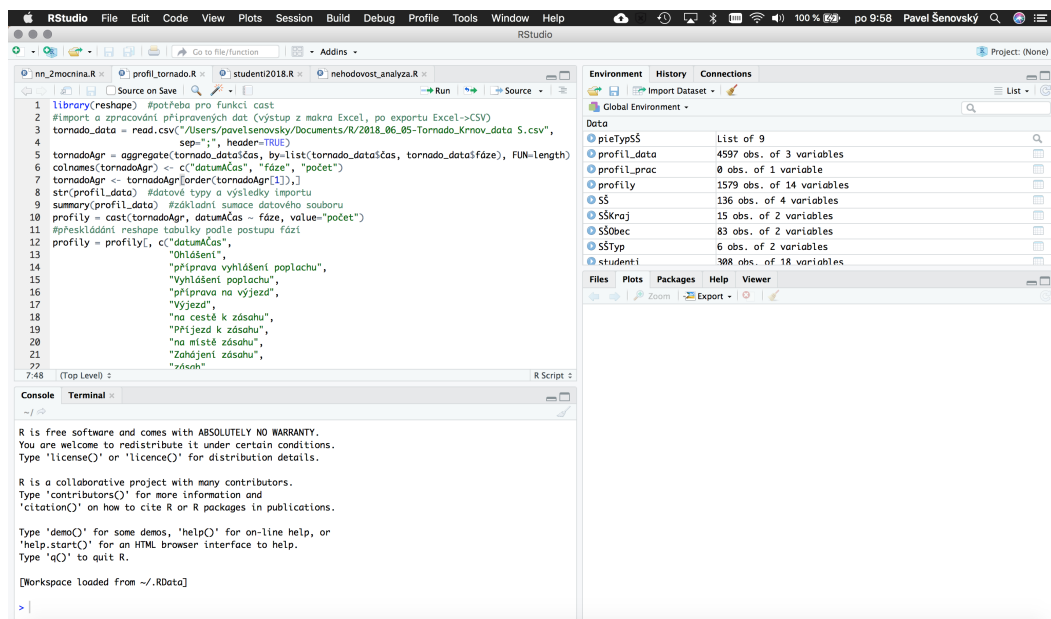
Podle stavu Vaší instalace Linuxu, mohou být vyžadovány také některé další balíky. Linux ale obvykle velmi dobře signalizuje, které balíky mu chybí. Takže odstranění těchto problémů by mělo být relativně bezbolestné.

Distribuce Linux navíc obsahují nástroje vyžadované pro kompilaci toolkitů, takže instalace dalších podpůrných balíčků obvykle není vyžadována.

Oficiálně podporované jsou pak i distribuce RedHat (Fedora) a SuSe. Pokud používáte jinou distribuci Linux, bude nutné provést nejspíše kompilaci ze zdrojových kódů.

7.2 RStudio

Zkusme prozkoumat RStudio. Po spuštění se zobrazí rozhraní podobné obr. 7.1. V mém případě je RStudio provozováno na Macu (OS X), podle zvoleného operačního systému se proto vzhled může drobně lišit. Funkčně, ale mezi různými operačními systémy rozdíl není.



Obrázek 7.1: Rozhraní RStudio v OS X

Rozhraní RStudio má čtyři základní části:

- zdrojové kódy našich analytických projektů, popř. prohlížeč dat (vlevo nahoře)
- konzole R (vlevo dole)
- prostředí (s jednotlivými načtenými datasety) - vpravo nahoře
- soubory, grafy, nápověda ... vpravo dole

Nejvíce času strávíme zápisem příkazů jazyka R - tedy v části vlevo nahoře. Zde můžeme mít otevřenu řadu záložek obsahující soubory .R, se kterými právě pracujeme.

Např. na obr. 7.1 je znázorněn soubor řešící vytvoření časového profilu výjezdů jednotek HZS k zásahům vyvolaným tornádem v Krnově (2018). Editor podporuje zvýrazňování syntaxe, poskytuje bublinkovou nápovědu k jednotlivým parametrům používaných funkcí a také nápovědu podobnou intelisence - napovídající např. jména samotných funkcí.

Zapsané příkazy lze pak spouštět buďto jednotlivě po řádcích, postupným klikáním na tlačítko *Run*, nebo lze označit segmenty kódu, popř. celý obsah souboru a *Run* pak provede všechny takto označené kroky.



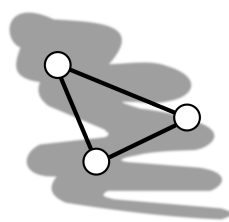
Kód R

Kód v R za určitých okolností dokáže být náročný. V tomto předmětu ale v R nebudeme programovat v tradičním smyslu tohoto slova. Kód je proto potřeba si představit spíše jako posloupnost funkcí tabulkového procesoru, ale bez nutnosti věci naklikat.

Z mnoha pohledů je R vlastně jednodušší než Excel. Na rozdíl od Excelu totiž R neřešíte, co se má dít na úrovni jednotlivých polí tabulek, ale co se má dít s daty jako takovými. Tento způsob práce je mnohem jednodušší, protože přesnou specifikaci postupu necháváte na R.

Efektivní použití nástroje ale vyžaduje, aby jeho uživatel provedl myšlenkový skok a akceptoval způsob, jakým R pracuje. To může být zejména v počátcích práce problém.

Okno konzole obsahuje interaktivní konzoli R, do které lze přímo zadávat příkazy. Většina příkazů se zadává prostřednictvím kódu v editoru vlevo nahoře. Spouštění kódu pak generuje výstupy v konzoli. V případě, že ale chceme realizovat jednorázovou akci - např. instalaci nové knihovny, lze příkaz zadat do konzole přímo.



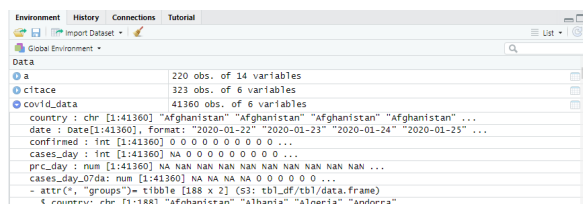
Proč R?

... a ne třeba Excel. MS Excel je skvělý produkt, ale v případě zpracování velkých objemů dat není právě nejpříjemnější. Excel také neobsahuje plnohodnotnou podporu statistických nástrojů, nebo nástrojů data miningu. K těmto účelům jej tak lze použít, ale uživatel s řešením těchto problémů příliš nepomůže.

Oproti tomu R prostřednictvím různých toolkitů má k dispozici celou škálu nástrojů pro řešení různých typů problémů, bez velikostních omezení. Produktivita práce je tak obvykle v případě R vyšší než u tabulkových procesorů. Cenou za rychlost je nutnost naučit se nové funkce (nebo se je naučit „vygooglovat“ nebo vykonzultovat s AI).

Při výkladu samotného jazyka tak budeme zkoumat jednak funkce R, jednak jejich výstup.

Vpravo nahoře je dostupný výpis prostředí R - zejména jaké proměnné jsou definovány se základními informacemi. V seznamu proměnných je vidět počet pozorování z počtu proměnných. To lze interpretovat jako počet řádků ve sloupcích - např. 4597 obs. of 3 variables znamená, že tabulka obsahuje 4597 řádků a 3 sloupce. Kliknutím na modrou šipku vedle názvu proměnné můžeme získat seznam jejích sloupců včetně základních informací o datovém typu sloupců a náhledu jeho hodnot, viz obr. 7.2.



Obrázek 7.2: Podrobnosti o sloupcích tabulek, matic a dataframů v RStudio

Dvojitým kliknutím na název proměnné ji lze spustit v prohlížeči. Např. proměnná *tornado_data* vypadá v prohlížeči jako na obr. 7.3. Spuštění prohlížeče dat je možné provést taktéž kliknutím na ikonu tabulky, která le vždy u proměnné vpravo, viz obr. 7.2.

	NázevJPO	fáze	čas
1	stanice Krnov - 811011	Ohlášení	18.6.2013 17:30
2	stanice Krnov - 811011	Vyhlášení poplachu	18.6.2013 17:30
3	stanice Krnov - 811011	Výjezd	18.6.2013 17:31
4	stanice Krnov - 811011	Příjezd k zásahu	18.6.2013 17:33
5	stanice Krnov - 811011	Zahájení zásahu	18.6.2013 18:05
6	stanice Krnov - 811011	Odjezd na základnu	19.6.2013 4:35
7	stanice Krnov - 811011	Příjezd na základnu	19.6.2013 4:58
8	stanice Krnov - 811011	na cestě k zásahu	18.6.2013 17:32
9	stanice Krnov - 811011	na místě zásahu	18.6.2013 17:34
10	stanice Krnov - 811011	na místě zásahu	18.6.2013 17:35
11	stanice Krnov - 811011	na místě zásahu	18.6.2013 17:36
12	stanice Krnov - 811011	na místě zásahu	18.6.2013 17:37
13	stanice Krnov - 811011	na místě zásahu	18.6.2013 17:38
14	stanice Krnov - 811011	na místě zásahu	18.6.2013 17:39
15	stanice Krnov - 811011	na místě zásahu	18.6.2013 17:40

Showing 1 to 15 of 27,361 entries

Obrázek 7.3: Data Viewer - proměnná *tornado_data*

Všimněte si, že do prohlížeče nejsou načtena všechna data, ale pouze 15 záznamů. Prohlížeč totiž primárně slouží pro průzkum načtených proměnných, nikoliv jejich editaci. R předpokládá, že data byla pořízena v jiném systému. Do R se tak pouze naimportují a pak zpracují.

V prohlížeči tak pohodlně zjistíme, jak data vypadají, což nám umožní se následně efektivně rozhodnout o tom, co s nimi budeme dělat. V prohlížeči lze ale data filtrovat a vyzkoušet si tak např. různé typy výběrů dat.

To samozřejmě neznamená, že data nelze upravovat - tyto úpravy ale nerealizujeme obvykle na úrovni jednotlivých buněk tabulky. Obvykle postupujeme tak, že z existujících dat odvozujeme data nová, popř. provádíme různé druhy transformací nad daty.

Konečně vpravo dole RStudio zobrazuje případná grafika, jako jsou grafy, apod. Lze zde také prohlížet vestavěnou nápovědu apod.

7.3 Základy jazyka R

Tato sekce obsahuje funkční segmenty R kódu, jejichž funkci si můžete prakticky vyzkoušet ve vlastní instalaci R. Doporučujeme proto nečíst text „na sucho“ - ale prakticky s ním experimentovat.

Operátory a proměnné

R podporuje všechny běžné operátory: $+$, $-$, $*$, $/$, $\wedge$ (umocnění), $\%\%$ (zbytek po dělení). Pro přiřazení lze použít buďto $=$ nebo $<-$ (lomená závorka následovaná pomlčkou). My ve výpisech budeme používat právě přiřazení formou $<-$, protože se v literatuře používá častěji.

Při použití mějte na paměti, že na směru šipky přiřazení záleží. Směr šipky určí kam se přiřadí. Takže $a <- 2$ přiřadí do proměnné a hodnotu 2, ale $a -> b$ přiřadí do proměnné b hodnotu v proměnné a , výraz $b -> 2$ je pak dokonce chybný, protože 2 je hodnota, do které nelze přiřazovat.

Experimentovat můžeme v tomto případě přímo v konzoli.

Výpis 7.2: Základní oprátory R a přiřazení výsledku do proměnné

```
1 > 2 + 2
2 [1] 4
3 > 3 %% 2
4 [1] 1
5 > a <- 5
6 > b <- 3
7 > c <- a * b
8 > a
9 [1] 5
10 > b
11 [1] 3
12 > c
13 [1] 15
```

V kódu výše rozlišujte mezi řádky začínající $>$ tyto řádky odpovídají příkazům zadávaných do konzole. Řádky bez $>$ jsou pak odpovědi systému na příkaz.

Tedy máme nějaké jednoduché sčítání ($2 + 2$), zbytek po dělení ($3 \% 2$) a do proměnné a přiřadíme hodnotu 5, do b hodnotu 3, obě proměnné sečteme a výsledek uložíme do proměnné c .

Obsah proměnné zobrazíme zavoláním jejího jména.

Pro úplnost uvádíme také operátory porovnání: rovno $==$, není rovno $!=$, větší $>$, větší nebo rovno $>=$, menší $<$, menší nebo rovno $<=$.

Základní funkce

Komentáře začínají znakem $\#$. Vše so po tomto znaku na daném řádku následuje, je považováno za komentář.

Výpis 7.3: Základní funkce v R

```
1 sqrt(4) #  $\sqrt{4} = 2$ 
2 abs(-4) #  $|-4| = 4$ 
3 round(x, 2) # zaokrouhlí x na 2 desetinná místa
4 log(x) # přirozený logaritmus
5 log10(x) # logaritmus x o základu 10
6 log2(x) # logaritmus x o základu 2
7 trunc(x) # odebrání desetinných míst (zůstane celá část x)
8 ceiling(x) # zaokrouhlení x nahoru
9 floor(x) # zaokrouhlení x dolů
10 exp(x) #  $e^x$ 
```



```

11 sin(x), cosin(x), tan(x) # trigonometrické funkce
12 asin(x), acos(x), atan(x) # inverzní trigonometrické funkce

```

Použití základních funkcí je relativně jednoduché, podívejme se jak na vektory.

Vektory

Výpis 7.4: Manipulace s vektory v R

```

1 a <- c(1, 2, 3, 4) # vytvoří vektor 1, 2, 3, 4
2 b <- c(TRUE, FALSE, TRUE, FALSE) # vytvoří vektor booleanovských hodnot
3 c <- c("one", "two", "three") # vytvoří vektor textových řetězců
4 class(a) # slouží pro zjištění datového typu (výsledek je numeric, character, integer, complex,
   logical)
5 d <- as.integer(3) # vytvoří proměnnou typu integer o hodnotě 3
6 e <- rep(v, times = 3) # vytvoří vektor tak, že zopakuje první parameter (v) times krát
7 f <- 1:5 # vytvoří vektor na základě sekvence 1, 2, 3, 4, 5, tedy od 1 do 5
8 g <- seq(100, 2000, by = 300) # vytvoří vektor na základě sekvence od 100 do 2000 s krokem 300

```

Ve výše uvedeném výpisu si všimněte použití rovniček v parametrech funkcí, např. na řádcích 6 a 8. `rep(v, times = 3)` znamená že obsah proměnné `v` zopakujeme (parametr `times`) 3x. Uvnitř funkcí pro přiřazování do parametrů těchto funkcí je použití rovniček povinné, nemůžeme tedy zde použít operátor přiřazení. Rovnítko v tomto případě tedy použijeme vždy, když použijeme název parametru.

Použití názvu parametrů není povinné. Pokud jej nepoužijeme, pak ale musíme trefit všechny parametry funkce v pořadí v jakém v té funkci jsou definovány. Funkce v R mívají často velké množství parametrů a tak si obvykle nechceme pamatovat je celý seznam. Zapamatování si pár nejvýznamnějších je ale obvykle celkem jednoduché, zejména pokud tyto parametry jsou pojmenovány celkem intuitivně.

V případě použití pojmenovaných parametrů můžeme volat tyto parametry funkcí v libovolném pořadí.

Všimněte si také, že ve funkci `rep` první parametr nebyl definován. Jedná se neformální dohodu, podle které první parametr funkce představuje data, se kterými, resp. na kterých má funkce pracovat.

Základní funkcí, pro vytvoření vektoru je `c()`. Do závorek - parametrů funkce `c` - se pak dávají jednotlivé prvky, ze kterých má být vektor složen. Tímto způsobem lze skládat také vektory. Zkusme vytvořit vektory `a`, `b` obsahující čísla a následně tyto vektory spojíme funkcí `c()`. Výsledný vektor bude obsahovat všechny prvky obou spojovaných vektorů.

Výpis 7.5: Spojování vektorů v R

```

1 > a <- c(1, 2, 3)
2 > b <- c(4, 5, 6)
3 > c <- c(a, b)
4 > c
5 [1] 1 2 3 4 5 6

```

Data.Frame

Nejpoužívanější datovou strukturou v R je tzv. *Data.Frame*. *Data.Frame* si lze jednoduše představit jako tabulku. Zkusme udělat jednoduchý *Data.Frame* na základě vektorů obsahujících údaje o měsíčních prodejkách:

Výpis 7.6: Realizace Data framu spojením vektorů v R

```

1 > ID <- c(1,2,3,4)
2 > mesic <- c("Leden", "Unor", "Brezien", "Duben")
3 > prodeje <- c(15000, 20000, 125000, 40000)
4 > region <- c("vychod", "zapad", "jih", "sever")
5 > prodejData <- data.frame(ID, mesic, prodeje, region)
6 > prodejData
7 ID mesic prodeje region
8 1 1 Leden 15000 vychod
9 2 2 Unor 20000 zapad
10 3 3 Brezien 125000 jih
11 4 4 Duben 40000 sever

```

My z praktických důvodů budeme načítat většinou *Data.Frame* z externích souborů. S jednotlivými sloupci tabulky lze poměrně dobře manipulovat.

Výpis 7.7: Manipulace se sloupci Data framu v R

```

1 prodejData[2] # vypíše sloupec mesic
2 prodejData[c("mesic", "region")] # vypíše sloupce měsíc a region
3 prodejData$prodeje # vypíše sloupec prodeje
4 prodejData[1:2] # vypíše první 2 sloupce
5 prodejData[-c(2,3)] # odebere sloupce 2 a 3
6 prodejData[-3] # odebere sloupec 3
7 prodejData$ID = NULL # odebere sloupec ID
8 # přidá další řádek
9 prodejData[nrow(prodejData) + 1,] = c(5, "Unor", 25000, "vychod")
10 prodejData = prodejData[-nrow(prodejData)] # smaže poslední řádek

```

Všimněte si trošku odlišné práce buďto s označováním sloupců přes \$ nebo přes []. Zatímco notace \$ umožňuje přistoupit pouze k jednomu (pojmenovanému sloupci), pak [] umožňuje filtrovat tabulku, např.

Výpis 7.8: Aplikace filtrů na sloupce Data framu v R

```

1 > podminka <- prodejData$prodeje > 20000 & prodejData$mesic == "Leden"
2 > prodejData[podminka,] # vybere řádky splňující podmínku
3 [1] ID mesic prodeje region
4 <0 rows> (or 0-length row.names)
5 > prodejData[!podminka,] # vybere řádky, které nesplňují podmínku
6 ID mesic prodeje region
7 1 1 Leden 15000 vychod
8 2 2 Unor 20000 zapad
9 3 3 Brezen 125000 jih
10 4 4 Duben 40000 sever
11 5 5 Unor 25000 vychod

```

Podmínka je vytvořena na prodeje v lednu větší než 20 000. V našich datech, ale řádky splňující tuto podmínku nejsou - R zahlásí <0 rows>. Alternativně můžeme explicitně vypsát řádky které podmínku nesplňují (!podminka). V tomto případě se nám zobrazí všechna data.

Tabulky je možno také řadit:

Výpis 7.9: Řazení v Data framu

```

1 prodejData[order(prodejData$prodeje),] # seřadit řádky datasetu prodejData podle prodeje
2 # seřadí podle prodeje a tam kde jsou stejné podle měsíce
3 prodejData[order(prodejData$prodeje, prodejData$mesic)]
4 prodejData[, order(names(prodejData))] # seřadí abecedně sloupce datasetu

```

Všimněte si, že u řazení a filtrací rozhoduje o tom, na co se budou aplikovat čárka v [] - tedy [řádky, sloupce].

Faktory

Pokud se podíváme na sloupec *mesic* v předchozí tabulce - ve skutečnosti měsíc nutně nemusí být vyjádřen textovým řetězcem - mohli bychom jej vyjádřit také číselně: 1 - leden, 2 - únor, ...

Podobně když uvažujeme o známkách ve škole pak můžeme známku vyjádřit slovně nebo číselně: 1 - výborně, 2 - velmi dobře, ... V případě měsíců i známek nám přitom záleží na přesném mapování čísel a jejich textového označení. Každopádně se z analytického pohledu jedná o tzv. nominální popř. ordinální proměnné.

K nominálním proměnným nelze připojit „kvantitu“ - tedy, pokud hodnotám takových proměnných přiřadíme čísla, pak tato čísla jsou pouze rozlišujícím znakem, ale neumožňují porovnání. Známkování je pak dobrým příkladem ordinální proměnné - zde lze provést řazení, podle zvoleného klíče.

R tento typ transformací řeší přes faktory. Uvažujme o příkladu známkování:

Výpis 7.10: Použití faktorů v R

```

1 znamky <- c("vyborne", "velmi dobre", "dobre", "nedostatecne") # vektor typu character
2 znamky <- factor(znamky)
3 znamky

```

Funkce *factor* přiřadí převáděným položkám integer hodnotu, tato hodnota, ale není viditelná. Jak to funguje můžeme zjistit porovnáním dvou prvků vektoru známky (po převodu na faktor). Můžeme zkusit např. `min(znamky)`, ale R se bude bránit. Pomocí *factor* jsme v našem příkladu totiž převedli proměnnou typu *character* na nominální proměnnou. My už víme, že v takovém případě porovnávání nemá smysl.

Ordinální proměnnou uděláme z nominální specifikací hodnot použité škály v rámci funkce *factor*:

Výpis 7.11: Vytvoření ordinální proměnné v R

```
1 > znamky <- c("vyborne", "velmi dobre", "dobre", "nedostatecne")
2 > znamky <- factor(znamky, levels=c("vyborne", "velmi dobre", "dobre", "nedostatecne"), ordered=TRUE)
3 > znamky[1] > znamky[2]
4 [1] FALSE
5 > min(znamky)
6 [1] vyborne
7 Levels: vyborne < velmi dobre < dobre < nedostatecne
8 > max(znamky)
9 [1] nedostatecne
10 Levels: vyborne < velmi dobre < dobre < nedostatecne
```

Nejen, že v tomto případě lze vzájemně porovnávat položky, lze používat i některé funkce (např. `min` a `max`).



Známkování - změna pořadí

Z čistě formálního pohledu by v příkladu výše mělo být pro známkování použito obrácené pořadí - použijte své nově nabyté znalosti a změňte pořadí tak, aby odpovídalo způsobu jak obvykle známkujeme.

Import a export dat v R

Jednu z nejčastějších úloh, kterou potřebujeme řešit prakticky v každém data miningovém projektu je import dat. Data lze importovat z různých datových zdrojů (různých databází, MS Excel, textových souborů apod.). Při importu je potřeba vždy zohlednit omezení, která nástroj pro import má.

Jako nejjednodušší a zároveň nejspolehlivější se většinou uvádí výměnný formát **Comma Separated Values (CSV)**. Jedná se o čistě textový formát, kde řádek tabulky tvoří jeden řádek textového souboru a sloupce jsou oddělovány zvoleným oddělovačem - např. čárkou (může být však použit také jiný oddělovač).

Import z CSV souboru by mohl vypadat následovně:

Výpis 7.12: Načtení CSV souboru

```
1 import_data <- read.csv(cesta, sep=";", header=TRUE)
2 str(import_data)
```

V tomto případě se bude importovat soubor identifikovaný cestou k tomuto souboru. Jako oddělovač sloupců je použit středník. Importovaný soubor obsahuje v prvním řádku hlavičku. Za import samotný je zodpovědná funkce `read.csv`.

Kromě této funkce existují další, které lze k tomuto účelu použít:

- `read.table` - nejuniverzálnější funkce pro import textových datových souborů
- `read.delim` - pro textové soubory s tabulátorem jako oddělovačem sloupců
- `read.csv2` a `read.delim2` - jako oddělovač desetinných míst se používá čárka

Všimněte si také funkce `str` v příkladu výše - ta je velmi důležitá protože vypíše informaci o identifikovaných datových formátech jednotlivých sloupců a celkový počet řádek. Tady je potřeba dodat, že způsob, jakým jsou data importována často neodpovídá záměrům, které s daty máme - řádky je proto potřeba často upravovat, měnit jejich formáty apod.

Pro import z relačních databází je možno použít toolkity specializované na danou databázi, např. RMySQL, ROracle, RPostgreSQL, RSQLite, RODBC. Posledně zmíněný je možno považovat za do velké míry univerzální, protože umožňuje přistoupit k databázi pomocí **ODBC** rozhraní. Přitom prakticky všechny relační databáze ODBC v nějaké formě podporují.

Analogickou operací k importu je export dat. V případě, že chceme použít CSV soubor, mohl by export vypadat následovně.

Výpis 7.13: Zápis dat do textového souboru

```
1 write.table(data_k_exp, file = "cesta k souboru.csv", sep=";", row.names=FALSE, fileEncoding = "UTF-8")
```

Export je tedy relativně přímočarý. Data musí před voláním funkce `write.table` uložena jako tabulka nebo `data.frame`. V případě, že data obsahují text s interpunkčními znaménky je potřeba specifikovat také parametr `fileEncoding` specifikující znakovou sadu, která má být použita během exportu. V našem případě je používáno UTF-8, což je více méně univerzální kódování.

Parametr `row.names` specifikuje, že se nemají při exportu používat popisky řádků. Popisky R přidává jinak k datům automaticky prostě tak, že k řádku přidá pořadové číslo tohoto řádku v datovém souboru.

Pro účely experimentování a zjednodušení načítání a ukládání souborů je vhodné nastavit pracovní cestu ke skriptům, v rámci kterého data zpracováváte, za předpokladu, že data budete mít ve stejné složce. V takovém případě lze místo cesty použít pouze název souboru a skripty budou fungovat tak, jak mají.

Nastavení pracovní cesty je možno provést v RStudio v menu: Session → Set Working Directory → To Source File Location. Soubor skriptu před tím musí být uložen na disk. V opačném případě se nastaví pracovní cesta na dočasné soubory operačního systému, což obvykle není žádoucí. Tzn. nejprve uložit skript a pak nastavit pracovní cestu.

Základní statistické funkce

Základní přehled funkcí:

Výpis 7.14: Základní statistické funkce v R

```
1 mean(x) #průměr z x
2 length(x) #počet prvků v x
3 median(x) #medián z x
4 unique(dataset) #z předloženého datasetu vytvoří vektor obsahující pouze unikátní položky datasetu
   (tedy bez opakování)
5 range(x) #vrací rozsah hodnot v předloženém datasetu (nejmenší a největší), vrací jej jako
   dvouprvkový vektor
6 quantile(x, probs=seq(0, 1, 0.25)) #R nezná quartily, v tomto případě se to obchází jednoduchou
   sekcencí od 0 do 1 po 0.25. Místo sekvence lze zadat i jednu položku
7 sd(x) #standardní odchylka (standard deviation)
8 min(x), max(x) #minimum a maximum
9 var(x) #rozptyl
10 summary(x) # poskytuje přehled základních statistik (min, první kvartil, medián, průměr, třetí
   kvartil, max), pokud u výše uvedených funkcí místo parametru ve formátu vektoru použijeme
   data.frame, provede se výpočet pro všechny sloupce framu.
```

Vykreslování grafů

R obsahuje řadu režimů a knihoven pro vykreslování grafů. V této podkapitole jsou ukázány příklady nejjednoduššího prostředí pro tvorbu grafů, které je v rámci R dostupné. V samostatné podkapitole věnované knihovně Plotly bude ale ukázány pokročilejší možnosti tvorby grafů.

Nejjednodušší funkce pro tvorbu grafů jsou následující:

- `plot` - běžné grafy
- `hist` - histogram
- `pie` - koláčový graf
- `boxplot` - graf typu box and whiskers

Bohužel tento typ grafů nevypadá vizuálně hezky. R poskytuje alternativu k tomuto prostředí pomocí knihovny `GGPlot2`, která umožňuje vytvářet graficky extrémně sofistikovanou grafiku. Cenou za větší možnosti knihovny je ale podstatně vyšší komplexita nastavení.

My v těchto skriptech nebudeme mít prostor pro zvládnutí této knihovny, případně se ale do studia můžete pustit sami. Grafické příklady (k motivaci) lze najít např. na Google při vyhledání „image ggplot2“.

Pro demonstraci můžeme použít vestavěný dataset `iris`, který obsahuje některé charakteristiky odrůd kosateců:

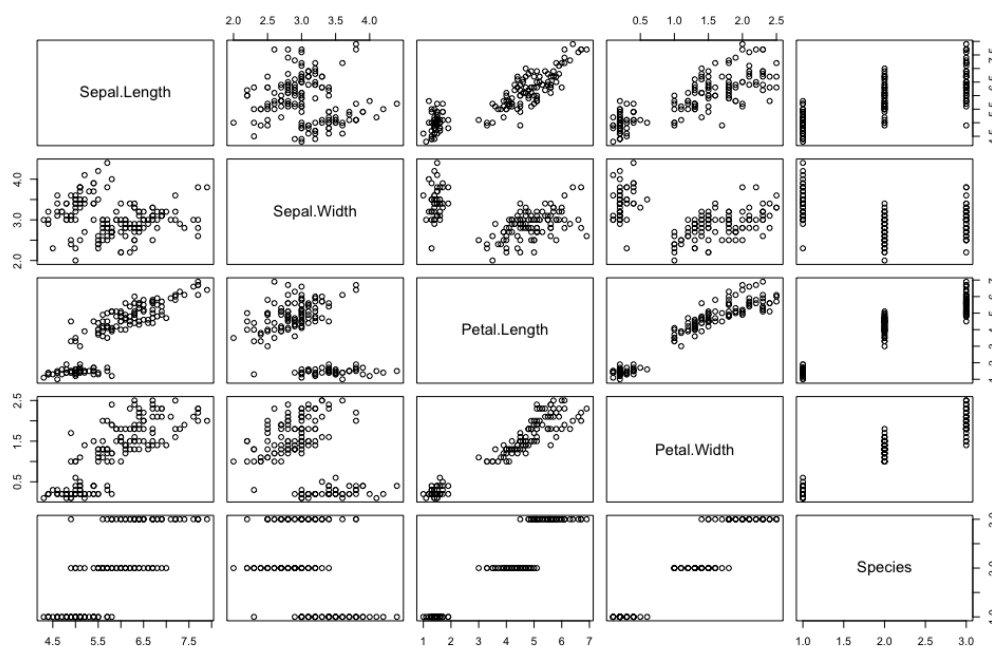
Výpis 7.15: Vykreslování základních typů grafů pro dataset IRIS

```

1 summary(iris)
2 plot(iris)
3 plot(iris$Sepal.Length~iris$Sepal.Width, col="red",
4      main="Delka vs sirka kosatcu", xlab="delka listu",
5      ylab="sirka listu")
6 hist(iris$Sepal.Length, col="blue", main="Rozlozeni delky listu",
7      xlab="delka listu", ylab="frekvence")
8 boxplot(iris, las=2,
9         main="Boxplot graf namerenych vlastnosti kosatcu", at=c(1,2,4,5,7),
10        names=c("Sep.length", "Sep.width", "Pet.length", "Pet.wight", "Species"),
11        col=c("red", "red", "blue", "blue", "green", "green"))

```

Datový soubor obsahuje popis vlastností vzorku rostlin - především pak délku a šířku listů a okvětních lístků. Funkce `plot(iris)` vykreslí párové srovnání sloupců datového souboru, viz obr. 7.4.



Obrázek 7.4: Párové srovnání vlastností kosatců v datové sadě iris

Názvy sloupců se vykreslují na diagonále grafu. Všimněte si, že grafy nad a pod diagonálou jsou podle diagonály zrcadlově obráceny.

Takový typ grafu se často používá pro rychlou vizualizaci hodnot v datovém souboru. Mnohem častěji, ale potřebujeme získat větší kontrolu nad vykreslováním grafu.

Delší forma zápisu specifikuje bodový graf, ale definuje pro něj řadu parametrů. Prvním parametrem jsou přitom data, která se mají vykreslit. Tato data se definují ve formátu `osa_X ~ osa_Y`. Oddělovačem os je pak znak tilda (vlnovka). Podle toho, co potřebujeme nastavit můžeme použít jeden nebo více parametrů.

V našem případě jsou nastavovány barva bodů (`col`), popisky os (`xlab`, `ylab`) hlavní nadpis grafu (`main`). Graf je znázorněn na obr. 7.5.

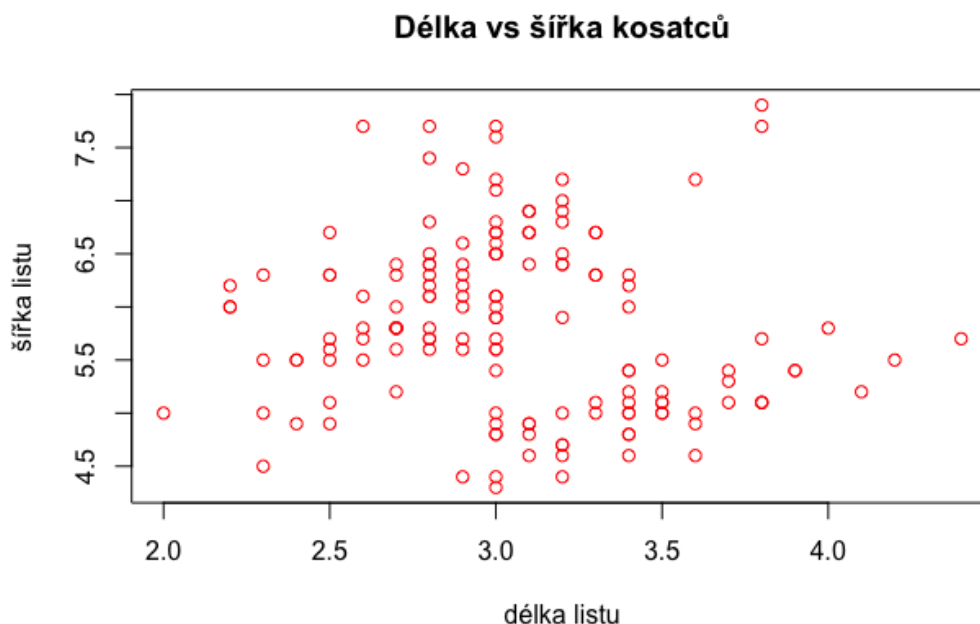
V rámci funkce `plot` lze také změnit způsob vykreslování. Iris dataset se ale k tomuto účelu nehodí. Pro demonstraci proto použijí námi v rámci skript vytvořený dataset `prodejData` a vizualizovat budeme prodeje. Všechny typy vykreslování (příkladu níže) jsou znázorněny na obr. 7.6.

Výpis 7.16: Funkce `plot` - jednotlivé typy grafů

```

1 plot(prodejData$prodeje, type="p")
2 plot(prodejData$prodeje, type="l")
3 plot(prodejData$prodeje, type="b")
4 plot(prodejData$prodeje, type="h")

```



Obrázek 7.5: Délka vs šířka listu kosatců v datasetu iris (bodový graf)

Vraťme se ale k datasetu iris - příklad histogramu je na obr. 7.7. I u histogramu lze použít řadu parametrů. Pro obr. 7.7 byla pomocí parametru `col` nastavena barva na modrou a přidány popisky os a grafu jako takového (parametry `xlab`, `ylab`, `main`).

Dalším typem grafu, který graf typu box and whiskers - viz obr. 7.8. Tento typ grafu je velmi populární pro rychlou vizuální kontrolu rozložení hodnot jednotlivých proměnných. Struktura vykreslovaného boxu je následující (zhora dolů):

- horní outliers (hodnoty $> 1.5 * \text{IQR}$, kde IQR (interquartile range) je oblast mezi 3. a 1. kvartilem)
- maximum
- 3. kvartil
- medián
- 1. kvartil
- minimum
- dolní outliers

Také funkce `boxplot` má řadu parametrů, kromě těch, které jsme použili u ostatních grafů jsem použil v příkladu parametr `las=2`, který seskupuje položky po dvojicích. Tímto způsobem jsem zdůraznil souvislost mezi položkami. Parametr `at` se zadává jako vektor popisující umístění jednotlivých položek v grafu. Všimněte si že pozice 3 a 6 jsou vynechány čímž vznikají v grafu mezery.

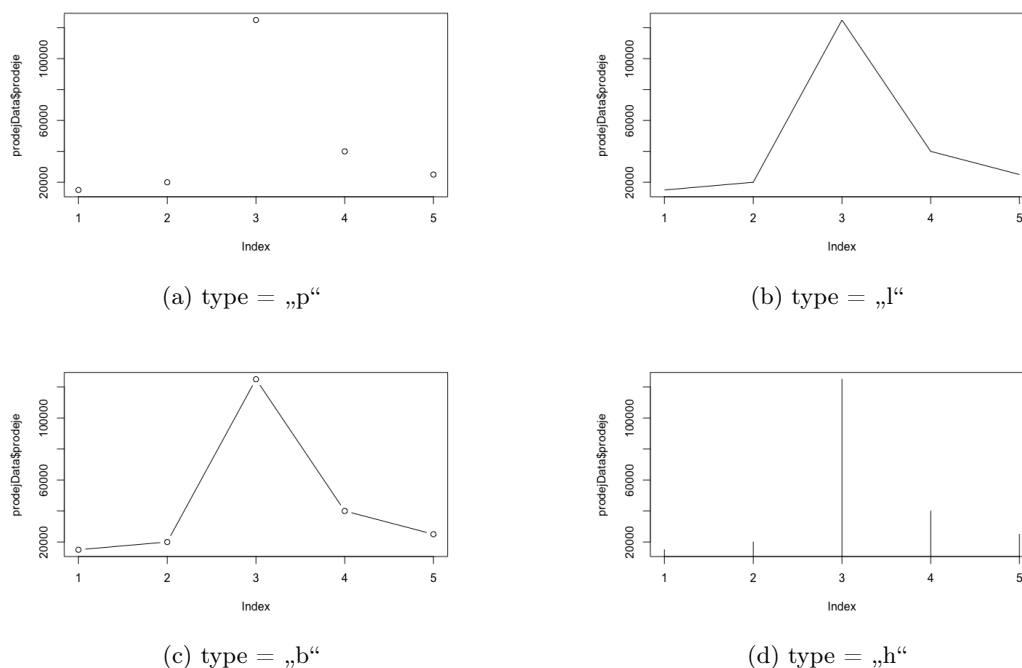
Pomocí parametru `names` lze přepsat názvy polí. Pomocí parametru `col` jsou definovány barvy. V tomto případě jsou barvy zadávány vektorem.

Poslední typ grafu, který jsem ještě neprobrali, je graf koláčový vytvářený funkcí `pie`. Tento graf je do určité míry specifický, protože vyžaduje určitou přípravu dat. Data, která jsou používána pro jeho vytvoření musí být agregovaná.

Funkce `pie` tak nemá agregaci zabudovanou přímo v sobě. Pro demonstraci použijeme náš upravený dataset prodejtů. Jeho úpravu a vykreslení koláčového grafu budeme realizovat následujícím skriptem:

Výpis 7.17: Tvorba koláčových grafů v R

```
1 vykon <- c("mizerny", "mizerny", "skvely", "prumerny", "skvely")
2 vykon <- factor(vykon, levels=c("mizerny", "prumerny", "skvely"))
3 prodejData$vykon <- vykon
4 prodeje <- table(prodejData$vykon)
5 prodeje
```



Obrázek 7.6: Dostupné typy vykreslování ve funkci plot

```
6 pie(prodeje, main="Rozlozeni prodeju")
```

Všimněte si způsobu, jakým došlo ke zpracování údajů v proměnné `prodeje`. Takové údaje je již možno zobrazit v koláčovém grafu, viz obr. 7.9.

```
mizerný průměrný skvělý
      2         1         2
```

Kromě grafů vestavěných je dostupná celá řada knihoven, které se zaměřují na sofistikovanější práci s grafy, tvorbu interaktivních grafů a celou řadu dalších problémů v této oblasti. Dobrou zkušenost mám např. s knihovnou `Plotly`, která umožňuje vytvářet velké množství různých typů interaktivních grafů. Kromě běžných grafů umožňuje např. také vizualizovat data na mapách. Interaktivitou se rozumí možnost uživatele vybrat data, o která má zájem - ať už formou odstranění některých nepotřebných řad z grafu nebo zoomem do určité konkrétní oblasti dat.

Této knihovně se věnujeme podrobněji v poslední podkapitole.

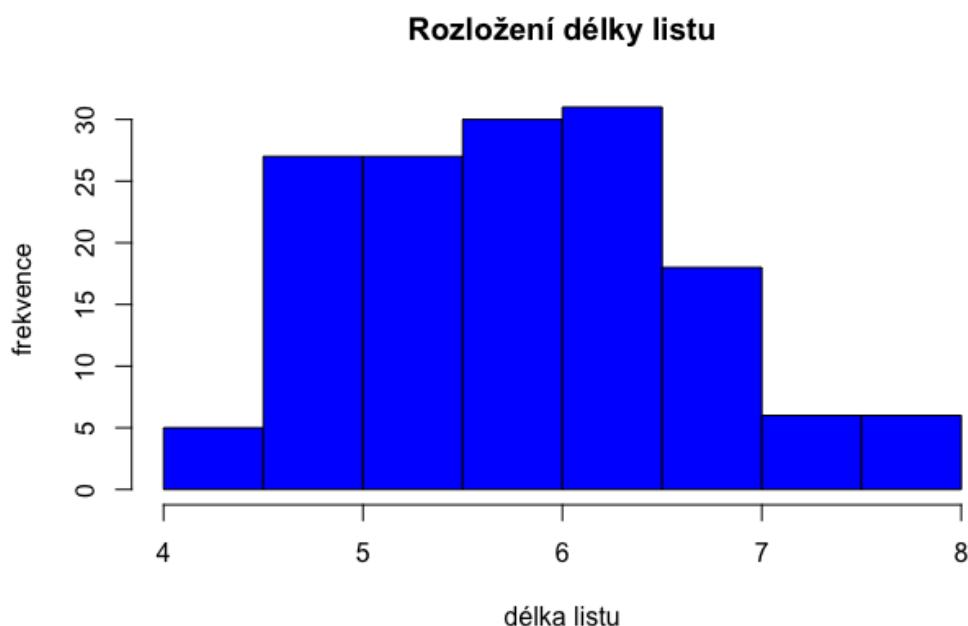
7.4 Knihovna dplyr

Knihovna `dplyr` je jednou z nejpoužívanějších knihoven v rámci jazyka R. Jejím hlavním účelem je přidání operátoru „roury“ (pipe), který mění (zjednodušuje) výrazně filozofii práce s R a také funkce umožňující jednoduché filtrování a úpravu dat.

`Dplyr` je součástí velké skupiny knihoven, které se často používají společně. Pro zjednodušení instalace a aktivace existuje tzv. metaknihovna, umožňující instalovat a nebo aktivovat všechny knihovny této skupiny najednou. Tato metaknihovna se jmenuje `tidyverse`. Instalaci a aktivaci naleznete na následujícím výpisu.

Výpis 7.18: Instalace a aktivace `tidyverse`

```
1 install.packages("tidyverse")
2 library("tidyverse")
```

Obrázek 7.7: Rozložení délky listu dataset iris (histogram)

Operátor roury `%>%` v knihovně byl svého času přelomový, ale v současnosti jeho význam již není takový, protože R podporuje operátor roury přímo v základní instalaci pomocí operátoru `|>`. Pro použití tohoto operátoru není potřeba instalace a aktivace žádné dodatečné knihovny. Ve výpisech dále je ale používán stále starší operátor `%>%`. Pro jeho funkčnost je nutno aktivovat knihovnu dplyr nebo tidyverse jako celek.

Z hlediska funkčnosti je pro naše účely oba operátory roury možno považovat za funkčně ekvivalentní.

Jelikož dplyr závisí na řadě dalších knihoven. Pro demonstraci použijeme externí data o aktuálním stavu nákazy COVID-19 dostupná z John Hopkins University.

Dostupná data se nachází ve třech samostatných datových zdrojích, které budeme postupně načítat a transformovat. K tomuto účelu použijeme v předchozí podkapitole vysvětlenou funkci `read.csv`. V tomto případě, ale budeme načítat data z webové adresy, nebudeme je tedy předem ukládat na disk.

Data jsou dostupná ve třech sadách pro:

- potvrzené případy (`time_series_covid19_confirmed_global.csv`)
- zaznamenaná úmrtí (`time_series_covid19_deaths_global.csv`)
- uzdravení (`time_series_covid19_recovered_global.csv`)

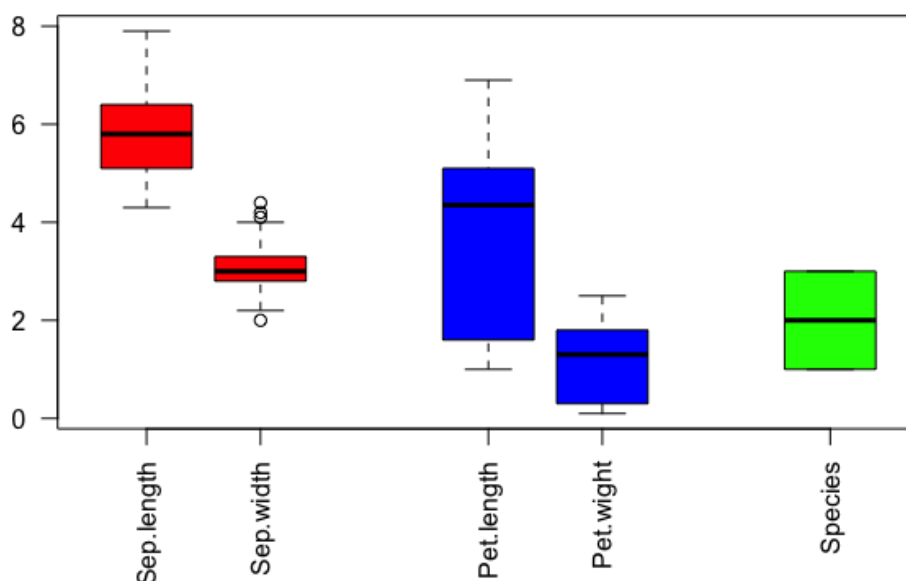
Data jsou dostupná ve formátu:

- Province.State - region státu,
- Country.Region - stát,
- Lat - zeměpisná šířka,
- Long - zeměpisná délka,
- Xměsíc.den.rok - sloupce obsahující sledovanou položku (počet potvrzených případů, úmrtí nebo uzdravených) podle souboru, který je právě zpracováván.

Výpis 7.19: Načtení dat o COVID z John Hopkins university do R

```
1 library("curl")
2 con <- curl("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_
  covid_19_time_series/time_series_covid19_confirmed_global.csv")
3 ts_conf <- read.csv(text = readLines(con))
4 close(con)
5 con <- curl("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_
  covid_19_time_series/time_series_covid19_deaths_global.csv")
```

Boxplot graf naměřených vlastností kosatců



Obrázek 7.8: Délka listu dataset iris (boxplot)

```

6 ts_death <- read.csv(text = readLines(con))
7 close(con)
8 con <- curl("https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_
  covid_19_time_series/time_series_covid19_recovered_global.csv")
9 ts_reco <- read.csv(text = readLines(con))
10 close(con)

```

Data jsou tedy v tzv. *širokém* datovém formátu. S takovým formátem se nepracuje úplně jednoduše. Široký formát znamená, že každý stát v datovém souboru má jeden řádek. Data samotná vztažená k jednotlivým datům jsou pak umisťována do sloupců Xdatum. Takový formát obvykle pro zpracování nepreferujeme - potřebujeme jej převést do *dlouhého* formátu.

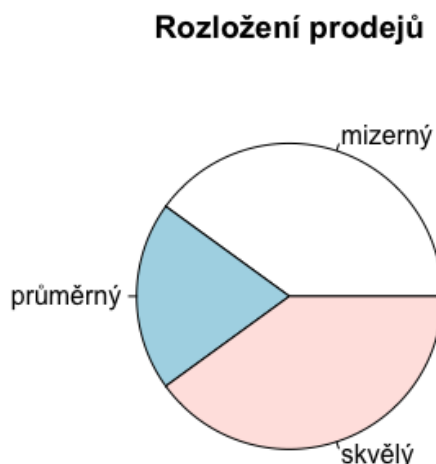
Veškeré potřebné transformace jsou zachyceny na příkladu zpracování datasetu potvrzených případů.

Výpis 7.20: Zpracování dat o potvrzených případech pomocí R

```

1 library("dplyr") #manipulace s daty
2 library("stringr")
3 library("tidyr") # tidy data
4 library("zoo") #pro výpočet plovoucího průměru
5 covid_data <- ts_conf %>%
6   select(-"Province.State", -Long, -Lat) %>%
7   gather(date, cases, -"Country.Region") %>%
8   rename(country = "Country.Region") %>%
9   group_by(country, date) %>%
10  summarize(confirmed = sum(cases))
11 covid_data$date <- str_remove(covid_data$date, "[X]")
12 covid_data$date <- as.Date(covid_data$date, '%m.%d.%y')
13 covid_data$country <- covid_data$country %>%
14   recode('Taiwan*' = "Taiwan",
15         'Korea, South' = "South Korea")
16 staty <- unique(covid_data$country)
17 newCovid <- NULL
18 for (stat in staty) {
19   a <- covid_data[covid_data$country == stat,]
20   a <- a[order(a$date),]
21   a <- mutate(a, cases_day = (confirmed - lag(confirmed, n = 1L)))
22   a <- mutate(a, prc_day = (cases_day / (lag(confirmed, n = 1L)/100)))

```



Obrázek 7.9: Rozložení počtu prodejů podle jejich úspěšnosti (pie)

```

23 newCovid <- rbind(newCovid, a)
24 }
25 covid_data <- newCovid %>%
26   mutate(cases_day_07da = rollmean(cases_day, k = 7, fill = NA))

```

Zpracování dat je pro začátečníka složitější, ale pokud se podíváme na údaje podrobněji, pak postup je poměrně jednoduchý. Nejprve je potřeba připojit jednotlivé knihovny. Knihovny dplyr a tidyr poskytují převážnou část funkčnosti, kterou využíváme pro zpracování dat.

Pomocí nástroje roura (pipe) `%>%`, popř. `|>` provádíme směřování výstupu. Např. na řádku 5 posíláme data načtená do `ts_conf` do funkce `select`, kde provádíme výběr sloupců které nás zajímají. Obcházíme tím nutnost použít řadu pracovních proměnných, do kterých bychom jinak museli ukládat mezivýsledky celého procesu.

Dplyr podporuje následující funkce:

- `mutate()` - vytvoření nového sloupce z dat v ostatních sloupcích (mutace dat)
- `arrange()` - specifikuje podle čeho se má provést řazení
- `filter()` - umožňuje specifikovat parametry výběru řádků tabulky
- `slice()` - umožňuje vybrat podmnožinu řádků na základě jejich polohy, např. vyber prvních 5 řádků z `ts_conf`: `ts_conf %>% slice(1:5)`
- `select()` - umožňuje specifikovat pozitivně nebo negativně sloupce, které mají, nebo naopak nemají být vybrány, např. `ts_conf %>% select(-"Province.State")` specifikuje, že z dat v `ts_conf` se má odstranit sloupec "Province.State".
- `rename()` - umožňuje přejmenovat sloupec nebo sloupce, např. `ts_conf %>% rename(country = "Country.Region")` provede přejmenování sloupce "Country.Region" na country. Pokud je potřeba přejmenovat více sloupců, pak se oddělují čárkou.
- `group_by()` - umožňuje provést seskupení dat podle hodnot ve specifikovaných sloupcích. Seskupení se provede tam, kde hodnoty v zadaných sloupcích jsou stejné. Seskupování provádíme obvykle tam, kde chceme aplikovat nějakou funkci - v našem případě získat kumulovanou hodnotu nakažených.
- funkce `group_by()` je pak revokována pomocí funkce `ungroup`. Tato funkce se aplikuje až v případě, kdy už data v seskupené podobě nepotřebujeme.
- `summarize()` - aplikuje sumarizační (agregační) funkci. V našem příkladu jsme použili funkci `sum()` pro součet, je možné ale použít řadu jiných funkcí `mean()` pro průměr, `median()` pro

medián, $n()$ pro počet seskupených záznamů a řada dalších.

- poslední skupinu funkcí tvoří funkce pro spojování tabulek. Z hlediska použití je nejčastější funkce `inner_join()`, která zabezpečuje vnitřní spojení, podporovány jsou ale také další typy spojení podle toho k jakým údajům se ve výsledku potřebujete dostat.

Funkce `gather()` v příkladu provádí rekódování dat z širokého formátu do dlouhého formátu.

Možná Vás napadne, co je vlastně ten široký, resp. dlouhý formát? Představte si data o vývoji počtu obyvatel v krajích. Základní náhled získáte v tab. 7.1. Aby se nám to dobře četlo, vybrali jsme jenom několik málo krajů. Údaje jsou v letech. Látá pak tvoří jednotlivé řádky. Takovému formátu říkáme široký.

Tabulka 7.1: Počet obyvatel v krajích v roce 2024 - široký formát (data ČSÚ [89])

rok	hl. m. Praha	Středočeský	Moravskoslezský	Olomoucký
2024	1 397 880	1 466 215	1 182 613	631 500
2023	...			

Stejná data můžeme převést do dlouhého formátu, viz tab. 7.2.

Tabulka 7.2: Počet obyvatel v krajích v roce 2024 - dlouhý formát (data ČSÚ [89])

rok	kraj	počet obyvatel
2024	hl. m. Praha	1 397 880
2024	Středočeský	1 466 215
2024	Moravskoslezský	1 182 613
2024	Olomoucký	631 500
2023	...	

Údaje v širokém formátu jsou dobře čitelné člověkem. Proti tomu dlouhý formát se hodí velmi dobře pro strojové zpracování. Z tohoto důvodu potřebujeme poměrně často převádět data mezi dlouhým a širokým formátem. Funkce `gather` nám v tom pomáhá.

Z výsledku našeho snažení obsaženého v proměnné `covid_data` nejprve z data pomocí funkce `str_remove()` odstraníme X. Tato funkce je obsažena v knihovně `stringr`. Výsledek je ale stále ve formátu textového řetězce. Pomocí funkce `As.Date()` jej převedeme na datum. Tato transformace je potřebná k tomu, aby data bylo možné podle data správně řadit a také k jednotlivým dnům v případě potřeby data agregovat.

Nejméně pochopitelnou částí výpisu výše je pravděpodobně cyklus. Pro jeho pochopení je potřeba si uvědomit, jak jsou vlastně data konstruována. Hlavním údajem, který je ve zpracovávaných datech ukládán je počet potvrzených případů v daném státu a v daném dni. Tento počet je ale kumulovaný, pokud nás zajímají údaje o vývoji v jednotlivých dnech, např. kolik nových případů se objevilo, pak je nutné tento údaj odvodit. K tomuto účelu je použita funkce `lag()` která umožňuje jít v časové řadě zpět nebo dopředu o zvolený počet kroku, v našem případě dní.

Z tohoto vyplývá, že počet nově infikovaných v daném dni je možno odvodit jako rozdíl kumulovaného počtu infikovaných ve dni hodnocení a stejného údaje den předem: `cases_day = (confirmed - lag(confirmed, n = 1L))`. Stejným způsobem je odvozeno procento nárůstu `prc_day`.

Konečně nad výsledkem je proveden výpočet sedmidenního klouzavého průměru případů, pomocí funkce `rollmean()` z knihovny `zoo`.

Obdobným způsobem můžeme zpracovat datové sady zemřelých `ts_death` a uzdravených `ts_recovered`.

Výpis 7.21: Zpracování dat o potvrzených případech úmrtí v důsledku COVID-19 pomocí R

```
1 covid_death <- ts_death %>%
2   select(-"Province.State", -Long, -Lat) %>%
3   gather(date, cases, -"Country.Region") %>%
4   rename(country = "Country.Region") %>%
5   group_by(country, date) %>%
6   summarize(confirmed = sum(cases))
7 covid_death$date <- str_remove(covid_death$date, "[X]")
8 covid_death$date <- as.Date(covid_death$date, '%m.%d.%y')
9 covid_death$country <- covid_death$country %>%
```

```

10 recode('Taiwan*' = "Taiwan",
11        'Korea, South' = "South Korea")
12 newCovid <- NULL
13 for (stat in staty) {
14   a <- covid_death[covid_death$country == stat,]
15   a <- a[order(a$date),]
16   a <- mutate(a, cases_day = (confirmed - lag(confirmed, n = 1L)))
17   a <- mutate(a, prc_day = (cases_day / (lag(confirmed, n = 1L) / 100)))
18   newCovid <- rbind(newCovid, a)
19 }
20 covid_death <- newCovid %>%
21   mutate(death_day_7da = rollmean(cases_day, k = 7, fill = NA))

```



Počty uzdravených

Jak je patrné z výpisu kódu v R pro zemřelé, je proces transformace dat totožný. Stejně tomu bude i s daty o uzdravených.

Zkuste proto samostatně navrhnout skript, kterým tato data transformujete.

V případě, že Vám to opravdu nejde můžete konzultovat hotový skript, který je dostupný na LMS.

Poslední operací, kterou s daty uděláme, je realizace propojení tabulek do jednoho celku. To nám umožní s daty manipulovat najednou.

Výpis 7.22: Propojení souvisejících dat do jednoho celku pomocí R

```

1 covid_join <- covid_data %>% inner_join(covid_death, by = c("country", "date"))
2 covid_join <- covid_join %>%
3   rename (
4     confirmed = confirmed.x,
5     cases_day = cases_day.x,
6     prc_day = prc_day.x,
7     death = confirmed.y,
8     death_day = cases_day.y,
9     death_day_prc = prc_day.y
10  ) %>%
11    mutate(death_prc = death / (confirmed / 100))
12 covid_join <- covid_join %>% inner_join(covid_reco, by = c("country", "date"))
13 covid_join <- covid_join %>%
14   rename (
15     confirmed = confirmed.x,
16     recovered = confirmed.y
17   )
18 covid_join$nemocni <- covid_join$confirmed - covid_join$recovered - covid_join$death

```



Dplyr - shrnutí

V této kapitole jsme odvedli velký kus práce. Naučili jsme se importovat data z Internetu a provést jejich transformaci do formátu, který můžeme dále zpracovávat. Naučili jsme se také, jak lze data obsažená v samostatných tabulkách propojovat, abychom získali ucelená data set.

Operace tohoto typu jsou pro data mining typické a zabírají velké množství času, pokud ale zvládneme funkce, které jsou nabízeny knihovnami jako je *dplyr* může být tato práce výrazně jednodušší.

V následující podkapitole budeme připravená data vizualizovat pomocí různých grafů.

7.5 Grafy pomocí knihovny Plotly

Plotly je oblíbená knihovna pro vykreslování grafů různého typu. V této podkapitole si ukážeme prakticky některé základní typy. Knihovna, ale umožňuje vykreslovat data také pomocí exotičtějších typů grafů. Grafy jsou takovým oknem, kterým můžeme data zkoumat v podobě, která je obvykle pochopitelnější než jsou data v hrubé - tabelární podobě.

Graf je tedy mocným nástrojem, kterým můžeme předat efektivně informaci a přesvědčit příjemce takové informace o její správnosti. Správné prezentaci dat je proto potřeba věnovat patřičnou pozornost.

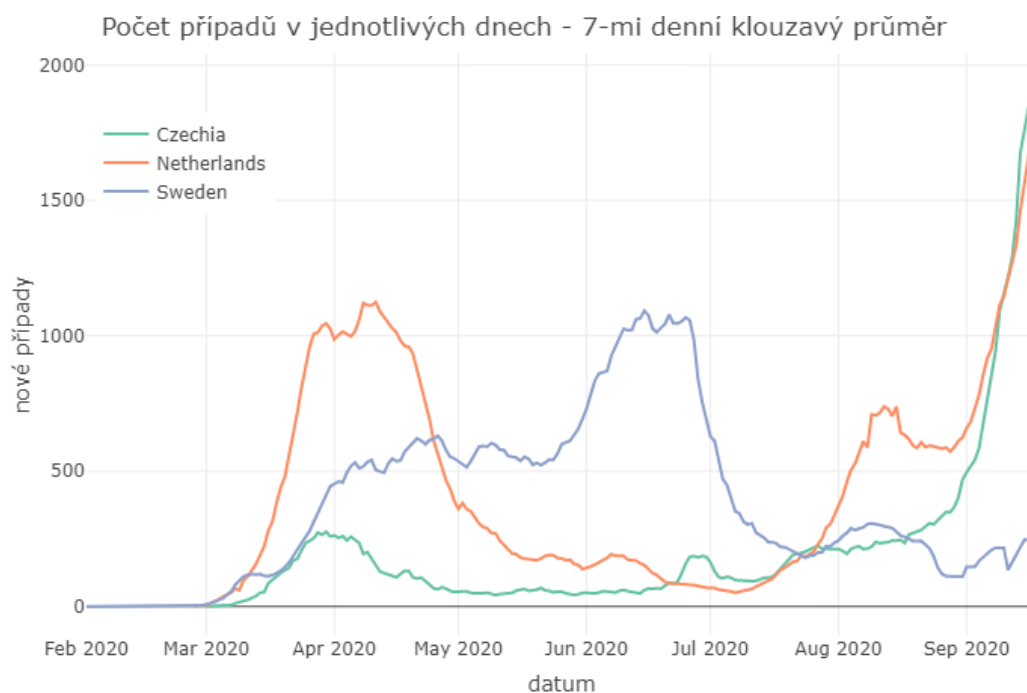
Pro knihovnu Plotly v R existuje skvěle zpracovaný tutoriál na <https://plotly.com/r/>. Tento tutoriál doporučuji konzultovat jako první v případě, že Vám nějaká data nepůjdou zobrazit způsobem, který očekáváte nebo jste prostě z hlediska svých potřeb přesáhli objem informací poskytovaných těmito skripty.

Jako první zkusme zpracovat jednoduchý graf vývoje sedmidenního klouzavého průměru počtu nemocných. Provedeme srovnání zemí, se kterými bývá někdy Česká republika srovnávána: tedy ČR, Nizozemí a Švédsko.

Výpis 7.23: Sedmidenní klouzavý průměr nakažených v ČR Nizozemí a Švédsku

```
1 covid_plot <- covid_data %>%
2   filter(cases_day > 0 & country %in% c("Czechia", "Sweden", "Netherlands")) %>%
3   group_by(country)
4 fig <- plot_ly(data = covid_plot,
5   x = ~covid_plot$date,
6   y = ~covid_plot$cases_day_07da,
7   mode = 'lines',
8   type = "scatter",
9   color = covid_plot$country) %>%
10  layout(
11    title = "Pocet pripadu v jednotlivych dnech - 7-mi denni klouzavy prumer",
12    xaxis = list(title = "datum"),
13    yaxis = list(title = "nove pripady"),
14    legend = list(x = 0.01, y = 0.9))
15 fig
```

Výsledkem je graf na obr. 7.10.



Obrázek 7.10: Počet případů COVID-19 v jednotlivých dnech - 7-mi denní klouzavý průměr

Všimněte si jakým způsobem pracujeme. Nejprve jsme si z obecných dat (`covid_data`, ale fungovalo by také použití spojené tabulky `covid_join`) vybereme údaje, která nás zajímají. V našem případě vyfiltrujeme státy ČR, Švédsko a Nizozemí. Výsledek uložíme do dočasné proměnné `covid_plot`. Alternativně bychom mohli data rovnou poslat rourou do funkce `plotly()`, která slouží pro přípravu grafu.

Graf připravujeme do samostatného objektu nazvaného v našem případě `fig`, který následně zavoláme a systém provede vykreslení grafu podle specifikovaných parametrů.

Parametry jsou následující:

- `data` - odkaz na tabulku obsahující data
- `x` a `y` - odkazy na sloupce obsahující data, která odpovídají hodnotám na osách `x` a `y`
- `mode` - mód vykreslování, v našem případě `lines`, tedy čára, může být ale také `markers` (body) nebo obojí dohromady `lines+markers`.
- `type` - typ grafu, v našem případě používáme `scatter`, což je bodový graf (XY)
- `color` - specifikuje podle čeho se mají rozlišit barvy v grafu

Samostatně se následně definuje rozložení prvků grafu jako je `title` - titulek grafu, `xaxis` a `yaxis` pro nastavení os. V našem případě nastavujeme pouze popisek a nic více. Lze ale např. nastavit, aby se použila logaritmická škála apod. Konečně položka `legend` umožňuje nastavit x-ovou a y-ovou souřadnici levého horního rohu legendy. Naše hodnota `x = 0.01` a `y = 0.9` odpovídá umístění skoro úplně vlevo a skoro nahoře.

Rozsah hodnot je vždy 0 - 1. Pravému spodnímu okraji budou odpovídat souřadnice třeba `x = 0.8`, `y = 0.1`. Umístění je ale vždy potřeba vizuálně vyzkoušet a podle potřeby dopracovat. Různě dlouhý text popisku a počet položek legendy může způsobit že třeba výše uvedené hodnoty nebudou postačovat. Pokud souřadnice neuvedeme, pak se legenda uvede vedle grafu - vůbec tedy do něj nezasáhne.

Všimněte si také, že výsledný graf je interaktivní. Pokud najedete kurzorem na křivku grafu v RStudio začne Vám ukazovat naměřenou hodnotu místa, kam ukazuje Váš kurzor. Kliknutím a tažením můžete specifikovat výřez, což umožňuje jednoduchý průzkum různých částí křivky.

To je výhodné zejména v okamžiku, kdy jsme na počátku procesu data miningu a hledáme cesty, jak lépe porozumět datům. Také nám to umožňuje pouze „na hrubo“ nastavit potřebné rozsahy s tím, že výřezem upravíme oblast grafu, která nás zajímá.

Po najetí kurzorem na graf se zobrazí také panel nástrojů (na horním okraji grafu). Zde je dostupný např. nástroj, kterým můžete výsledek Vaší práce exportovat do PNG formátu vhodného pro vložení do MS Word nebo jiného textového procesoru.



Jednoduchý graf - úprava zemí

Upravte předchozí graf tak, aby zobrazoval sedmidenní klouzavé průměry nakažených v ČR a států s ní sousedících.

Zkusme vytvořit graf, který zobrazí body ale v logaritmické škále. Využijeme toho, že exponenciální růst se pod takovým měřítkem jeví jako diagonála. Abychom tento efekt viděli na našich datech budeme vykreslovat denní přírůstek vůči celkovému (kumulovanému) počtu nakažených.

Výpis 7.24: Logaritmické osy grafu - demonstrace vizualizace exponenciálního růstu

```
1 covid_plot <- covid_data %>%
2   filter(country %in% c("China", "Italy", "Spain", "US", "Czechia", "South Korea")) %>%
3   group_by(country)
4 fig <- plot_ly(data = covid_plot,
5   x = ~covid_plot$confirmed,
6   y = ~covid_plot$cases_day,
7   mode = 'markers',
8   type = "scatter",
9   color = covid_plot$country) %>%
10  layout(
11    title = "Exponencialni rust CR, Italie, Spanelsko, USA, Cina a Jizni Korea",
12    xaxis = list(title = "kumulativni pocet pripadu",
13      type = "log"),
14    yaxis = list(title = "denni pocet pripadu",
```

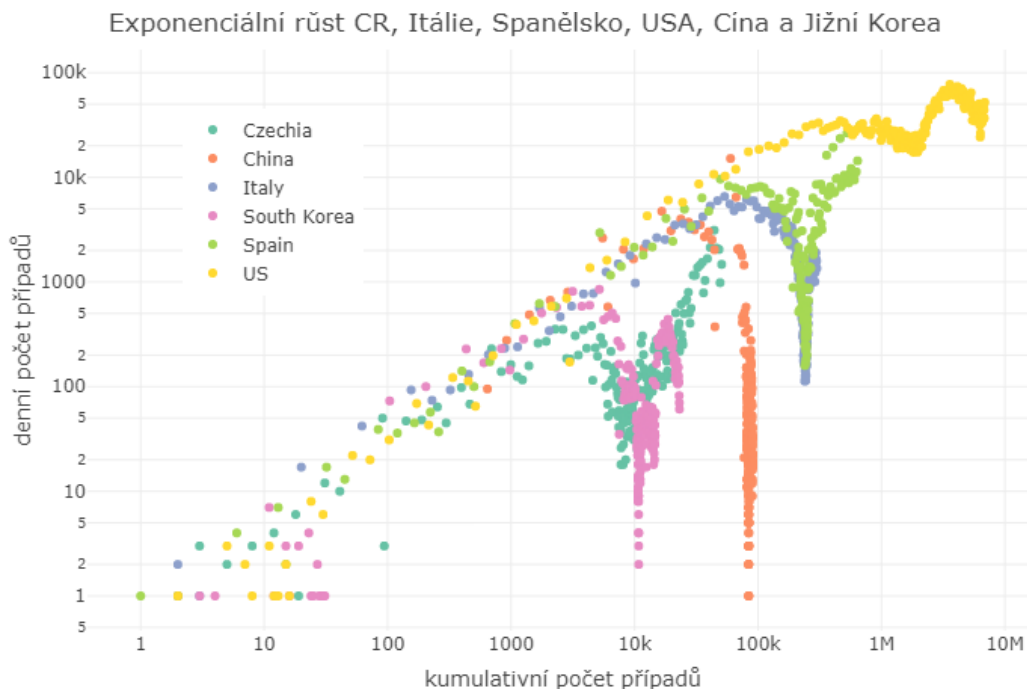


```

15     type = "log"),
16     legend = list(x = 0.1, y = 0.9))
17 fig

```

Změn je relativně málo - zajímají nás jiné státy, mechanismus výběru ale zůstal stejný. Největší změna je v layoutu grafu, kde pro jednotlivé osy kromě popisku (title) je nastavován také typ (type) a to konkrétně logaritmický. Výsledný graf je znázorněn na obr. 7.11.



Obrázek 7.11: Exponenciální nárůst nakažených COVID-19 ve vybraných státech - demonstrace logaritmických měřítek os

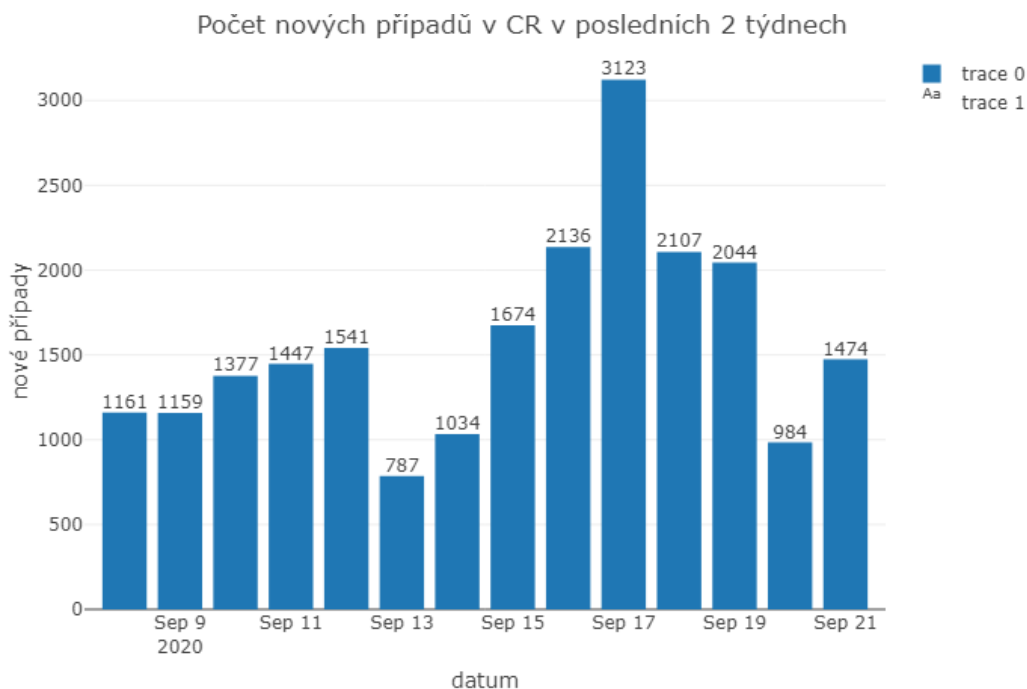
Jako poslední příklad, který v této části textu použijeme je sloupcový graf. Na našich datech zkusíme vizualizovat vývoj denního počtu nově diagnostikovaných COVID-19 v ČR za poslední 2 týdny. Postup níže by Vám již měl připadat povědomý. Výsledný graf je dostupný na obr. 7.12.

Výpis 7.25: Sloupcový graf - vývoj denního počtu nově diagnostikovaných COVID-19 v ČR za poslední 2 týdny

```

1 covid_plot <- covid_data %>%
2   filter(date > (max(date) - as.difftime(14, unit="days")) & country %in% c("Czechia")) %>%
3   group_by(country)
4 fig <- plot_ly(data = covid_plot,
5   x = ~covid_plot$date,
6   y = ~covid_plot$cases_day,
7   type = "bar") %>%
8   layout(
9     title = "Pocet novych pripadu v CR v poslednich 2 tydnech",
10    xaxis = list(title = "datum"),
11    yaxis = list(title = "nove pripady")) %>%
12   add_text(
13     text = ~covid_plot$cases_day,
14     textposition = "top middle",
15     cliponaxis = FALSE)
16 fig

```



Obrázek 7.12: Vývoj denního počtu nově diagnostikovaných COVID-19 v ČR za poslední 2 týdny



Experimentování s grafy

Jediný způsob, jak výše uvedené informace vstřebat je experimentovat. Revidujte proto předchozí příklady. Pokud např. Vás napadly otázky, jak situace vypadá v jiných státech - nyní máte nástroj a také příležitost jak to zjistit.

Pro inspiraci můžete použít také rozšířený příklad uložený na LMS.

Konzultujte <https://plotly.com/r/> - zaujaly Vás nějaké grafy uváděné v tutoriálu. Zkuste je použít na našich datech!



Místo shrnutí

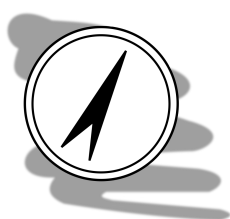
V této kapitole jsme nainstalovali prostředí R a editor RStudio. Následně jsme prošli některé z nejčastěji používaných postupů a funkcí.

Pokud jste pečlivě zkoušeli příklady uvedené v kapitole a chápete, jak systém funguje, gratuluji - zvládli jste základy R. Samozřejmě pro získání jistoty v práci je potřeba prakticky realizovat v tomto prostředí nějakou práci.

Pokud jste jednotlivé příklady nerealizovali - teď je na to vhodná doba. Začněte procházet kapitolu, pěkně od začátku a zkuste, jak funguje každá funkce, o které jsme v kapitole psali.

Kapitola 8

Pravidlové metody



Náhled kapitoly

V této kapitole se zaměříme na pravidlové metody data miningu.

Po prostudování této kapitoly budete vědět

- jak fungují pravidlové metody data miningu
- pro jaké typy problémů je vhodné je použít
- jaké jsou jejich výhody ale také omezení

umět

- prakticky použít metody *rozhodovacích stromů* a asociativních pravidel v R
- použít knihovnu *carret* pro nastavení hyper-parametrů rozhodovacího stromu pro zvýšení kvality odvozených modelů



Čas pro studium

Pro prostudování budete potřebovat 2 - 4 hodiny. V závěru kapitoly je vysvětleno použití knihovny *carret*. Tato knihovna je určena pro nastavení hyperprostoru parametru statistických metod. Tímto způsobem se můžeme elegantně zbavit závislosti na pokusech s přenastavováním parametrů modelu po jednom.

Model sám se tedy zadaptuje. Tato část kapitoly je náročnější, na druhou stranu má také pro Vás největší přínos, protože stejné principy je možno použít na většinu modelů, se kterými se v tomto předmětu setkáme a řadou dalších, jako jsou třeba regresní modely, se kterými jste se mohli potkat v průběhu studia jiných předmětů.

Pravidlové metody nám umožňují řešit problém generováním pravidel ve formátu IF podmínka THEN závěr. Forma a účel metod se ale liší.

Při výkladu se zaměříme na tři metody:

- rozhodovací pravidla
- rozhodovací stromy
- asociační pravidla

Vzhledem k tomu, že rozhodovací pravidla a stromy řeší klasifikační problémy a sdílejí řadu společných omezení a navíc je možno rozhodovací stromy na pravidla převést, probereme tyto metody společně.

V samostatné podkapitole jsou pak probrána asociační pravidla. Ty jsou určeny pro řešení problému deskripce a dolování nugetů.

8.1 Příprava dat

Všechny výše uvedené metody mají více-méně podobné nároky na vstupní údaje. Rozdíl je pouze v tom, že rozhodovací stromy a pravidla řeší klasifikační problémy. K správnému odvození pravidel je proto nutné, aby připravená data obsahovala již zaklasifikované příklady. Tato data by přitom z pohledu statistiky měla být reprezentativní vzhledem k problému, který má být data miningem řešen.

Připravenou datovou sadu by zároveň mělo být možné rozdělit do dvou menších, které můžeme označit jako *trenovací množina* a *validační množina*. Trenovací množina se používá pro „natrénování“ pravidel, pomocí validační množiny se provádí ověření kvality natrénovaného modelu.

V žádném případě není možné kvalitu modelu hodnotit na základě trenovací množiny. Platí totiž, že jelikož byl model na základě trenovací množiny vytvořen, dobře reprezentuje data v ní obsažená. Otázka ale je, zda také dobře reprezentuje problém (resp. jeho řešení).

Trenovací množina je obvykle větší než množina validační. Orientačně lze uvažovat o poměru 2:1 nebo 3:1, přitom by ale mělo platit, že obě množiny jsou vůči problému statisticky reprezentativní. Z tohoto důvodu je stanovení přesného poměru závislé na řešeném problému.

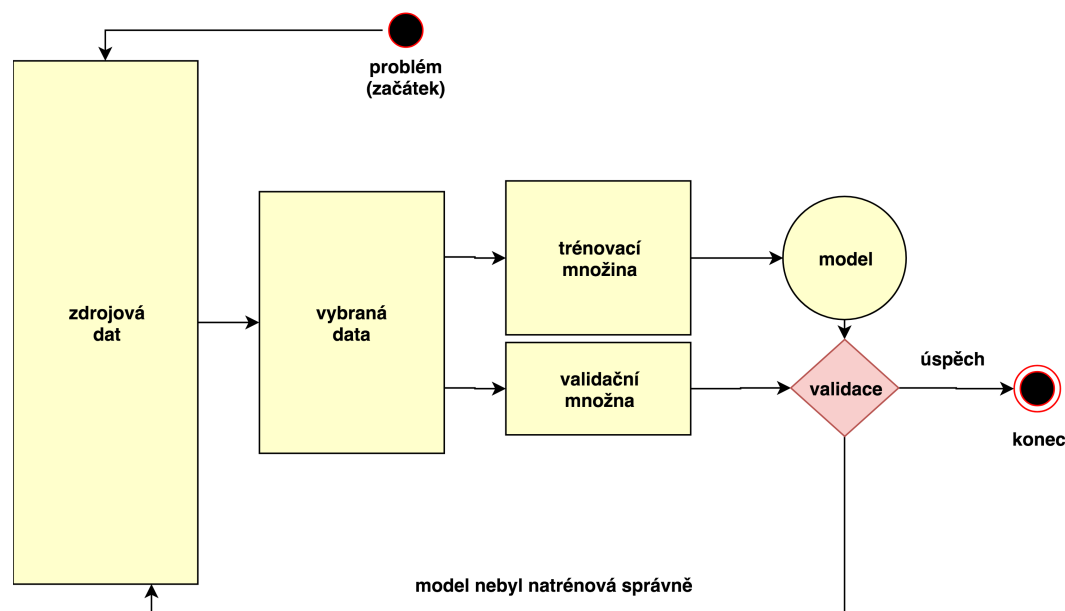
Data jsou do obou množin přidělována náhodně. Náhodný výběr by měl vést k tomu, že také chyba odvozeného modelu bude náhodná. Při nenáhodném výběru riskujeme, že do množiny zaneseme systematickou chybu a množina pak nebude správně reprezentovat problém.

Pokud takový problém vznikne v trenovací množině bude mít výsledný model pro některé oblasti hodnocení větší chybu než očekávanou. V případě, že pak takový problém nastane u validační množiny ztratíme možnost existenci takového problému vůbec odhalit.

Výše uvedená metoda validace je nejspíše tou nejjednodušší, kterou lze pro účely validace použít a zároveň, která je ještě přijatelně funkční. Pokročilejší metodou validace je metoda *křížové validace* (cross validation). Nejčastější je *k-fold křížová validace*, kde k je vždy celé číslo vyšší než 1 určující na kolik částí se datový soubor rozdělí. Např. pro $k = 5$ se datový soubor rozdělí náhodně na 5 částí, 4 se použijí pro trénink a 1 pro validaci. Vzniká nám tedy poměr 80:20, podobně jako jsme měli u validace běžné. Proti běžné validaci se ale toto rozdělení zopakuje 5-krát (tedy k -krát).

Výhodou použití je odhad výkonu modelu (chyby modelu), ale za cenu výrazně vyšších výpočetních nároků.

Graficky je postup použití množin znázorněn na obr. 8.1.



Obrázek 8.1: Filozofie použití trenovací a validační množiny

Proces zobrazený na obr. 8.1 je univerzálně použitelný. Není tedy specifický pro metody rozhodovacích stromů, popř. rozhodovacích pravidel.

Do určité míry alternativním postupem je, že pro odvození modelu použijeme některý z postupů *vzorkování* (samplerování). Základem je výběr náhodného vzorku dat z dostupné datové sady. Vzorek

musí být dostatečně velký, aby měl potenciál zachytit podstatné vlastnosti, které chceme zachytit modelem. Tento postup je velmi účinný, pro jeho realizaci je ale potřeba velké množství dat.

U jednotlivých sloupců datového sobotu je pak potřeba určit zda se jedná o sloupec vstupní nebo cílový. Cílový sloupec pak obsahuje samotné zaklasifikování.

Z pohledu nároků na data je potřeba provést přípravu tak, aby data neobsahovala:

- chybějící hodnoty
- spojitě numerické veličiny
- ... pokud možno chyby

V data miningu existují analytické metody, které jsou schopny se s chybějícími hodnotami vypořádat. Bohužel pravidlové metody nejsou jedny z nich. Z tohoto důvodu je nutno data připravit tak, aby chybějící údaje neobsahovaly. Lze zvolit různé strategie k řešení tohoto problému.

Nejjednodušší řešení může být ve smazání řádků, kde v jakémkoliv sloupci chybí hodnota. Nevýhodou tohoto přístupu je, že se tímto způsobem ztrácí z datového souboru určitá část informací. To také může znamenat, že datový soubor ztratí určitou část své vypovídající schopnosti a to jen z důvodu např. jedné chybějící položky na řádku.

Tento způsob je tak vhodný pouze v případě, kdy poměr řádků s chybějící hodnotou k celkovému počtu řádků datového souboru je zanedbatelný.

Alternativní cestou k řešení problému je chybějící hodnoty doplnit. V některých případech mohou být chybějící hodnoty odvozeny ze zbývajících hodnot - např. s použitím nějakého statistického modelu nebo modelu na bázi neuronových sítí. Ne vždy je ale možno se touto cestou ubírat. V takovém případě je možno doplnit:

- nejčastější hodnotou,
- proporcionálním rozdělením hodnot,
- náhodnou hodnotou.

Doplnění nejčastěji se vyskytující hodnotou (v daném sloupci) je založeno na myšlence, že hodnota, která se nejčastěji objevuje v existujících datech se bude nejčastěji objevovat i v datech, která chybí. Doplněním této hodnoty by tak měla vzniknout minimální chyba.

Aby výše uvedené fungovalo je ale nutné aby v datovém souboru skutečně v daném sloupci existovala ona dominantní hodnota. Zároveň musí být splněn předpoklad, že rozložení chybějících hodnot následuje stejná pravidla jako hodnoty existující. To ale nutně nemusí být splněno. Lze uvažovat o scénáři, kdy absence hodnot souvisí např. s citlivostí přístroje - přístroj hodnotu není schopen naměřit. V takovém případě, ale rozložení chybějících hodnot nenásleduje rozložení hodnot zbývajících - ale měla by být nahrazena hodnotou základní (od které měření neprokázalo odchylku).

Pokud v souboru neexistuje jedna dominantní hodnota, pak lze použít proporcionální rozložení existujících hodnot datového souboru. Takže pokud ve sloupci je rozložení hodnot 1 - 50 %, 2 - 30 % a 3 - 20 %. Pak ve stejném poměru je nutné doplňovat hodnoty i do těch chybějících. Opět je zde předpoklad, že chybějící hodnoty ve skutečnosti následují pravidla hodnot přítomných.

Poslední možností je doplnění náhodné hodnoty. V takovém případě akceptujeme obvykle, že nevíme, jaký charakter by mohla mít chybějící data a proto doplňujeme hodnotu náhodnou (v určitém specifikovaném intervalu). S touto strategií můžeme přijmout jisté předpoklady o chybě, kterou doplněním údajů do dat vnášíme - bude náhodná.

Bez ohledu na to jakou strategii přípravy dat zvolíme, vnášíme do dat chybu. Tato chyba může být významná a nebo také zanedbatelná. Předem ale lze jen obtížně odhadnout jaký bude výsledek. Proto je tak důležitá příprava kvalitní validační množiny, která nám umožní vliv těchto chyb na naši schopnost dosáhnout žádoucího výsledku kvantifikovat.

Pokud pro analýzu používáme R, můžeme použít některou z knihoven podporujících tzv. *imputační metody*, jako je např. knihovna Hmisc, která obsahuje podporu nejpoužívanějších metod tohoto typu. Příklad použití je zachycen níže.

Výpis 8.1: Doplnění chybějících údajů pomocí Hmisc knihovny funkcí impute

```
1 library(Hmisc)
2 data$column <- impute(data$column, fun = mean)
```

Výše uvedený postup funguje tak, že v datovém rámci nazvaném data ve sloupci nazvaném column nahradí chybějící hodnoty hodnotou průměrnou, vypočtenou z existujících hodnot v daném sloupci.

Metoda doplnění se vkládá do parametru `fun`. Proto místo průměru můžeme použít jakoukoliv jinou existující funkci, nebo si dokonce napsat funkci vlastní.

`Impute` funkce je specifická tím, že pro doplnění používá vždy pouze jeden sloupec. To nemusí být ve všech případech dostatečné. V některých případech bychom např. mohli chtít doplnit chybějící údaj na základě podobnosti s jiným úplným příkladem obsaženým v datech. Předpoklad je přitom takový, že pokud je příklad velmi podobný ve všech ostatních sloupcích, předpokládáme, že bude podobný i v tom chybějícím. K doplnění v takovém případě potřebujeme informace z více než jednoho sloupce dat. V takovém případě bychom použili některou z alternativních funkcí, popř. bychom museli vyvinout vlastní.

Pokud jsou v datovém souboru spojitě numerické veličiny je potřeba provést jejich rozdělení na intervaly. Jedná se o jednu z prvních úloh statistiky a tak způsob realizace najdete prakticky v každé učebnici statistiky, viz např. [43] (kapitola 2.2).



Adekvátnost modelu

Omezení na ordinální hodnoty se v případě rozhodovacích stromů týká primárně ručního výpočtu, tak jak je demonstrováno v těchto skriptech. Poslední podkapitole, ale demonstrujeme použití rozhodovacího stromu pro odvození sofistikovanějšího modelu a využíváme k tomu mnohem pokročilejší nástroje dostupné jako knihovna v R. Tento nástroj, ale výše uvedené omezení nemá!

Vždy jsme tedy omezení postupy a nástroji, které jsme zvolili pro řešení problému. **Omezení se tak liší případ od případu.** Úspěch modelování je vždy determinován naší schopností porozumět zpracovávaným datům a také nástrojům, které využíváme.

Vybaveni základními znalostmi, můžeme se pustit do metod samotných.

8.2 Rozhodovací stromy

Segmentují znalosti do podoby stromu. Pravidla vznikají tak, že procházíme strom od kořene k listu podle hodnot v jednotlivých sloupcích (prediktorech). Sloupce nám tedy ve stromu budou představovat uzly, hodnoty ve sloupcích pak budou představovat větvení stromu. Listový uzel pak představuje hodnotu cílového sloupce. Kombinací hodnot prediktorů tak postupně případ zatřídíme.

Sloupce pro větvení nevybíráme náhodně, ani v pořadí, v jakém se objevují v datovém souboru, ale na základě k předpokládané schopnosti vysvětlit hledaný koncept. Opřít se můžeme o statistické metriky založené na posuzování variability v datech, např. pomocí informační entropie (viz níže), gini indexu a řady dalších.

Existuje řada algoritmů, které pro segmentaci lze použít. jedním z nejjednodušších je algoritmus **Top Down Induction of Decision Trees (TDIDT)** známý pod řadou názvů - rozděl a panuj (odděl a panuj), AQ algoritmus a další.

Princip je jednoduchý:

1. jeden sloupec se zvolí jako kořen stromu
2. data se rozdělí podle hodnot zvoleného sloupce - uzel se rozvětví
3. pokud v takto vzniklých uzlech nepatří všechny data do stejné cílové třídy opakuj od bodu 1.

Prakticky to znamená, že sloupec se objeví ve stromu pouze jednou (pokud je použit tento algoritmus) a některé sloupce se nemusí objevit ve stromu vůbec.

Při volbě vhodného datového sloupce pro větvení stromu je možné se opřít např. o *informační entropii*. Tento pojem by Vám neměl být zcela neznámý - konečně v *Bezpečnostní informatice* [90] v druhém ročníku bakalářského studia.

Připomeňme si vzorec pro výpočet entropie (8.1):

$$H = - \sum_{t=1}^T (p_t \log_2 p_t) \quad (8.1)$$

Kde H je informační entropie, p_t je pravděpodobnost výskytu třídy t . Z praktického pohledu pro výpočet entropie hodnota pravděpodobnosti není známá, ale můžeme ji aproximovat pomocí relativní četnosti n_t , spočítané z předpřipravených dat v datasetu.

Dosažením do vzorce (8.1) získáme vzorec (8.2).

$$H(A_v) = - \sum_{t=1}^T \frac{n_t(A_v)}{n(A_v)} \log_2 \frac{n_t(A_v)}{n(A_v)} \quad (8.2)$$

Pravděpodobnost p_t jsme nahradili poměrem počtu případů nabývajících určité třídy $n_t(A_v)$ k celkovému počtu případů $n(A_v)$.

Entropii atributu (sloupce) pak spočteme podle vzorce (8.3):

$$H(A) = - \sum_{v \in \text{hodnoty}(A)} \frac{n(A_v)}{n} H(A_v) \quad (8.3)$$

Jelikož entropie popisuje neurčitost - hledáme takové sloupce v datech, jejichž hodnota entropie $H(A)$ je malá. Pro větvení tak vybíráme sloupec s nejnižší vypočtenou hodnotou entropie.

Zkusme provést orientační výpočet jednoduchého příkladu: *Naším úkolem bude vytvořit sadu pravidel pro klasifikaci klientů banky z pohledu bonity vzhledem k případnému schválení úvěru. Pro navržení sady pravidel máme k dispozici údaje o klientech a rozhodnutí o úvěru, která byla provedena v minulosti. Tyto údaje jsou zobrazeny v tabulce 8.1.*

Tabulka 8.1: Přidělování úvěrů banky (převzato z [20])

klient	příjem	konto	pohlaví	nezam.	úvěr
k1	vysoký	vysoké	žena	ne	ano
k2	vysoký	vysoké	muž	ne	ano
k3	nízký	nízké	muž	ne	ne
k4	nízký	vysoké	žena	ano	ano
k5	nízký	vysoké	muž	ano	ano
k6	nízký	nízké	žena	ano	ne
k7	vysoký	nízké	muž	ne	ano
k8	vysoký	nízké	žena	ano	ano
k9	nízký	střední	muž	ano	ne
k10	vysoký	střední	žena	ne	ano
k11	nízký	střední	žena	ano	ne
k12	nízký	střední	muž	ne	ano

V datech je cílový sloupec *úvěr* oddělen o zbytku souboru. Sloupec klient tvoří identifikátor každého řádku datového souboru a proto nám nebude tento sloupec vstupovat do výpočtu.

Pro spočtení entropie budeme muset nejprve spočítat, jak jednotlivé hodnoty sloupců vedou na hodnotu cílového sloupce. Pro příjem nám tak vznikne čtyřpolní tabulka (viz tabulka 8.2), jelikož příjem nabývá pouze dvou hodnot (vysoký/nízký) a úvěr nabývá také pouze dvou hodnot (ano/ne).

Tabulka 8.2: Příjem vs úvěr

	úvěr ano	úvěr ne
příjem vysoký	5	0
příjem nízký	3	4

Výpočet entropie sloupce *příjem* pak proběhne následovně podle vzorců 8.2 a (8.3):

$$H(\text{prijem}) = \frac{5}{12} H(\text{prijem}(\text{vysoky})) + \frac{7}{12} H(\text{prijem}(\text{nizky}))$$

$$H(\text{prijem}(\text{vysoky})) = -\frac{5}{5} \log_2 \frac{5}{5} - \frac{0}{5} \log_2 \frac{0}{5} = 0 + 0 = 0$$

$$H(\text{prijem}(\text{nizky})) = -\frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} = 0,9852$$

$$H(\text{prijem}) = \frac{5}{12} 0 + \frac{7}{12} 0,985 = 0,5747$$

Pro ostatní sloupce zápis omezím už pouze na výpočet jeho entropie. (Pokud Vám ale zcela není jasný mechanismus výpočtu - rozepište jej na kus papíru. Rovnice níže Vám pak mohou posloužit jako zpětná kontrola.)

$$H(\text{konto}) = \frac{4}{12} 0 + \frac{4}{12} 1 + \frac{4}{12} 1 = 0,6667$$

$$H(\text{pohlavi}) = \frac{6}{12} 0,9183 + \frac{6}{12} 0,9183 = 0,9183$$

$$H(\text{nezam}) = \frac{6}{12} 0,65 + \frac{6}{12} 1 = 0,825$$

Pokud porovnáme hodnotu entropie všech vstupních sloupců, vyplne z toho jasně sloupec *příjem* jako nejvhodnější k větvení. Toto větvení nám datový soubor rozdělí na dvě části. Pět řádků, kde je příjem vysoký konzistentně vede na přidělení úvěru. Na konci této větve tak již bude koncový uzel. Zbývajících sedm případů je ale nutno dále zkoumat.

Pro další krok proto z datového souboru vyřadíme již konzistentně zatřízené řádky (k1, k2, k7, k8, k10) - upravenou tabulku 8.3 použijeme pro další krok analýzy. Sloupec příjem již také nevstoupí do výpočtu.

Tabulka 8.3: Přidělování úvěrů - 2. větvení

klient	příjem	konto	pohlaví	nezam.	úvěr
k3	nízký	nízké	muž	ne	ne
k4	nízký	vysoké	žena	ano	ano
k5	nízký	vysoké	muž	ano	ano
k6	nízký	nízké	žena	ano	ne
k9	nízký	střední	muž	ano	ne
k11	nízký	střední	žena	ano	ne
k12	nízký	střední	muž	ne	ano

Vypočteme entropie:

$$H(\text{konto}) = \frac{2}{7} 0 + \frac{3}{7} 0,9183 + \frac{2}{7} 0 = 0,3936$$

$$H(\text{pohlavi}) = \frac{4}{7} 1 + \frac{3}{7} 0,9183 = 0,965$$

$$H(\text{nezam}) = \frac{2}{7} 0 + \frac{5}{7} 0,971 = 0,9793$$

Z výpočtu je patrné, že sloupec *konto* je nejvhodnější pro další větvení. V tomto případě konto vysoké vede konzistentně na přidělení úvěru, zatímco konto nízké vede konzistentně na jeho zamítnutí. V těchto případech se tedy bude jednat o listové uzly. Konto střední ve dvou případech vede na přidělení a v jednom případě na zamítnutí úvěru.

Proto opět vyřadíme už zatříděné řádky (viz tabulka 8.4) a pokračujeme dále ve výpočtu.

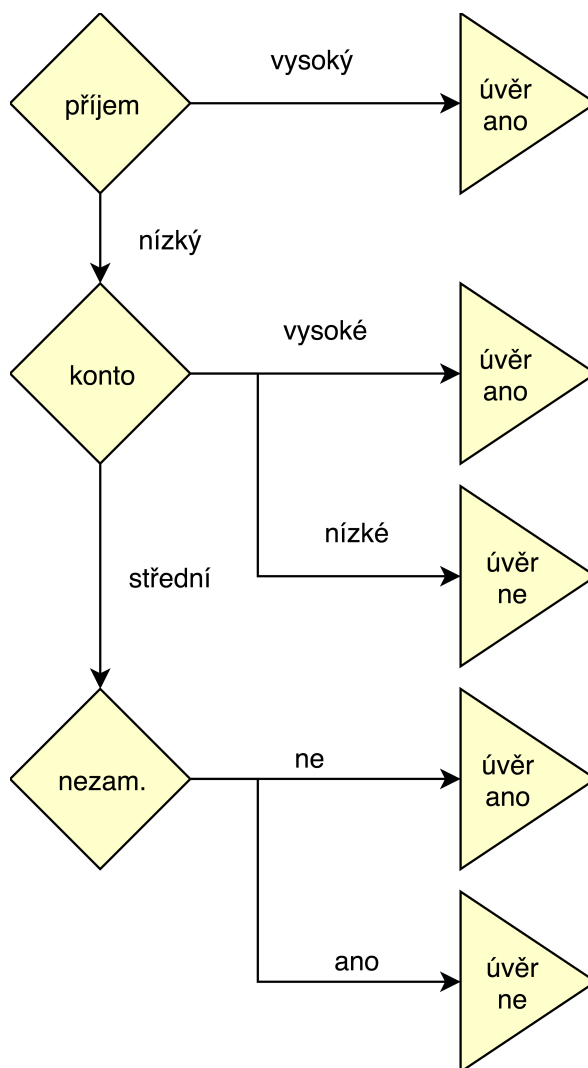
$$H(\text{pohlavi}) = \frac{2}{3} 1 + \frac{1}{3} 0 = 0,6667$$

$$H(\text{nezamestnany}) = \frac{2}{3} 0 + \frac{1}{3} 0 = 0$$

Tabulka 8.4: Přidělování úvěrů - 3. větvení

klient	příjem	konto	pohlaví	nezam.	úvěr
k9	nízký	střední	muž	ano	ne
k11	nízký	střední	žena	ano	ne
k12	nízký	střední	muž	ne	ano

Výsledek je zcela jasný - pro větvení použijeme sloupec nezaměstnaný a větvení nám umožňuje konzistentně zaklasifikovat všechny zbývající případy. Výsledný strom by mohl vypadat jako na obr. 8.2.



Obrázek 8.2: Klienti banky - ručně vytvořený rozhodovací strom

Do podoby pravidel bychom mohli převést následovně:

Výpis 8.2: Metoda rozděl a panuj - strom klientů banky

- 1 IF příjem(vysoky) THEN uver(ano)
- 2 IF příjem(nizky) && konto(vysoke) THEN uver(ano)
- 3 IF příjem(nizky) && konto(nizke) THEN uver(ne)
- 4 IF příjem(nizky) && konto(stredni) && nezam(ne) THEN uver(ano)
- 5 IF příjem(nizky) && konto(stredni) && nezam(ano) THEN uver(ne)

Každé pravidlo je samostatně funkční. Vyhodnocování probíhá postupně od prvního pravidla k

poslednímu. Mělo by platit, že se kladně vyhodnotí pouze jedna podmínka (závěr pravidla se vykoná pouze jednou). Vyhodnocena však budou všechna pravidla, což z pohledu požadovaného výpočetního výkonu není efektivní.

Alternativně by bylo možné zorganizovat do podoby *rozhodovacího seznamu*. V takovém případě se dají podmínky výše přepsat do následující podoby:

Výpis 8.3: Metoda rozděl a panuj - strom klientů banky; převod do podoby rozhodovacího seznamu

```

1 IF příjem(vysoky) THEN
2   uver(ano)
3 ELSE IF konto(vysoke) THEN
4   uver(ano)
5 ELSE IF konto(nizke) THEN
6   uver(ne)
7 ELSE IF nezam(ne) THEN
8   uver(ano)
9 ELSE uver(ne)
```

Z pohledu vyhodnocování je rozhodovací seznam jednodušší, protože jednotlivé části podmínky se vzájemně vylučují. To výrazně omezuje počet vyhodnocení, které je potřeba provést ke správnému zaklasifikování hodnoceného příkladu. Všimněte si také, že maximální počet vyhodnocení, které je nutno provést je 4. Pokud to srovnáme se sadou pravidel - je nutno provést vždy 12 vyhodnocení.

Při vyhodnocování velkého množství případů tak může být dosaženo poměrně značné časové úspory použitím rozhodovacího seznamu.



Efektivita v dataminingu

Rozhodovací seznam, který jsme představili výše uspořádá trochu výpočetního výkonu při každém použití, což při velkém množství opakování vede k signifikantním úsporám. Tento přístup lze zobecnit pro všechny metody používané v dataminingu. Vždy hledáme jakékoliv optimalizace, které vedou k zefektivnění a to i malému.

V dataminingu zpracováváme opravdu velké objemy dat, proto i drobná úspora při zpracování jednotlivého řádku dat může mít velké agregované dopady pro celou datovou sadu. V některých případech může dokonce taková úspora rozhodnout o tom, zda dataminingový projekt bude ve finále úspěšný a organizace z něj tedy bude realizovat očekávaný přínos.

Čas lze spořít také na straně odvození stromu. Pro nalezení nejvhodnějšího sloupce pro větvení stromu je možno použít místo informační entropie třeba *ginni index*. Tento index lze chápat jako zjednodušení informační entropie viz vzorec (8.4). Index nám umožňuje rozlišit mezi sloupci z pohledu jejich příspěvku k „řešení entropie“ - umožní seřadit sloupce z pohledu vhodnosti k větvení stromu, ale zároveň nám neumožňuje kvantifikovat rozdíly mezi nimi. Pro kvantifikaci rozdílů mezi sloupci v tomto případě už ve výpočtu není obsaženo dost informací.

$$Ginni = 1 - \sum_{t=1}^T p_t^2 \quad (8.4)$$

Místo pravděpodobnosti použijeme opět relativní četnosti (8.5) a ginni index celého sloupce spočteme dle vzorce (8.6).

$$Ginni(A_v) = 1 - \sum_{t=1}^T \left(\frac{n_t(A_v)}{n(A_v)} \right)^2 \quad (8.5)$$

$$Ginni(A) = \sum_{v \in \text{hodnota}(A)} \frac{n(A_v)}{n} Ginni(A_v) \quad (8.6)$$

I pokud vše maximálně zjednodušíme je jasné, že ruční výpočet není nikterak příjemný a jeho složitost výrazně roste s počtem vyhodnocovaných záznamů především ve smyslu počtu sloupců a hodnot v nich. Naší snahou by proto mělo být v maximální možné míře využít analytické nástroje, které výpočet udělají za nás.

Pro následující realizaci příkladu v R budeme potřebovat vstupní soubor CSV obsahující tabulku 8.1. Tabulku můžete připravit např. v tabulkovém procesoru MS Excel a vyexportujte jej do formátu CSV.

K odvození stromu použijeme toolkit „Tree“ [67]. Před prvním použitím je tento toolkit potřeba instalovat z CRAN, viz první řádek výpisu níže.

Instalaci knihovny je lepší provést v konzoli R. Ve výpisu (řádek) jedna sice máme instalaci společně s použitím knihovny, ale je to pouze z prostorových důvodů. Instalaci knihovny budeme dělat pouze jednou, zatímco analytický skript nebo jeho části spouštíme zpravidla opakovaně.

Instalovaných rozšíření pro R je obvykle větší množství (podle toho, co od R očekáváme). Z tohoto důvodu neškodí jednou za čas instalovaná rozšíření aktualizovat. Pokud používáte RStudio je aktualizace jednoduchá - kliknete na menu *Tools* → *Check for package updates*.

RStudio pak instalované balíky ukáže v tabulce a umožní je aktualizovat po jednom nebo hromadně.

Výpis 8.4: Binární strom přidělování úvěrů klientům banky

```
1 install.packages("tree")
2 library("tree")
3 klienti <- read.csv("klienti_banky.csv", sep = ";", header = T,
4   encoding = "UTF-8", stringsAsFactors = T)
5 #následující nastavení je nutné pouze z důvodu velmi malého počtu měření
6 #v trénovací množině
7 tree_nastaveni <- tree.control(12, mincut = 1, minsize = 2, mindev = 0.00000001)
8 model_tree <- tree(uver ~ ., data = klienti, control = tree_nastaveni)
9 summary(model_tree)
10 plot(model_tree) #vykreslí strom pomocí grafu
11 text(model_tree, pretty = 0, cex = 0.6) #textové popisky ke grafu
```

Poměrně velká část kódu by pro Vás již měla být srozumitelná, zbytek probereme. Pro použití toolkitu je potřeba nainstalovaný balík připojit pomocí funkce *library* s názvem toolkitu jako parametrem. Funkcí *read.csv* si budete muset přizpůsobit podle toho, kde přesně se CSV soubor nachází a jaké jsou jeho parametry. Můj CSV soubor používá např. jako oddělovač sloupců znak středníku.

Funkci *tree.control* je nutné použít pouze proto, že náš příklad je ... školní. U reálných příkladů je obvykle možné tato nastavení vynechat. jde o to, že reálný strom obvykle pro svou konstrukci vyžaduje, aby každý listový uzel byl pokryt určitým minimálním počtem řádků v trénovací množině. V našem příkladu však tato minimální hodnota není splněna a tak bez přenastavení by se nám neodvodil strom správně.

Proto tento řádek prostě přepište a příliš se jím netrapte :-).

Samotný strom je odvozen pomocí funkce *tree*. V našem případě má funkce tři parametry:

1. model
2. data, ze kterých bude model sestaven
3. control - nastavení stromu (u praktických příkladů obvykle není potřeba uvádět)

V modelu máme odděleny jednotlivé role pomocí znaku `~`. Začínáme specifikací proměnné obsahující cílovou proměnnou. Za `~` pak následují vstupní sloupce analýzy oddělené znakem `+`. Plus v tomto případě není označením sčítání, ale indikace sloupců (prediktorů), které se mají použít při odvození modelu. Vzhledem k tomu, že používáme všechny sloupce mimo cílový, můžeme použít zástupný znak `„.“`, který označuje všechny sloupce. Proto `„formula = uver ~ .“` znamená totéž, co `„formula = uver příjem + konto + pohlaví + nezam“`.

Logicky další parametry odkazují v případě *data* na tabulku s daty a *control* na proměnnou s výsledkem funkce *tree.control* - pokud je strom přenastaven.

Funkce *summary* je jednou z nejzajímavějších funkcí, kterou nám dává R k dispozici. Tato funkce poskytuje shrnutí k tomu, co jí předáme jako parametr. V úvodu do R jsme ji použili pro rychlý popis vlastností datového souboru. Pokud ale této funkci předáme natrénovaný model zobrazí jeho vlastnosti - v našem případě:

Výpis 8.5: Binární strom přidělování úvěrů klientům banky - metriky a výsledky

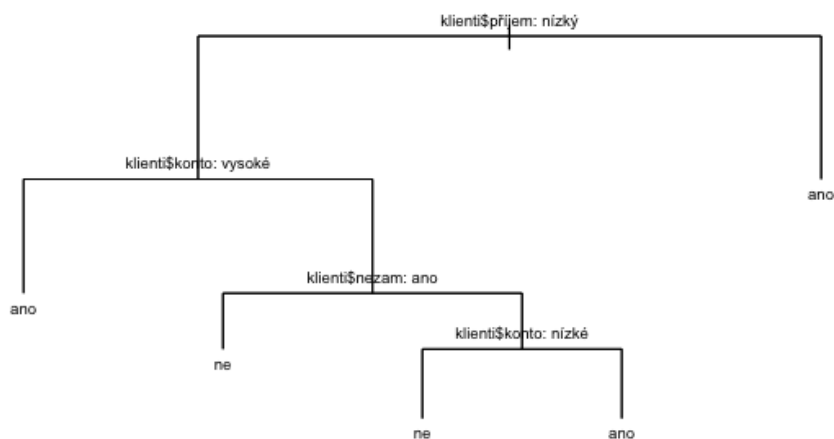
```

1 Classification tree:
2 tree(formula = uver ~ ., data = klienti, control = tree_nastaveni)
3 Variables actually used in tree construction:
4 [1] "prijem" "konto" "nezam"
5 Number of terminal nodes: 5
6 Residual mean deviance: 0 = 0 / 7
7 Misclassification error rate: 0 = 0 / 12

```

V našem případě bylo odvozeno pět listových uzlů a model dokonale pokryl všechny trénované příklady (residual mean variance = 0 a misclassification error rate = 2). Ze čtyř vstupních proměnných se pro vytvoření stromu použily jenom tři - stejně, jako když jsme strom tvořili ručně.

Samotnou strukturu stromu vykreslíme příkazem *plot* a funkce *text* pak do grafu přidá textové popisky grafu. Samotný graf je přiložen na obr. 8.3.

Obrázek 8.3: Klienti banky - strom vygenerovaný toolkitem *Tree*

Všimněte si, že ve srovnání s obr. 8.2 je způsob zobrazení poněkud jiný. To je dáno tím, že toolkit *Tree* používá tzv. binární stromy - tedy jiný algoritmus výpočtu než jsme použili k výpočtu ručnímu.

Interpretace je následující - větvení odpovídá určité konkrétní hodnotě sloupce. na obr. 8.3 je např. první větvení příjem: nízký. Binární strom má v každém uzlu pouze dvě větve (proto binární strom), které odpovídají pravdě a nepravdě. V naše případě tedy příjem: nízký - pravda (směr doleva) vede na další uzly stromu, ale směr doprava příjem: nízký - nepravda vede už na listový uzel úvěr - ano.

Jak realizovat případ, kdy naopak chceme konzultovat strom? Tedy na základě trénovací množiny jsme odvodili strom a nyní máme už data bez přiřazených tříd - resp. přiřazení potřebujeme realizovat na základě průchodu stromem.

Já jsem situaci zjednodušil tím, že pro klasifikaci jsem použil náhodně vybraný řádek trénovací množiny, ale bez sloupce *úvěr*. To není korektní postup jak jsme si řekli v kapitole *Příprava dat*, pro demonstraci postupu nám to ale bude stačit.

Výpis 8.6: Binární strom přidělování úvěrů klientům banky - hodnocení kvality modelu

```

1 scoreMnozina <- klienti
2 predpoved <- predict(model_tree, scoreMnozina, type="class")
3 predpoved #zobrazí, jak byly jednotlivé případy zaklasifikovány
4 table(pred = predpoved, true = scoreMnozina$uver) #konstrukce confusion matrix
5 mean(predpoved != scoreMnozina$uver) #výpočet průměrné chyby v klasifikaci

```

Předpověď výsledku je realizována funkcí *predict*. Všimněte si tzv. *confusion matice* (viz tab. 8.5). Ta umožňuje porovnat jak byly klasifikovány případy a jak měly být klasifikovány případy.

Tabulka 8.5: Binární strom přidělování úvěrů klientům banky - confusion matice

pred	true	
	ano	ne
ano	8	0
ne	0	4

Podle tabulky výše jsou předpovědi v řádcích (pred) a skutečné hodnoty pak ve sloupcích (true).

Předtím než půjdeme dále k praktičtějšímu příkladu, zastavme se na chvíli u confusion matice. Tato matice porovnává skutečné hodnoty s hodnotami odvozenými modelem. To nám umožňuje jednoduše, na jednom místě vidět, vidět, jak velká část dat byla skórována chybně.

V ideálním případě hodnoty jiné než 0 měly být pouze na diagonále matice, protože ta znamená že model trefil správnou hodnotu. V našem školním příkladu jsme tedy byli absolutně úspěšní a všechny příklady jsme zaklasifikovali správně. U reálně řešených problémů ale obvykle takové přesnosti nelze dosáhnout. Úspěšnost modelování je pak vyhodnocována na základě detekované chyby.

Confusion matice nám v takovém případě může pomoci lépe pochopit mezery v modelu např. ve smyslu nedobré schopnosti klasifikovat určité třídy. To nás může navést na řešení ve smyslu doplnění dalších případů do datové sady, které budou lépe charakterizovat danou cílovou třídu. Model ale bude potřeba následně znovu odvodit a znovu zhodnotit z pohledu kvality.

8.3 Případová studie rozhodovacího stromu - AMES dataset



Případová studie

V předchozí podkapitole jsme si ukázali, jak rozhodovací stromy vlastně fungují. S trochou štěstí by jste měli být schopni případně nějaký jednoduchý strom spočítat ať už ručně nebo s použitím výpočetní techniky a knihovny *tree* R.

Co se podívat na nějaký rozsáhlejší dataset a vyzkoušet si možnosti rozhodovacích stromů „v plné palbě“ se vším, co nám nabízejí? ... máme před sebou spousty práce, ale s potenciálně velkým přínosem.

Pro případovou studii použijeme dataset AMES, který byl zveřejněn v roce 2011 [27]. Dataset obsahuje data z prodeje 2903 domů ve městě Ames (Iowa, USA) v letech 2006 - 2010. Pro R jsou data dostupná v knihovně AmesHousing [46].

Pro jednotlivé prodeje bylo sledováno celkem 82 proměnných, které daný prodej charakterizovaly. Jelikož v této případové studii se setkáváme s datasetem poprvé, podíváme se na strukturu proměnných:

- *Order* - číslo pozorování (z hlediska modelování irelevantní)
- *PID* (Parcel identification number) - identifikační číslo parcely. Údaj je případně možno využít pro připojení dalších informací o pozemku, v této případové studii tento údaj ale také nevyužijeme
- *MS SubClass* - popisuje obydlí, které bylo prodáváno
- *MS Zoning* - identifikátor klasifikace zóny, ve které se budova nachází
- *Lot Frontage* - Linear feet of street connected to property
- *Lot Area* - plocha pozemku [in^2]
- *Street* - typ přístupu k pozemku/domu
- *Alley* - typ uličky/cesty vedoucí k pozemku/domu
- *Lot Shape* - becný tvar domu
- *Land Contour* - „plochost“ budovy (nakolik budova narušuje celkové kontury/panorama krajiny)
- *Utilities* - typ připojených infrastruktur (např. plyn, voda, elektřina, apod.)
- *Lot Config* - konfigurace pozemku
- *Land Slope* - sklon pozemku
- *Neighborhood* - sousedství - fyzické umístění na území města Ames

- *Condition 1* - blízkost k různým službám
- *Condition 2* - blízkost k různým službám, pokud je dostupných více než jedna
- *Bldg Type* - typ obydlí
- *House Style* - styl obydlí
- *Overall Qual* - hodnotí celkový použitý materiál a povrchovou úpravu domu
- *Overall Cond* - hodnotí celkové podmínky domu
- *Year Built* - rok dokončení stavby
- *Year Remod/Add* - rok, kdy došlo k poslední remodelaci domu (pro domy, které neprošly remodelací je rok stejný jako rok dokončení stavby)
- *Roof Style* - typ střechy
- *Roof Matl* - materiál použitý pro stavbu střechy
- *Exterior 1* - externí povrch domu
- *Exterior 2* - externí povrch domu (pokud byl použit více než jeden materiál)
- *Mas Vnr Type* - : typ zdiva
- *Mas Vnr Area* - zastavěná plocha [in^2]
- *Exter Qual* - hodnocení kvality materiálů použitých na exteriér domu
- *Exter Cond* - hodnocení současného stavu materiálu na exteriéru domu
- *Foundation* - typ základů
- *Bsmt Qual* - hodnocení výšky sklepa
- *Bsmt Cond* - hodnocení obecného stavu sklepa
- *Bsmt Exposure* - přístupnost sklepa, týká se vchodu nebo stěn na úrovni povrchu
- *BsmtFin Type 1* - rating „dokončené“¹ plochy sklepa
- *BsmtFin SF 1* - Typ 1 [in^2]
- *BsmtFinType 2* - rating plochy sklepa, pokud je tvořen více typy
- *BsmtFin SF 2* - Typ 2 [in^2]
- *Bsmt Unf SF* - „nedokončená“ plocha sklepa [in^2]
- *Total Bsmt SF* - celková plocha sklepa [in^2]
- *Heating* - typ vytápění
- *HeatingQC* - kvalita a celkový stav vytápění
- *Central Air* - centrální klimatizace
- *Electrical* - elektrické systémy
- *1st Flr SF* - plocha prvního podlaží [in^2]
- *2nd Flr SF* - plocha druhého podlaží [in^2]
- *Low Qual Fin SF* - nízko kvalitní „dokončení“, všechna podlaží
- *Gr Liv Area* - nadzemní obytná plocha [in^2]
- *Bsmt Full Bath* - sklep, úplná² koupelna
- *Bsmt Half Bath* - sklep, poloviční koupelna
- *Full Bath* - plná koupelna v nadzemní části domu
- *Half Bath* - poloviční koupelna v nadzemní části domu
- *Bedroom* - ložnice v nadzemní části (neobsahuje ložnice ve sklepech)
- *Kitchen* - Kuchyně v nadzemní části
- *KitchenQual* - kvalita kuchyně
- *TotRmsAbvGrd* - celkový počet místností v nadzemní části (neobsahuje koupelny)
- *Functional* - funkcionálnost domu (předpokládá se typická pokud sleva není zdůvodněná)
- *Fireplaces* - počet krbů
- *FireplaceQu* - kvalita krbu
- *Garage Type* - umístění garáže
- *Garage Yr Blt* - rok, ve kterém byla garáž dokončena
- *Garage Finish* - vnitřní úprava garáže
- *Garage Cars* - velikost garáže specifikovaná počtem aut, které je možno do ní zaparkovat
- *Garage Area* - velikost garáže [in^2]
- *Garage Qual* - kvalita garáže
- *Garage Cond* - stav garáže

¹jako dokončené plocha je vnímán takový sklep, který je dokončen podobně jako běžné součásti domu, tedy včetně zavedené elektřiny, vytápění, dodělané podlahy, vyrovnané stropy a omítnuté stěny.

²umyvadlo, sprcha, vana a záchod (za každou chybějící část 1/4 hodnocení dolů - např. bez vany a záchodu se jedná o poloviční koupelnu)

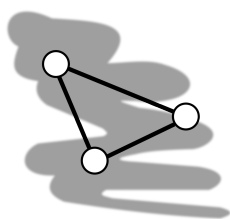
- *Paved Drive* - : zpevněná příjezdová cesta
- *Wood Deck SF* - plocha dřevěné terasy [in^2]
- *Open Porch SF* - plocha otevřené verandy [in^2]
- *Enclosed Porch* - plocha uzavřené verandy [in^2]
- *3-Ssn Porch* - plocha verandy pro tři roční období [in^2]
- *Screen Porch* - čelní plocha verandy [in^2]
- *Pool Area* - velikost bazénu [in^2]
- *Pool QC* - kvalita bazénu
- *Fence* - kvalita plotu
- *Misc Feature* - různé vlastnosti, které nejsou pokryty jinými kategoriemi
- *Misc Val*: hodnota různých vlastností [USD]
- *Mo Sold* - měsíc, ve kterém byl realizován prodej
- *Yr Sold* - rok prodeje
- *Sale Type* - typ prodeje
- *Sale Condition* - podmínky prodeje

82 sloupců a tisíce řádků představují určitou výzvu. V takovém počtu proměnných nemůžeme postupovat naivně ve snaze po jednom zkoušet, jaká kombinace sloupců povede k nejlepšímu výsledku. Potřebujeme tedy mít k dispozici sofistikovaný nástroj, který nás odstíní do značné míry od complexity celkového výpočtu.

Proměnné jsou různého typu, některé jsou kategoriální, jiné obsahují numerické hodnoty. Řada proměnných také obsahuje chybějící hodnoty. Jinými slovy dataset je zatížen problémy, které bychom při ručním výpočtu podle předchozí kapitoly nebyli schopni vypočítat. Se správnými knihovnami pro R, ale výpočet není problém, viz výpis kódu níže.

Příklad níže je inspirován příkladem publikovaným v Boehmke a Greenwel [22]. Oproti v knize zveřejněné verzi se zaměřuje pouze na jeden model zpracování datasetu (model přístupný v proměnné *ames_dt3*), na který aplikuje všechny potřebné postupy. Oproti verzi příkladu v knize jsou doplněny některé dodatečné postupy pro prezentaci výsledků.

Příkladem doplněného postupu je funkce *split.fun* sloužící k úpravě vzhledu výsledného stromu.



Hands-On Machine Learning with R

Z hlediska dalšího zkoumání parametrů může být výhodné konzultovat příklady přímo z knihy. Výhodou je, že kniha je ve fultextové podobě, včetně příkladů dostupná bezplatně on-line na adrese <https://bradleyboehmke.github.io/HOML/>

Výpis 8.7: Regresní strom pro AMES dataset

```
1 library("AmesHousing") #obsahuje dataset AMES
2 library("rpart") #knihovna pro práci s rozhodovacími stromy
3 library("rpart.plot") #knihovna pro vykreslování stromů
4 library("caret")
5 library("vip")
6 library("dplyr")
7 library("plotly")
8
9 ames_train <- make_ames()
10 ames_dt3 <- train(
11   Sale_Price ~ .,
12   data = ames_train,
13   method = "rpart",
14   trControl = trainControl(method = "cv", number = 10),
15   tuneLength = 20
16 )
17 rpart.plot(ames_dt3$finalModel, split.fun=split.fun)
18 #výpočet CP
19 x <- ames_dt3$finalModel$cp[,1]
20 y <- ames_dt3$finalModel$cp[,3]
21 fig <- plot_ly(
22   x = x,
```

```

23 y = y,
24 mode = 'lines+markers',
25 type = "scatter") %>%
26 layout(
27   title = "Velikost stromu vs presnost vypoctu",
28   xaxis = list(title = "parametr komplexity (cp)",
29     autorange="reversed"),
30   yaxis = list(title = "relativni chyba")
31 )
32 fig
33 fig <- plot_ly(
34   x = ames_dt3$results$cp,
35   y = ames_dt3$results$RMSE,
36   mode = 'lines+markers',
37   type = "scatter") %>%
38 layout(
39   title = "Stredni kvadraticka chyba stromu",
40   xaxis = list(title = "parametr komplexity (cp)"),
41   yaxis = list(title = "RMSE (stredni kvadraticka chyba - krizova validace)")
42 )
43 fig
44 uzly <- length(ames_dt3$results$RMSE):1
45 fig <- plot_ly(
46   x = uzly,
47   y = ames_dt3$results$RMSE,
48   mode = 'lines+markers',
49   type = "scatter") %>%
50 layout(
51   title = "Zavislost stredni kvadraticke chyby stromu na poctu listovych uzlu",
52   xaxis = list(title = "velikost stromu"),
53   yaxis = list(title = "RMSE (stredni kvadraticka chyba - krizova validace)")
54 )
55 fig
56 vip(ames_dt3, num_features = 26, bar = FALSE)

```

Výpočet stromu a jeho vizualizaci provedeme pomocí knihoven *rpart* [80] a *rpart.plot* [55].

Data bereme z knihovny AmesHousing. Použijeme data ve zpracované podobě přístupná pomocí funkce *make_ames()*. Strom samotný odvozujeme funkcí *rpart* ze stejnojmenné knihovny. Parametry jsou poměrně intuitivní. Syntaxe parametru *formula* může na první pohled vypadat divně, ale není. *Sale_price* je cílový atribut, odděluje cílový atribut od zbytku modelu (ze kterého má být odvozen) a „.“ je zástupný symbol pro všechny sloupce. O této možnosti jsme se zmiňovali již při výkladu předchozího příkladu.

Parametr *data* obsahuje odkaz na proměnnou obsahující data ke zpracování. Výsledný strom je možno vypsat v textové podobě vypsáním obsahu proměnné *ames_dt1*, my ale místo toho použijeme funkci *rpart.plot*, ze stejnojmenné knihovny, pro vykreslení stromu, viz obr. 8.4.

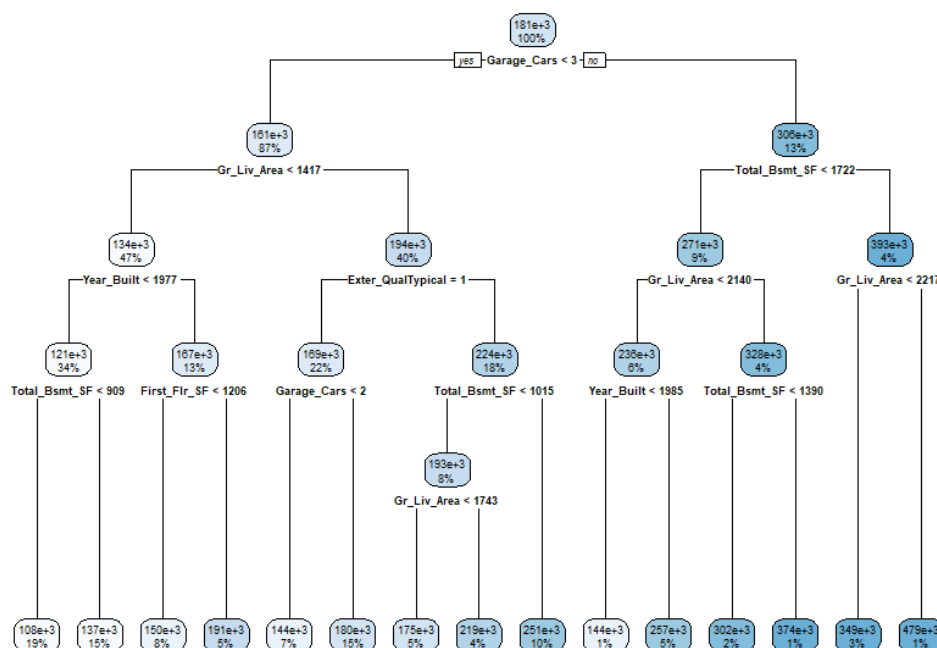
Pro regresní stromy je typické, že mohou být takřka libovolně košaté. Uvědomme si, že pro větvení může být použit kterýkoliv sloupec a jeho kterákoliv hodnota. Jak je ale vidno na obr. 8.4, je strom ve skutečnosti jednodušší. Funkce *rpart* totiž má implementovaný také mechanismus prořezávání stromu.

Tento mechanismus spočívá v automatickém porovnávání chyb stromu generovaných stromy různě nastavenými parametry, především pak různě nastaveným počtem listových uzlů. K tomuto účelu se používá 10-ti násobná křížová validace (10-fold **Cross Validation (CV)**). Princip křížové validace je poměrně jednoduchý. Z datasetu se náhodně vyberou data do trénovací a validační množiny, pro kterou se odvodí model a spočte chyba. Desetinásobná znamená, že získáme 10 takových výběrů, každý jiný.

Nastavení CV se provádí pomocí *trainControl* funkce. To je poměrně zásadní výhoda, pro kterou používáme knihovnu *caret*. Tato knihovna nám slouží jako abstrakční vrstva nad různými modely. Nyní pracujeme s rozhodovacím stromem, pokud bychom ale pracovali s lineární regresí, pouze bychom změnili metodu na *lm* a změnili hyperparametry.

Tímto postupem se snažíme vyhnout zkreslením v odvozeném modelu způsobené náhodným výběrem trénovací množiny.

Tento optimalizační princip je dobře viditelný na obr. 8.5, který vizualizuje vztah velikosti modelu/stromu měřeného počtem uzlů a jeho přesnosti. S přibývajícím počtem uzlů se sice přesnost modelu zvyšuje, ale přínos je s každým dalším uzlem menší a menší. Z hlediska optimalizace tak můžeme zvolit



Obrázek 8.4: Regresní strom pro AMES dataset

minimální zpřesnění výpočtu, které považujeme za přínosné a pokud přesáhneme tuto hranici bude přinášet každé další zvýšení komplexity stromu pouze zanedbatelné zvýšení bezpečnosti.

Při použití caret knihovny, ale obvykle preferujeme spíše použití metriky **Root-mean-square error (RMSE)**, tedy střední kvadratické chyby, viz obr. 8.6. Tento pohled je ve skutečnosti velmi podobný jako je obr. 8.5, rozdíl je v orientaci osy x. Každý bod na obou křivkách představuje listový uzel stromu. S přibývajícím počtem uzlů se zlepšuje kvalita predikce a klesá hodnota chyb.

Ještě lépe viditelný přínos většího počtu listových uzlů je viditelný pokud parametr komplexity *cp* nahradíme počtem uzlů, viz obr. 8.7. V tomto případě je vzdálenost mezi jednotlivými uzly grafu konstantní, což nám umožňuje vidět postupné zlepšování RMSE až do úrovně okolo 40 000 pro 15 listových uzlů. Následně ale RMSE znovu roste!

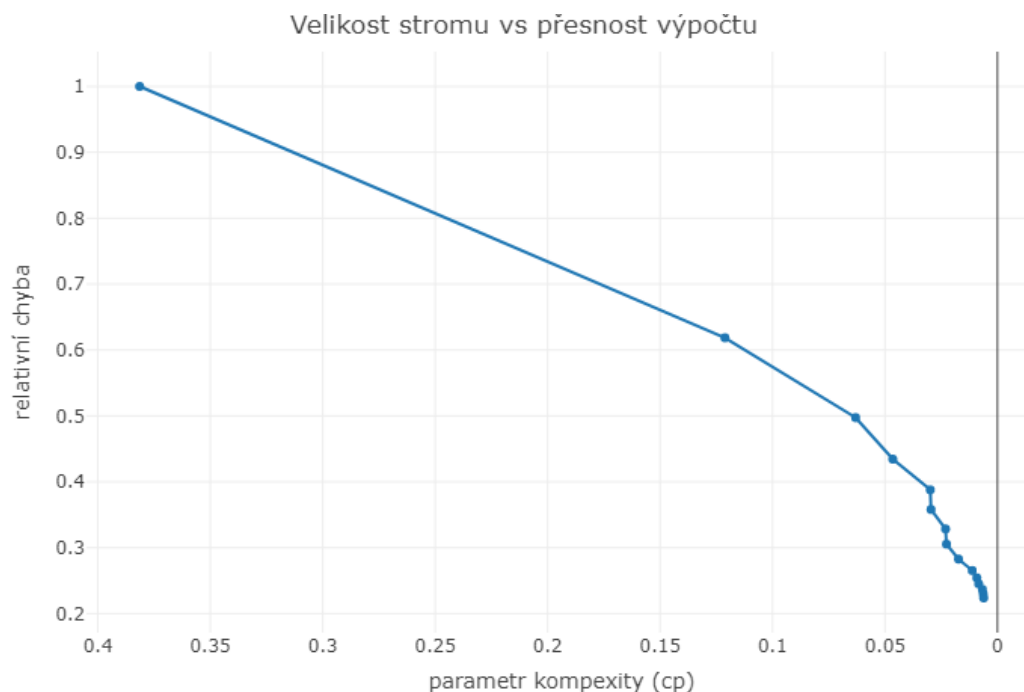
Pokud zkontrolujeme obr. 8.4 zjistíme, že pro konstrukci stromu byla zvolena právě optimální hodnota 15-ti listových uzlů. Celkový počet listových uzlů jsme v rámci funkce *train* nastavili na 20 pomocí parametru *tuneLength*. Počet listových uzlů je právě hyperparametrem, který ovlivňujeme v případě rozhodovacího stromu nejčastěji.

Hodnotu 20 je možno určit odhadem tak, že pokusně spočteme strom přímo pomocí funkce *rpart* a podíváme se kolik listových uzlů navrhne. V našem případě navrhne 11 listových uzlů. Lze očekávat, že zvýšením počtu uzlů může být dosaženo lepších výsledků. Parametr *tuneLength* by proto měl očividně být > 11 , zároveň je ale možno předpokládat, že z hlediska výpočtu budeme přibližně okolo optima a tak nemá smysl, aby hodnota parametru byla příliš vysoká (např. 100).

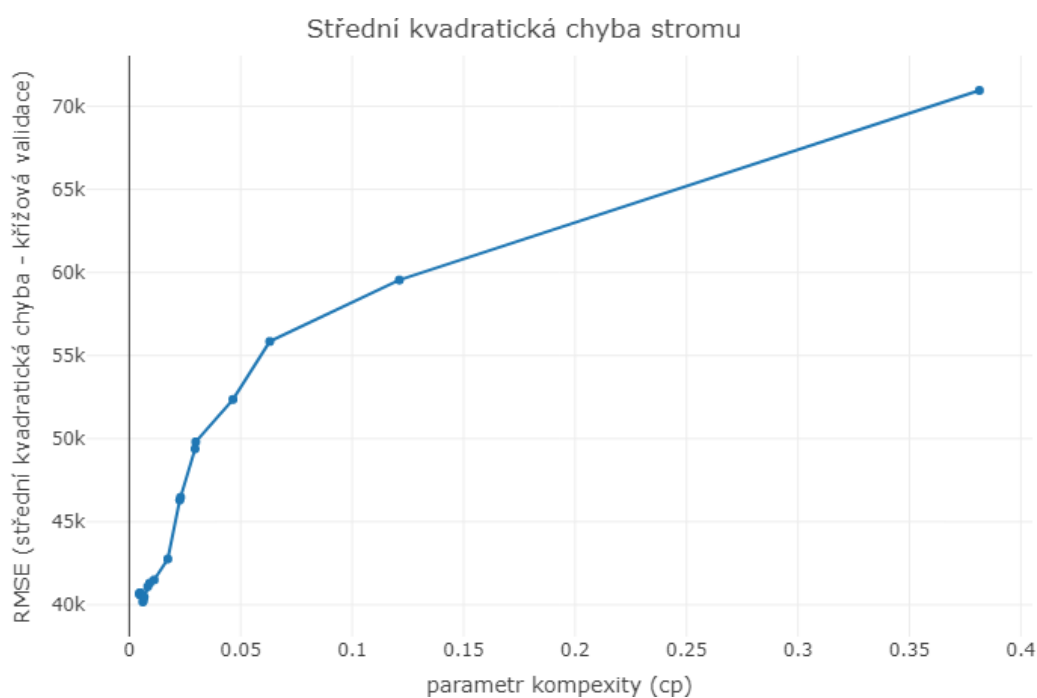
Platí, že čím vyšší hodnota parametru, tím více stromů algoritmus spočítá. S použitím 10-ti násobné CV a *tuneLength* = 20 je tak ve skutečnosti vypočteno 200 stromů! Vzhledem k rychlosti v současnosti používané výpočetní techniky i přes to výpočet proběhl ve skoro reálném čase. Pro rozsáhlejší datové sady je potřeba brát ná výše uvedené ohled.

Poslední informací, kterou můžeme z modelu dostat je hodnocení důležitosti jednotlivých proměnných. Pro hodnocení můžeme využít knihovnu *vip* a stejnojmennou funkci z ní. Tato funkce vezme model a vypíše specifikovaný počet proměnných seřazených od nejvýznamnější po nejméně významnou, viz obr. 8.8.

Při přípravě vizualizace výsledku je vždy otázkou, kolik proměnných se má v grafu objevit. Jelikož má dataset 82 sloupců (z toho jeden cílový) je očividné, že maximální hodnota bude rovna 81. Ve většině případů je pro odvození modelu používána pouze relativně malá část sloupců. Pokusná po-



Obrázek 8.5: Vztah velikosti stromu a přesnosti výpočtu pro AMES dataset

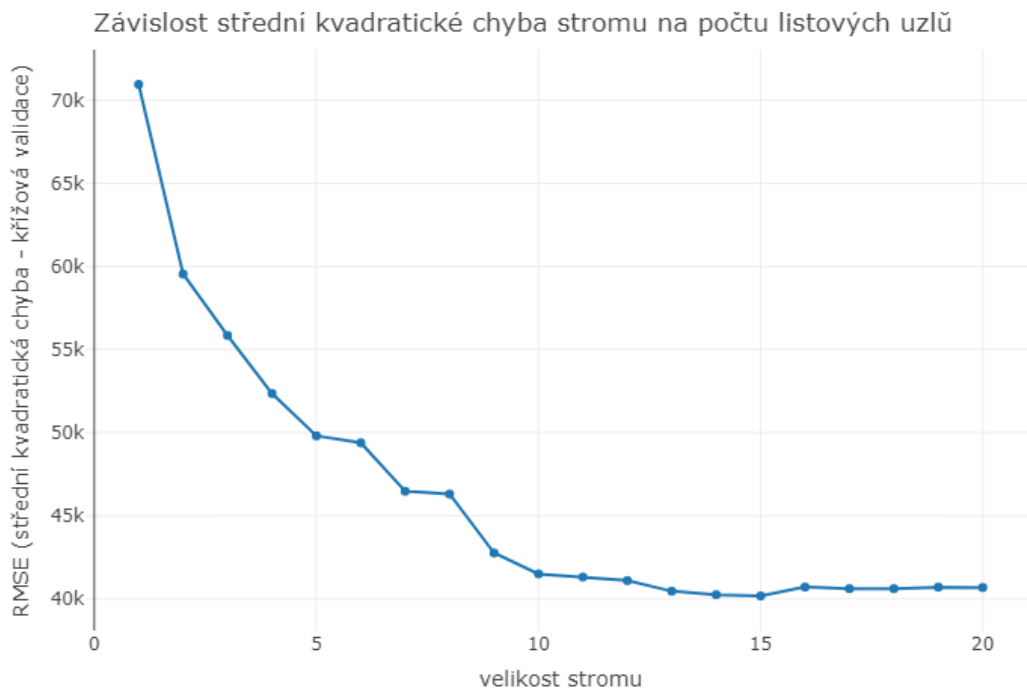


Obrázek 8.6: Komplexita stromu a jeho přesnost pro AMES dataset měřeno pomocí RMSE

částeční hodnota může proto být např. 40 odpovídající přibližně polovině sloupců. Podle výsledku lze upravovat počet zobrazovaných sloupců směrem nahoru nebo dolů podle potřeby.

V našem případě má jistou vysvětlovací schopnost pouze 25 sloupců. Na obr. 8.8 je znázorněno 26 sloupců, aby hodnota posledního sloupce datasetu byla 0. Tímto způsobem je na obr. jasně demonstrováno, že v datasetu nejsou další významné sloupce.

Ke knihovně caret je potřeba doplnit ještě jednu věc - používáme ji výhradně pro vývoj a ladění modelů. Účelem je, abychom našli optimální nastavení hyperparametrů, které pak použijeme pro natrénování finální verze modelu.



Obrázek 8.7: Vztah velikosti stromu a přesnosti výpočtu pro AMES dataset měřeno pomocí RMSE



Parametry modelu vs hyperparametry modelu

V textu jsme se setkali s novými pojmy *parametry* a *hyperparametry* modelu. Co to vlastně je? Formálně lze parametr definovat jako: *proměnná nebo hodnota odvozená z datasetu*. Dobrým příkladem parametru v případě lineární regrese je třeba parametr sklonu přímky. Oproti tomu hyperparametr je: *proměnná nebo hodnota, kterou není možné odvodit z dat*. Parametry i hyperparametry ovlivňují chování modelu a způsob jakým predikuje hodnoty.

Příkladem hyperparametru pro rozhodovací stromy je minimální počet měření, které musí obsahovat listový uzel. Většina modelů (včetně rozhodovacích stromů) má větší množství hyperparametrů.

Jelikož hyperparametry není možno odvodit z dat, musíme prostor hodnot jednotlivých hyperparametrů manuálně prohledávat. To je také naší zásadní motivací pro použití knihovny *caret*, která s „laděním“ hyperparametrů výrazně pomáhá.

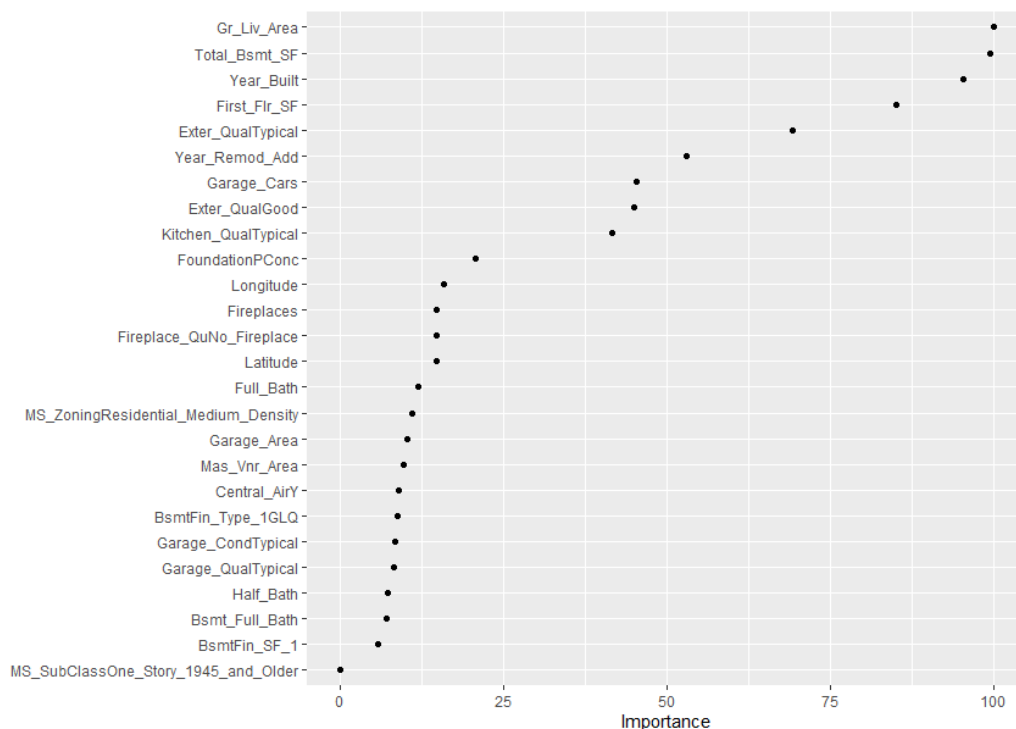
V praxi by se proto postupovalo tak, že pomocí *caret* projdeme prostor možných nastavení jednotlivých hyperparametrů modelu. Pro každou kombinaci hodnot těchto hyperparametrů spočteme model a zhodnotíme jeho kvalitu validací. *Caret* nám následně vrátí takovou kombinaci hyperparametrů, která vedla v rámci validace k nejlepšímu výsledku. Toto nastavení aplikujeme na zvolenou metodu, abychom získali finální verzi modelu.

V našem případě proto hyperparametry odvodíme použitím *caret*, která volá výpočetní funkci *rpart* pro modelování rozhodovacích stromů. Finální model získáme už přímým voláním funkce *rpart* s odvozenými nastaveními hyperparametrů.

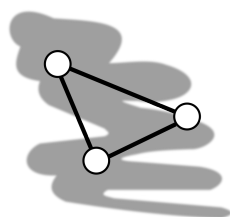
Důvodem pro tento postup je opět efektivita. Při nasazení finálního modelu nechceme už platit cenu za schopnost ladit hyperparametry. Tuto část ladění máme již za sebou.

8.4 Rozhodovací pravidla

K rozhodovacím pravidlům jsme dospěli již v předchozí kapitole, jako jedné z možností jak reprezentovat výsledek modelu rozhodovacího stromu. Co ale dělat v případě, kdy chceme odvodit pravidla aniž bychom odvozovali nejprve strom?



Obrázek 8.8: Hodnocení významnosti parametrů funkcí vip pro AMES data - model rozhodovacího stromu



Případová studie

Dostali jsme se na konec případové studie... byla obsáhlá. Podobným způsobem budeme postupovat také pro další metody, které jsou v tomto předmětu probírány. Použijeme stejný příklad, aby jsme si v dalších kapitolách zjednodušili práci. V dalších kapitolách se tak zaměříme už pouze na interpretaci odlišností souvisejících s jinými typy modelu, které budeme používat.

Budeme ale stále používat stejný rámec hodnocení tzn.:

- výpočet modelu pomocí funkce train
- vizualizace modelu (pokud lze)
- interpretace RMSE
- hodnocení významnosti parametrů modelu (vip)

Pro demonstraci postupu použijeme opět příklad klientů banky, viz tabulka 8.1. Protože tento příklad budeme dělat pouze ručně, zvolíme nejjednodušší způsob odvození těchto pravidel - budeme postupovat v datech sekvenčně.

Zvolíme první řádek, který vede na hledaný koncept - v našem případě na přidělení úvěru. Můžeme si zjednodušit situaci tak, že budeme předpokládat, že případy které konzistentně nevedou na úvěr(ano) povedou na úvěr(ne). To nám umožní zabývat se pouze podmnožinou pravidel.

Prvním řádkem, který splňuje naše požadavky je shodou okolností řádek první (k1). Z něj můžeme formulovat pravidlo: IF příjem(vysoký) && konto(vysoké) && pohlaví(žena) && nezaměstnaný(ne) THEN úvěr(ano).

Zkontrolujeme že dané pravidlo vede konzistentně na úvěr(ano), tedy že neexistuje jiný řádek obsahující stejné výchozí hodnoty, který by vedl na výsledek úvěr(ne). V našem případě tomu tak není.

Následně se pokusíme pravidlo zjednodušit. Postupujeme opět sekvenčně. Řádek k2 se liší od k1 v položce pohlaví. Z toho lze odvodit, že tento sloupec v tomto pravidle není určující a vyřadíme jej. Pravidlo se zjednoduší a zároveň nám pokryje místo jednoho řádku, dva.

Pohledem na k4 můžeme pravidlo ještě více zjednodušit do podoby IF konto(vysoké) THEN úvěr(ano). Tímto způsobem získáváme pravidlo, které umožňuje spolehlivě zaklasifikovat řádky k1,

k2, k4 a k5. Tyto řádky z dalšího odvozování můžeme vyřadit.

Postupujeme dále - dalším nepokrytým případem úvěr(ano) je k7. Odvodíme pravidlo IF Příjem(vysoký) && konto(nízké) && pohlaví(muž) && nezaměstnaný(ne) THEN úvěr(ano). Při porovnání s řádky k8 a k10 můžeme zjednodušit toto pravidlo na IF příjem(vysoký) THEN úvěr(ano). Pokryty tak máme řádky k7, k8 a k10, které s další analýzy vyřadíme.

Zbývá k12. Ten je poslední srovnáním s existujícími pravidly můžeme toto pravidlo zjednodušit do podoby: IF konto(střední) && nezaměstnaný(ne) THEN úvěr(ano).

Vzniká nám sada pravidel

Výpis 8.8: Rozhodovací pravidla

```
1 IF konto(vysoke) THEN uver(ano)
2 IF prijem(vysoky) THEN uver(ano)
3 IF konto(stredni) && nezamestnany(ne) THEN uver(ano)
```

Srovnajte tuto sadu pravidel se sadou, kterou jsme odvodili v předchozí podkapitole. Všimněte si, že obě sady jsou rozdílné, přesto obě sady spolehlivě pokrývají celou trénovací množinu.

Tento stav je z hlediska data miningu poměrně častý. Vzhledem ke složitosti problému používané algoritmy totiž nehledají **optimální řešení** (ve smyslu nejlepší řešení), ale **suboptimální** (ve smyslu nejlepší reálně dosažitelné řešení, které stále ještě splňuje podmínky zadání).

Z praktických důvodů ale doporučujeme používat spíše rozhodovací stromy. Jsou dobře podporovány analytickými nástroji, což je přesně to, co potřebujeme pro řešení reálných problémů v rámci dataminingu.



Koncepty pohyblivé v čase

Dobrá funkce rozhodovacích stromů je závislá na schopnosti trénovací množiny reprezentovat modelovaný problém. V okamžiku, kdy dojde významnějšímu posunu, vede to automaticky k vyšší chybovosti modelu. Tedy přesnost modelu v čase obvykle klesá. Někdy označujeme tuto skutečnost jako *drift modelu*.

Existují metody, jak se rozhodovací stromy mohou s tímto problémem vypořádat - implementací *mechanismu zapomínání*. Ten předpokládá, že do trénovací množiny budeme v pravidelných intervalech přidávat nové případy a „zapomínat“ ty staré. Pak přepočítáme strom a výsledek by měl být lepší.

Výše uvedené ale od nás vyžaduje, abychom dlouhodobě sledovali model z pohledu jeho schopnosti dlouhodobě plnit naše požadavky a určit čas, kdy je takový zásah potřeba realizovat.

Kontrola kvality modelu v průběhu jeho života je něco, co lze v obecné rovině doporučit všem modelům řešících klasifikační problémy. Drift modelu není něco, co by se omezovalo pouze na rozhodovací stromy.

8.5 Asociační pravidla

Oproti rozhodovacím stromům a pravidlům nesloučí pravidla asociační ke klasifikaci případů. Umožňují nám ale lépe poznat data a vazby v nich. Tento druh algoritmů nám tak může posloužit pro deskripci nebo hledání nugetů.

Asociační pravidla byla vyvinuta v 80. letech minulého století Agrawalem a kol. [19] pro analýzu nákupního košíku. V tomto případě je potřeba analýzu nákupního košíku brát doslova. Výzkum se tedy zaměřil na hledání vazeb mezi jednotlivými položkami, které zákazníci nakupují, např. v supermarketech.

Můžeme to vnímat trochu komicky jako v boxu *kečup vs hořčice*, ale aplikace jsou reálné. Výhoda asociačních pravidel je, že je možné zkoumat a testovat libovolné vztahy mezi dvojicemi položek, ale nemusíme se na ně vázat - testovat lze trojice atd.



Kečup nebo hořčice

Dobrým příkladem vazby mezi zbožím může být třeba párek a hořčice. Neznamená to, že lidé kupují vždy tyto položky dohromady, spíše že se tato kombinace v nákupních koších objevuje častěji (je statisticky signifikantní).

Ti z Vás, kterým v nákupním košíku chybí ještě kečup, tak vězte, že se chuťově tragicky pletete :-).

Fáze odvozování tak do určité míry připomíná confusion matici, se kterou jsme pracovali výše. V kontextu příkladu párek - hořčice to můžeme shrnout tabulkou 8.6. Zkoumáme reakci zákazníka na koupi (nebo naopak nekoupení) párku - koupil hořčici.

Z tohoto pohledu párek bude v předpokladové části pravidla (antecedent), hořčice pak v závěrové části (sukcedent).

Tabulka 8.6: Párek vs hořčice

	hořčice suc	!hořčice !suc	Σ
párek (ant)	a	b	r
!párek (!ant)	c	d	s
Σ	k	l	n

V tabulce 8.6 tak a znamená kolikrát byl koupen párek a hořčice, b kolikrát byl koupen párek, ale ne hořčice... atd.

Ke dvojici pak lze vypočítat řadu charakteristik, např. podporu (8.7) nebo spolehlivost (8.8).

$$P(\text{ant} \wedge \text{suc}) = \frac{a}{a + b + c + d} = \frac{a}{n} \quad (8.7)$$

$$P(\text{suc}|\text{ant}) = \frac{a}{a + b} = \frac{a}{k} \quad (8.8)$$

Podpora tedy není nic jiného než poměr případů splňujících předpoklad i závěr pravidla k celkovému počtu případů. Spolehlivost pak představuje podmíněnou pravděpodobnost závěru.

Pro praktickou ukázkou je potřeba ještě „předpřipravit“ několik charakteristik. Absolutní počet objektů splňujících předpoklad spočteme dle (8.9).

$$P(\text{ant}) = \frac{a + b}{a + b + c + d} = \frac{r}{n} \quad (8.9)$$

Analogicky absolutní počet objektů splňujících závěr je možno spočítat dle (8.10).

$$P(\text{suc}) = \frac{a + c}{a + b + c + d} = \frac{k}{n} \quad (8.10)$$

S těmito vzorci můžeme vypočítat *lift*, viz (8.11).

$$\text{lift}(\text{ant} \implies \text{suc}) = \frac{P(\text{suc}|\text{ant})}{P(\text{suc})} = \frac{P(\text{ant} \wedge \text{suc})}{P(\text{ant})P(\text{suc})} \quad (8.11)$$

Lift je tedy definován jako poměr případů podporující pravidlo k součinu apriori pravděpodobností, že jednotlivé části pravidla nastanou bez ohledu na cokoliv jiného. Přeloženo do češtiny předchozí věta znamená, že lift nám umožňuje vypočítat nakolik je výskyt jedné nebo více položek (podle toho, jak

je asociace formulována, levá strana pravidla, LHS) spojen s výskytem jiné položky (pravá strana pravidla, RHS) ve srovnání s tím, co bychom očekávali, kdyby položky byly skutečně nezávislé.

Hodnota lift může vyjít větší, menší nebo rovna 1. Hodnoty menší než 1 ukazují na negativní závislost (negativní korelace) popř. substituční efekt podle charakteru dat. Hodnoty větší než 1 ukazují na pozitivní závislost, popř. komplementární efekt.

Řekněme, že pro náš příklad nákupu párku a hořčice. Vyjde nám hodnota lift např. 1,67. Interpretovat ji můžeme tak, že je 1,67x vyšší pravděpodobnost že zákazník kupující párek koupí také hořčici než průměrný zákazník.

Základem asociačních pravidel je tak generování kombinací hodnot jednotlivých sloupců a následný výpočet vlastností takových kombinací. Počet kombinací je i pro malé datové soubory obvykle velký. Např. pro datový soubor klientů banky (viz tab. 8.1) je možno vytvořit 323 kombinací hodnot.

Proto je více méně nemožné počítat asociační pravidla ručně. Zkusme demonstrovat tvorbu asociačních pravidel na příkladu. Tentokrát nepoužijeme klienty bank - protože bychom se mohli dozvědět šokující pravdy o tom, že např. pokud je člověk nezaměstnaný má nízký příjem apod. Pro demonstraci použijeme dataset Titanic, obsahující informace o pasažérech legendárního Titanicu a informaci o tom, zda přežili nebo ne.

Dataset můžete stáhnout buď z Intenetu (jedná se o open source dataset) nebo třeba z LMS, kde je k dispozici lokální kopie datasetu. Nejprve provedeme načtení a průzkum datasetu.

Výpis 8.9: Načtení datasetu Titanic do R

```
1 load("cesta/titanic.raw.rdata")
2 str(titanic.raw)
3 summary(titanic.raw)
```

RData je vnitřní formát, ve kterém pracuje R. Tento formát umožňuje ukládat datové sady včetně datových formátů, což je obzvláště výhodné u složitějších datasetů. Vyhneme se tím kroku, kdy např. data načtená z CSV souboru je nutno při každém načtení převést do vhodného formátu.

Funkce *load* umožňuje takový soubor načíst. Obdobným způsobem lze také data ukládat, pomocí funkce *save*.

Výpis 8.10: Ukládání a načítání dat v R

```
1 save(klienti, file = "cesta/klienti.rdata")
2 #uložení více datasetů do souboru
3 save(klienti, prodejData, file = "cesta/cviceni.rdata")
4 load("cesta/cviceni.rdata") #opětovně načtení souboru
```

Funkce *str* vypíše vlastnosti datového souboru. Dataset Titanic obsahuje celkově 2201 pozorování (pasažérů + posádky lodi) ve čtyřech proměnných:

1. class - v jaké třídě byla osoba ubytována (1st, 2nd, 3rd, crew)
2. sex - pohlaví osoby (male/female)
3. age - věk osoby v okamžiku nehody (adult/child)
4. survived - zda osoba přežila (no/yes)

Vhled do struktury souboru nám pak poskytne funkce *summary*, její výstup je v tabulce 8.7.

Tabulka 8.7: Rozložení datasetu Titanic

Class (třída)	Sex (pohlaví)	Age (věk)	Survived (přežil)
1st: 325	Female: 470	Adult: 2092	No : 1490
2nd: 285	Male: 1731	Child: 109	Yes: 711
3rd: 706			
Crew: 885			

Jedná se tedy o datový soubor, který je sice na jedné straně větší, avšak na straně druhé je strukturálně jednoduchý. Samotnou analýzu a vypsání pravidel můžeme provést následovně:

Výpis 8.11: Odvození asociačních pravidel pro dataset Titanic

```

1 library(arules)
2 pravidla <- apriori(titanic.raw) #natrénování asociačních pravidel
3 pravidla #vypiše kolik pravidel bylo natrénováno
4 inspect(pravidla) #vypiše pravidla

```

Pro práci s asociačními pravidly použijeme toolkit *arules* [37]. Tento toolkit implementuje dva populární algoritmy pro odvozování asociací a to tzv. *apriori* algoritmus a *Eclat* algoritmus. Pro náš příklad jsme použili první z těchto algoritmů pomocí funkce *apriori* s datasetem jako jejím parametrem.

Před prvním použitím toolkitu nezapomeňte, že je nutné jej nainstalovat (`install.package("arules")`).

Apriori algoritmus pro náš dataset odvodí 27 pravidel, která můžeme prozkoumat pomocí funkce *inspect*. Celý výpis pravidel vypadá následovně:

	lhs	rhs	support	confidence	lift	count
[1]	{}	=> {Age=Adult}	0.9504771	0.9504771	1.0000000	2092
[2]	{Class=2nd}	=> {Age=Adult}	0.1185825	0.9157895	0.9635051	261
[3]	{Class=1st}	=> {Age=Adult}	0.1449341	0.9815385	1.0326798	319
[4]	{Sex=Female}	=> {Age=Adult}	0.1930940	0.9042553	0.9513700	425
[5]	{Class=3rd}	=> {Age=Adult}	0.2848705	0.8881020	0.9343750	627
[6]	{Survived=Yes}	=> {Age=Adult}	0.2971377	0.9198312	0.9677574	654
[7]	{Class=Crew}	=> {Sex=Male}	0.3916402	0.9740113	1.2384742	862
[8]	{Class=Crew}	=> {Age=Adult}	0.4020900	1.0000000	1.0521033	885
[9]	{Survived=No}	=> {Sex=Male}	0.6197183	0.9154362	1.1639949	1364
[10]	{Survived=No}	=> {Age=Adult}	0.6533394	0.9651007	1.0153856	1438
[11]	{Sex=Male}	=> {Age=Adult}	0.7573830	0.9630272	1.0132040	1667
[12]	{Sex=Female, Survived=Yes}	=> {Age=Adult}	0.1435711	0.9186047	0.9664669	316
[13]	{Class=3rd, Sex=Male}	=> {Survived=No}	0.1917310	0.8274510	1.2222950	422
[14]	{Class=3rd, Survived=No}	=> {Age=Adult}	0.2162653	0.9015152	0.9484870	476
[15]	{Class=3rd, Sex=Male}	=> {Age=Adult}	0.2099046	0.9058824	0.9530818	462
[16]	{Sex=Male, Survived=Yes}	=> {Age=Adult}	0.1535666	0.9209809	0.9689670	338
[17]	{Class=Crew, Survived=No}	=> {Sex=Male}	0.3044071	0.9955423	1.2658514	670
[18]	{Class=Crew, Survived=No}	=> {Age=Adult}	0.3057701	1.0000000	1.0521033	673
[19]	{Class=Crew, Sex=Male}	=> {Age=Adult}	0.3916402	1.0000000	1.0521033	862
[20]	{Class=Crew, Age=Adult}	=> {Sex=Male}	0.3916402	0.9740113	1.2384742	862
[21]	{Sex=Male, Survived=No}	=> {Age=Adult}	0.6038164	0.9743402	1.0251065	1329
[22]	{Age=Adult, Survived=No}	=> {Sex=Male}	0.6038164	0.9242003	1.1751385	1329
[23]	{Class=3rd, Sex=Male, Survived=No}	=> {Age=Adult}	0.1758292	0.9170616	0.9648435	387
[24]	{Class=3rd, Age=Adult, Survived=No}	=> {Sex=Male}	0.1758292	0.8130252	1.0337773	387
[25]	{Class=3rd, Sex=Male, Age=Adult}	=> {Survived=No}	0.1758292	0.8376623	1.2373791	387
[26]	{Class=Crew, Sex=Male, Survived=No}	=> {Age=Adult}	0.3044071	1.0000000	1.0521033	670
[27]	{Class=Crew, Age=Adult, Survived=No}	=> {Sex=Male}	0.3044071	0.9955423	1.2658514	670

Co to ale znamená - jak můžeme tato pravidla interpretovat? *lhs* je zkratka pro left hand size (levá strana), pravidla), *rhs* - je zkratka pro right hand size (pravá strana). *Support* je podpora, viz 8.7, *confidence* je spolehlivost, viz 8.8 a *lift* vypočteme dle 8.11. *Count* pak představuje počet řádků datasetu vyhovující pravidlu.

První pravidlo je jednoduše interpretovatelné - ukazuje pouze na stratifikaci datového souboru - tedy faktu, že na Titanicu bylo 2201 osob, z toho 2092 dospělých, což představuje 95 % z celkového počtu osob na plavidle.

Problém s asociačními pravidly je ale v jejich interpretaci - uvažme např. pravidla na řádcích 9 - 11 - co přesně znamenají. Když nám vyšla u pravidla 9 podpora pro pravidlo nepřežil → muž přibližně 62 % s hodnotou *lift* 1,16 - co to znamená? *Lift* > 1 naznačuje negativní závislost, tedy muž → nepřežil (nepřežil z toho důvodu, že byl muž), proč tomu tak je?

Lze formulovat různé teorie, ale faktem je, že izolovaně to nemá smysl. V tomto případě by nás zajímala alternativní pravidla pro porovnání, např. žena → nepřežila. Ve skutečnosti ani toto by nám

příliš nepomohlo. Ale se znalostí problému, můžeme pochopit co se ve skutečnosti stalo - jelikož nehoda Titaniku byla pečlivě zdokumentována a vyšetřena.

R. M. S. Titanik se potopil 15. dubna 1912 po srážce s ledovcem. Ke kolizi došlo v 23:40 místního času a k potopení lodi došlo v 02:20. Jako hlavní příčiny vysokého počtu obětí byl malý počet záchranných člunů a chybné vedení záchranných prací, kdy tyto čluny byly spouštěny na vodu navíc ještě ne zcela zaplněné.

Z hlediska záchrany byla preferována záchrana dětí a žen - odtud vyšší počet obětí z řad mužů. Lze také vysledovat spojení mezi třídou a záchrannou a to sestupně - pasažéři v první třídě byli zachraňováni častěji než ti z třídy druhé a ti zase častěji než pasažéři z třídy třetí.

Zdůvodnění v tomto případě lze hledat ve vzdálenosti, kterou museli pasažéři a posádka urazit k záchranným člunům. První třída byla nejbližší palubě, zatímco posádka byla umístěna nejbližší dnu.

Pokud se podíváme na pravidla touto optikou - co z nich můžeme vyčíst? ... nic moc, můžeme si ale pomoci, když specifikujeme, jaké pravidla nás ve skutečnosti zajímají. Řekněme, že se chceme zaměřit na přežití dětí.

Výpis 8.12: Asociační pravidla pro děti (dataset Titanic)

```
1 pravidla <- apriori(titanic.raw, parameter = list(minlen=3, supp=0.002, conf=0.2),
2   appearance = list(
3     rhs=c("Survived=Yes"),
4     lhs=c("Class=1st", "Class=2nd", "Class=3rd", "Age=Child", "Age=Adult"),
5     default="none"), control = list(verbose=F))
6 pravidla.sorted <- sort(pravidla, by="confidence")
7 inspect(pravidla.sorted)
```

Za tokových podmínek budou natrénována následující pravidla:

	lhs	rhs	support	confidence	lift	count
[1]	{Class=2nd,Age=Child}	=> {Survived=Yes}	0.010904134	1.0000000	3.0956399	24
[2]	{Class=1st,Age=Child}	=> {Survived=Yes}	0.002726034	1.0000000	3.0956399	6
[3]	{Class=1st,Age=Adult}	=> {Survived=Yes}	0.089504771	0.6175549	1.9117275	197
[4]	{Class=2nd,Age=Adult}	=> {Survived=Yes}	0.042707860	0.3601533	1.1149048	94
[5]	{Class=3rd,Age=Child}	=> {Survived=Yes}	0.012267151	0.3417722	1.0580035	27
[6]	{Class=3rd,Age=Adult}	=> {Survived=Yes}	0.068605179	0.2408293	0.7455209	151

Tento pohled nám už umožňuje srovnat šanci na přežití podle věku - co z tohoto pohledu vyvozujete? Na pomoc si vezměte údaje o zachráněných - můžete použít buď přímo dataset Titanic nebo použijte pro srovnání předpřipravená data dostupná např. na Wikipedii (doporučuji anglickou verzi článku - má přehlednější tabulku).

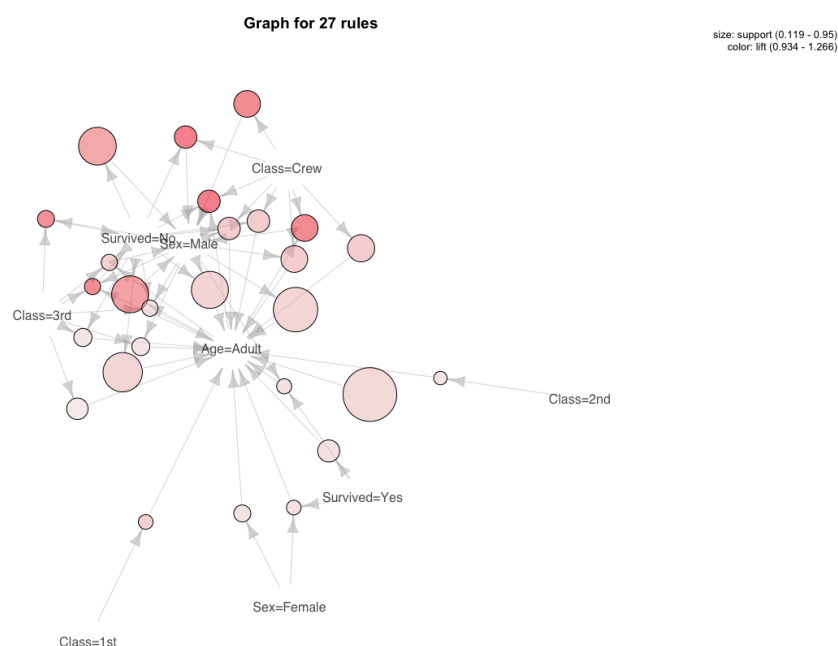
K orientaci může napomoci také vizualizace výsledku pomocí grafu. K tomuto účelu lze použít knihovnu `arulesViz` (opět před prvním použitím je nutné ji nainstalovat). Vizualizovat můžeme jako původní (viz obr. 8.9) tak více cílenou sadu pravidel (viz obr. 8.10).

Výpis 8.13: Grafická interpretace asociačních pravidel (dataset Titanic)

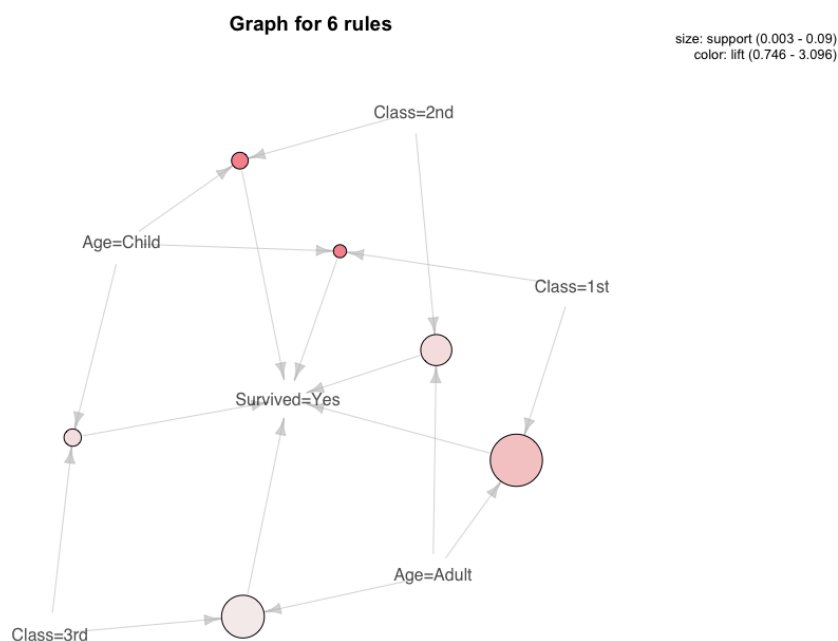
```
1 library(arulesViz)
2 plot(pravidla, method="graph")
3 plot(pravidla.sorted, method="graph")
```

Všimněte si, že pro vygenerování druhé sady asociačních pravidel jsme museli upravit parametry `apriori` metody. Zejména jsme snížili naše nároky na spolehlivost - to nám umožní hledat pravidla, která pokrývají v analyzovaném prostoru malý počet dokumentovaných případů.

Jak je patrné z výše uvedeného - výpočet pravidel samotný je jednoduchý. To co je složité, je jejich interpretace. Pro úspěšnou analýzu proto musíme mít podrobné informace o zkoumaném problému. Odvozené znalosti bychom proto měli dále podrobit kritickému zkoumání. (Alespoň než na jejich základě začneme realizovat revoluci.)



Obrázek 8.9: Asociační pravidla - dataset Titanic (27 základních pravidel)

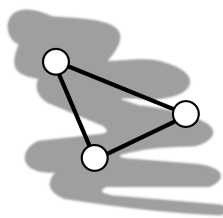


Obrázek 8.10: Asociační pravidla - dataset Titanic (zaměřeno na přeživší podle věku a třídy)



Příklad - pravidla vedoucí k přežití

Analogicky k pravidlům pro děti ve výpisu výše, formulujte asociační pravidla pro dospělé. Na základě toho, co jsme si řekli o asociačních pravidlech interpretujte výsledek - našli jste zajímavá pravidla?



Titanic dataset a jeho analýza

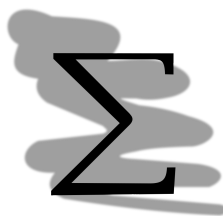
Tento dataset je jedním z nejpoužívanějších pro demonstraci tvorby asociačních pravidel. Na Internetu tak lze dohledat desítky příkladů, které dataset analyzují z různých pohledů. Tyto příklady můžete vyhledat např. pomocí vyhledávače Google pomocí řetězce: *association rules titanic example*.

Řada příkladů používá také alternativní metody prezentace výsledků, včetně některých zajímavých možností jejich vizualizace.



Realizujte příklady z této kapitoly v R

... pokud jste tak ještě neučinili. Praktická realizace příkladů Vám umožní lépe zpracovat probíranou látku, která je po teoretické stránce poměrně složitá.



Shrnutí kapitoly

Pravidlové metody, řeší problémy data miningu tvorbou pravidel. *Rozhodovací pravidla a stromy* se zaměřují na řešení klasifikačních problémů, zatímco *asociační pravidla* jsou určena k deskripci popř. hledání nugetů.

Výsledkem práce jsou vždy pravidla ve formátu IF předpoklad THEN závěr. Vyhodnocování pravidel lze zefektivnit převodem pravidel do formy rozhodovacího seznamu. V takovém případě je vyhodnocována pouze jedna podmínka, jejíž jednotlivé části se vzájemně vylučují.

Při konstrukci rozhodovacího stromu potřebujeme vodičko pro jeho efektivní stavbu. K tomuto účelu se často používá informační entropie. Pro větvení stromu používáme takový sloupec dat jehož entropie je nejmenší. Alternativně lze k tomuto účelu použít také Ginni index.

Vytvořený rozhodovací strom lze převést na pravidla.

Asociační pravidla umožňují analyzovat různé asociace existující mezi daty. K řešení lze použít celou řadu algoritmů - jedním z nejpoužívanějších je apriori algoritmus. Odvozená pravidla lze ohodnotit podle spolehlivosti, důvěryhodnosti a liftu. Finální interpretaci ale musí provést člověk v kontextu řešeného problému.



Kontrolní příklady

1. Mějme data o klientech banky ve formátu: konto, zaměstnaný, výše příjmu a přidělen úvěr - identifikujte cílový sloupec pro analýzu asociačními pravidly.
2. Jaké jsou nároky na data pravidlově orientovaných metod?
3. Jak rozhodujeme o vhodnosti datové položky k větvení rozhodovacího stromu?
4. Jaký je rozdíl mezi trénovací a validační množinou?



Řešení příkladů

1. Asociační pravidla se používají pro deskripci nebo hledání nugetů - cílový sloupec proto neurčujeme.
2. Problém jsou chybějící data a spojitě numerické veličiny.
3. Můžeme použít např. informační entropii - k větvení používáme položku s minimální entropií.
4. Trénovací množina je obvykle větší a používá se pro odvození modelu, validační se použije následně pro dohad chyby odvozeného modelu.

Kapitola 9

Bayesovská klasifikace



Náhled kapitoly

V této kapitole probereme možnosti použití bayesovského klasifikátoru pro řešení klasifikačních problémů

Po prostudování této kapitoly budete vědět

- jak fungují bayesův naivní klasifikátor
- jaké jsou výhody a omezení metody

umět

- prakticky použít metodu



Čas pro studium

Pro prostudování budete potřebovat 2 - 4 hodiny.

9.1 Úvod do Bayusovské klasifikace

Baysovský klasifikátor je odvozován od práce reverenda Theodora Bayese z krátkého pojednání o pravděpodobnosti, v rámci kterého zkoumá mezi pravděpodobností jedné události podmíněné událostí druhou a její inverzí. V moderní teorii pravděpodobnosti místo označení Bayesův teorém, popř. Bayesova věta nebo zákon používá spíše známější označení *podmíněná pravděpodobnost*.

Tento pojem by Vám neměl být zcela neznámý, neboť je vyučován v prakticky všech studijních oborech v rámci předmětů zaměřených na statistiku.

Vzorec je následující (9.1):

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \quad (9.1)$$

Ve vzorci (9.1) pracujeme s podmíněnou pravděpodobností toho že nastane H za podmínky, že nastalo E . V kontextu našeho zájmu o data mining můžeme H označit za hypotézu a E jako důkaz (z angl. evidence). Podmíněnou pravděpodobnost pak vypočteme jako podíl součinu podmíněné pravděpodobnosti E za předpokladu, že nastalo H a apriori pravděpodobnosti E s apriori pravděpodobností H .

$P(E|H)$ je právě tou mysteriální inverzí, na kterou jsme narazili v prvním odstavci této kapitoly. Apriori pravděpodobností E a H pak rozumíme zastoupení jednotlivých důkazů a hypotéz v datovém souboru bez ohledu na další informace.

$P(H|E)$ je pak aposteriorní pravděpodobnost - ta umožňuje změřit jak se změní pravděpodobnost H , za předpokladu, že nastalo E .

Zkusme tento koncept demonstrovat na příkladu. Mějte přitom prosím na paměti, že zatím se bavíme pouze o podmíněné pravděpodobnosti - nikoliv přímo o Baysově klasifikátoru.

Příklad: Uvažujme školu, které víme, že ji navštěvuje 60 % chlapců a 40 % dívek. O studentkách víme, že nosí kalhoty nebo sukně a to přibližně půl na půl, předpokládáme, že všichni studenti nosí kalhoty. Pozorovatel vidí v dálce osobu, studující na dané škole, ale jedině, co je schopen rozeznat je, že nosí kalhoty. Jaká je pravděpodobnost toho, že se jedná o dívku?

Výše uvedený příklad svou formulací nespadá do dataminingu, ale statistiky. To nám ale nevadí, naopak nám to umožní lépe pochopit podmíněnou pravděpodobnost a následně také její vazbu na klasifikaci, protože závěry zobecníme aš použijeme se z nich. Ve finále pak výše uvedený příklad přeformulujeme a převedeme jej tak na problém řešitelný dataminingem - tedy klasifikační problém.

Naši hypotézou tedy je, že student je dívka. Ze zadání proto můžeme odvodit, že podíl dívek na celkovém počtu studentů je 40 %, proto $P(H) = 0,4$.

Pravděpodobnost toho, že studující nosí kalhoty za předpokladu, že se jedná o dívku je 50 %, proto $P(E|H) = 0,5$.

Nejsložitější je pak výpočet $P(E)$, tedy pravděpodobnosti toho, že náhodně vybraný studující nosí kalhoty. Do této pravděpodobnosti nám vstupují všichni chlapci a polovina dívek. $P(E) = 0,5 \cdot 0,4 + 1 \cdot 0,6 = 0,8$.

Po dosazení do vzorce (9.1) tak získáme:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} = \frac{0,5 \cdot 0,4}{0,8} = 0,25$$

Tedy odpověď na naši otázku: Jaká je pravděpodobnost toho, že v dálce viděný student s kalhoty je dívka je 25 %.

Abychom problém převedli na klasifikační, museli bychom otázku přeformulovat. Pokud vidím v dálce studujícího s kalhoty jedná se spíše o chlapce nebo o dívku? Formálně tedy problém převádíme na klasifikační s cílovým atributem s možnostmi chlapec/dívka.

Tento případ je triviální, protože odpověď známe - jedná se spíše o chlapce, protože všichni chlapci v příkladu nosí kalhoty (na rozdíl od dívek) a zároveň je jich více.

To je ale výhoda, protože nám umožňuje posunout vnímání problému. Klasifikujeme do dvou tříd - chlapec/dívka. Vstupem do klasifikace jsou pak informace o podílu chlapců a dívek na škole a také informace o preferencích dívek stran nošení sukně.

S těmito informacemi můžeme spočítat pravděpodobnosti toho, osoba viděná v dálce je chlapec a že je dívka. Jako odpověď (zaklasifikování) pak bereme hypotézu s vyšší pravděpodobností. Matematicky tuto úvahu můžeme vyjádřit následovně (9.2):

$$H_{max} = H_J, \text{ pokud } P(H_J|E) = \max_t \frac{P(E|H_t)P(H_t)}{P(E)} \quad (9.2)$$

Tedy výsledek určitě bude jedna z existujících hypotéz H , jejíž aposteriorní pravděpodobnost bude nejvyšší. 9.2. Všimněte si také $P(E)$ - pokud se nad tímto výrazem zamyslíme zjistíme, že jeho hodnota je pro všechny hypotézy stejná (je konstantní). Pokud naším cílem je pouze identifikace nejpravděpodobnější hypotézy za dodržení předpokladu, pak konstantní hodnota ve jmenovateli může být zanedbána, protože o pořadí mezi jednotlivými hypotézami rozhodne velikost čitatele. Zanedbání tak neovlivní pořadí hypotéz.

To znamená, že můžeme tento výraz odstranit aniž bychom ztratili možnost identifikovat správně nejpravděpodobnější hypotézu H_{max} , ale s tím, že vypočtená hodnota již nebude mít charakter podmíněné pravděpodobnosti dle vzorce (9.1). Výsledek proto bude již pouze bezrozměrné číslo, už se tedy nebude jednat o procento.

Zjednodušený vzorec je níže, viz (9.3)

$$H_{max} = H_J, \text{ pokud } y = \max_t P(E|H)P(H_t) \quad (9.3)$$

Konečně jsme se tedy dostali k data miningu. S výše uvedenými vzorci je ale spojen jeden problém - berou v úvahu pouze jeden podpůrný „důkaz“. Pokud hypotéza má charakter výsledku klasifikace, který je prakticky vždy tvořen jednou položkou, pak důkazy mají charakter vstupů. O nich víme z předchozí kapitoly, že jich obvykle máme větší množství a to i pro velmi jednoduché příklady.

Vzorce (9.2) a (9.3) tedy budeme potřebovat upravit, aby braly v úvahu podporu hypotézy větším počtem důkazů, viz (9.4). Tento vzorec bývá často označován jako *naivní bayesovský klasifikátor*.

$$P(H|E_1, \dots, E_K) = \frac{P(E_1, \dots, E_K|H)P(H)}{P(E_1, \dots, E_K)} \quad (9.4)$$

Tento vzorec je možno po formálních úpravách zapsat následovně (9.5):

$$P(H|E_1, \dots, E_K) = \frac{P(H)}{P(E_1, \dots, E_K)} \prod_{k=1}^K P(E_k|H) \quad (9.5)$$

Symbol $\prod$ označuje součin. Svým použitím se tak podobá symbolu $\sum$, který sčítá. I v tomto případě představuje $P(E_1, \dots, E_K)$ statickou hodnotu, která se pro různé hypotézy nemění. Můžeme ji tedy z výpočtu vyřadit, aniž by bylo narušeno pořadí hypotéz z hlediska pravděpodobnosti, viz (9.6).

$$H_{max} = H_J, \text{ pokud } y = P(H) \prod_{k=1}^K P(E_k|H) \quad (9.6)$$

Podobně jako v případě konstrukce rozhodovacích stromů pomocí informační entropie je nutné aproximovat pravděpodobnosti pomocí relativních četností.

Výše uvedená forma klasifikátoru není jediná možná. Např. pro spojité proměnné lze použít *Gausův naivní Bayesův klasifikátor*, který vypadá následovně (9.7), pro binární hodnoty lze použít *Bernulliho naivní Bayesův klasifikátor* apod.

$$p(H = v|E_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}} \quad (9.7)$$

Pro naše demonstraci, použijeme původní *naivní Bayesův klasifikátor*, tak jak je specifikován vzorcem (9.5). Řešit budeme klasifikační problém: konkrétně naši oblíbenou klasifikaci bonity klientů banky. Jako vstup použijeme data z tabulky 8.1.

Z tabulky vypočteme jednotlivé komponenty výpočtu:

$$P(\text{úvěr(ano)}) = 8/12 = 0,67$$

$$P(\text{úvěr(ne)}) = 4/12 = 0,33$$

$$P(\text{příjem(vysoký)}|\text{úvěr(ano)}) = 5/5 = 1$$

$$P(\text{příjem(nízký)}|\text{úvěr(ano)}) = 4/7 = 0,43$$

$$P(\text{příjem(vysoký)}|\text{úvěr(ne)}) = 0/5 = 0$$

$$P(\text{příjem(nízký)}|\text{úvěr(ne)}) = 4/7 = 0,57$$

$$P(\text{konto(vysoké)}|\text{úvěr(ano)}) = 4/4 = 1$$

$$P(\text{konto(střední)}|\text{úvěr(ano)}) = 2/4 = 0,5$$

$$P(\text{konto(nízké)}|\text{úvěr(ano)}) = 2/4 = 0,5$$

$$P(\text{konto(vysoké)}|\text{úvěr(ne)}) = 0/4 = 0$$

$$P(\text{konto(střední)}|\text{úvěr(ne)}) = 2/4 = 0,5$$

$$P(\text{konto(nízké)}|\text{úvěr(ne)}) = 2/4 = 0,5$$

$$P(\text{pohlaví(žena)}|\text{úvěr(ano)}) = 4/6 = 0,67$$

$$P(\text{pohlaví(muž)}|\text{úvěr(ano)}) = 4/6 = 0,67$$

$$\begin{aligned}
P(\text{pohlaví(žena)}|\text{úvěr(ne)}) &= 2/6 = 0,33 \\
P(\text{pohlaví(muž)}|\text{úvěr(ne)}) &= 2/6 = 0,33 \\
P(\text{nezaměstnaný(ne)}|\text{úvěr(ano)}) &= 5/6 = 0,83 \\
P(\text{nezaměstnaný(ano)}|\text{úvěr(ano)}) &= 3/6 = 0,5 \\
P(\text{nezaměstnaný(ne)}|\text{úvěr(ne)}) &= 1/6 = 0,17 \\
P(\text{nezaměstnaný(ano)}|\text{úvěr(ne)}) &= 3/6 = 0,5
\end{aligned}$$

Řekněme, že budeme chtít zaklasifikovat případ k1, tedy: příjem(vysoký), konto(vysoké), pohlaví(žena), nezaměstnaný(ne).

Porovnávat pak budeme:

- $P(\text{úvěr(ano)})P(\text{konto(vysoké)}|\text{úvěr(ano)})P(\text{pohlaví(žena)}|\text{úvěr(ano)})P(\text{nezaměstnaný(ne)}|\text{úvěr(ano)})$
 $= 0,67 \cdot 1 \cdot 0,67 \cdot 0,83 = \mathbf{0,37}$
- $P(\text{úvěr(ne)})P(\text{konto(vysoké)}|\text{úvěr(ne)})P(\text{pohlaví(žena)}|\text{úvěr(ne)})P(\text{nezaměstnaný(ne)}|\text{úvěr(ne)})$
 $= 0,33 \cdot 0 \cdot 0,33 \cdot 0,17 = \mathbf{0}$

Dle výše uvedeného můžeme doporučit přidělení úvěru, což odpovídá záznamu v trénovací množině.

Všimněte si, že komponenty výpočtu jsme si předpřipravili a následně z nich pouze vybíráme čísla, která pak násobíme. Právě výpočet komponent je realizován v trénovací fázi klasifikátoru, součin vybraných komponent se pak použije během konzultace.

Z hlediska výpočtu se tak jedná o velmi efektivní postup. Tento typ klasifikátoru, je taky jednoduše použitelný pro koncepty, které se mění v čase. Nové případy tak stačí přidat do trénovací množiny a upravit vypočtené hodnoty komponent. V důsledku přidání nových případů pak dokonce nemusí dojít k přepočtení všech komponent.

Vzhledem k tomu, že přidáním nových případů do trénovací množiny dochází ke změnám v poměrech jednotlivých hodnot, přidáním nových případů automaticky kompenzuje případný drift modelu způsobený časovým vývojem modelovaného konceptu, bez nutnosti implementace mechanismu zapomínání, kterým jsme tento problém řešili u rozhodovacích stromů.

Byť výpočet není nikterak složitý, je očividné, že realizovat reálný dataminingový projekt bez podpory výpočetní techniky by bylo minimálně ... nepohodlné, pokud ne extrémně nepříjemné. Jak nám tedy s naivním Bayesovým klasifikátorem může pomoci třeba R?

Výpis 9.1: Naivní Bayesův klasifikátor pro řešení problému klasifikace klientů banky

```

1 library("e1071")
2 klienti <- read.csv("cesta/klienti_banky.csv", sep = ";", header = TRUE)
3 model <- naiveBayes(uver ~ ., data = klienti)
4 model #vypíše natrénované podmíněné pravděpodobnosti
5 scoreMnozina <- read.csv("cesta/klienti_banky_score.csv", sep = ";", header = TRUE)
6 predpoved <- predict(model, scoreMnozina)
7 predpoved #zobrazí, jak byly jednotlivé případy zaklasifikovány
8 table(pred = predpoved, true = scoreMnozina$uver) #konstrukce confusion matrix

```

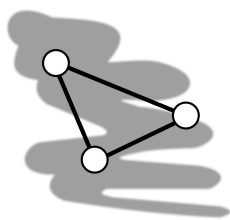
Naivní Bayesův klasifikátor je implementován např. v ne úplně intuitivně pojmenovaném toolkitu e1071 [52] známém také pod názvem *Misc Functions of the Department of Statistics, Probability Theory Group* Technické univerzity ve Vídni.

Způsob použití je pak stejný jako u jiných klasifikačních problémů - specifikujete funkci, která je zodpovědná za vytvoření modelu, v našem případě *naiveBayes*. Parametry jsou pak stejné jako v jiných funkcích tohoto typu - specifikace modelu a dat, ze kterých se model má natrénovat.

Předpověď je pak generována funkcí *predict*. Jejími parametry jsou natrénovaný model a data, na které se má model aplikovat. V příkladu výše jsou pak ještě vypsány vlastnosti natrénovaného modelu, kód pro načtení datasetů a confusion matice.

Požadavky na data by Vám měly být jasné ze způsobu, jakým probíhá výpočet. Naivní Bayesův klasifikátor má problém v situacích, kdy příklady hledaného konceptu nejsou přítomny buďto vůbec, nebo nejsou přítomny v dostatečném zastoupení. V případě, že příklady nejsou zastoupeny vůbec, model nemá z čeho odvodit pravděpodobnosti. V případě malého zastoupení pak tento typ klasifikátorů má tendenci podceňovat pravděpodobnost málo zastoupených komponent.

Logickou obranou je snaha zajistit, aby trénovací množina byla dostatečně reprezentativní, vzhledem ke zkoumanému konceptu.



Porovnejte Bayesův klasifikátor a model stromu v R

Porovnejte způsob, jakým pracujeme s naivním Bayesovým klasifikátorem a funkce vytvářející rozhodovací strom z předchozí kapitoly.

Vidíte společné znaky obou přístupů ... a čím se liší?



Příklady v R

Pokud jste ještě nevyzkoušeli realizovat příklad v R, teď je ta správná doba :-).

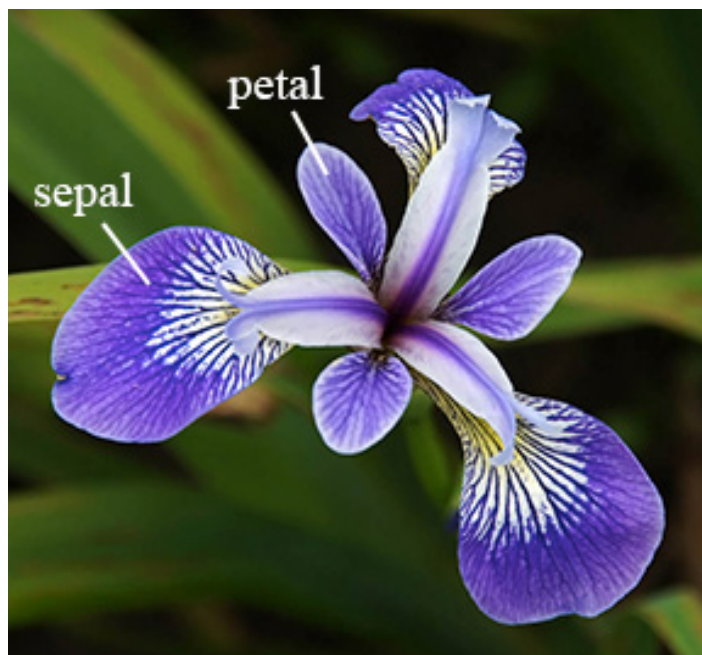
9.2 Případová studie

Pro případ Bayesova klasifikátoru budeme postupovat trošičku jinak než v případě rozhodovacích stromů. Jde o to, že v předchozí podkapitole představený přístup, resp. knihovna je použitelná prakticky pouze pro řešení klasifikačních problémů. AMES dataset ale vyžaduje zpracování odlišným způsobem - vyžaduje regresní model, byť by to nebyl klasický regresní model.

Existuje několik variant bayesovské regrese, které bychom mohli použít k řešení, ale použití vyžaduje poměrně značnou přípravu a zvládnutí dalšího nástroje (navíc k běžnému jazyku R). Pokud vás tato problematika zaujala, můžete podrobnosti najít v dokumentaci software Stan [18]. Konektivita k R je pak umožněna knihovnou rstan [17].

Zvládnutí tohoto nástroje ale dalece přesahuje možnosti tohoto předmětu. Proto použijeme raději tradiční dataset iris, které se používá často pro demonstrace klasifikačních problémů. V tomto případě budeme klasifikovat kosatce (iris) na základě zkoumání velikosti jejich okvětních lístků.

Tento dataset má pouze 5 proměnných. 4 proměnné sledují šířku a délku okvětních lístků (sepal, petal) - viz obr. 9.1 a sloupec species obsahuje druhové zatřídění kosatce.



Obrázek 9.1: Kosatec (iris) - grafický popis významu sloupců (převzato z [83])

Dataset je jiný, přesto kód pro natrénování modelu pomocí knihovny caret by Vám měl připadat už poměrně známý. Základní principy dataminingu jsou totiž pořád stejné.

Výpis 9.2: Naivní Bayesův klasifikátor pro řešení problému klasifikace kosatců na základě vlastností jejich okvětních lístků

```

1 library("klaR")
2 library("caret")
3 library("e1071")
4
5 data("iris")
6 iris_nb <- train(
7   Species ~ .,
8   data = iris,
9   method = "nb",
10  trControl = trainControl(method = "cv", number = 10)
11 )
12 iris_nb
13 table(predict(iris_nb$finalModel,iris[,-5])$class,iris$Species) #confusion matice

```

Ve funkci `train` jsme provedli drobné změny. Jako metodu používáme `nb` což je zkrácené označení pro *naive Bayes* z knihovny `e1071`. Z funkce `train` vypadlo nastavení hyperparametrů, které v případě naivního bayesovského klasifikátoru postrádá smysl.

Přesnost výsledného modelu je κ okolo 94 %, což je poměrně dobrý výsledek. κ v tomto případě označuje metriku *Cohenova κ* měřící shodu mezi dvěma hodnotiteli, kteří klasifikují N případů do C vzájemně se vylučujících kategorií. V našem případě roli hodnotitelů zastupuje predikce a reálná měření.

Alternativní pohled představuje, confusion matice, viz tab. 9.1.

Tabulka 9.1: Bayesovná klasifikace pro iris dataset - confusion matice

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	47	3
virginica	0	3	47



Iris dataset vs rozhodovací strom

Proveďte analýzu iris datasetu pomocí rozhodovacích stromů. Modifikujte příklad z předchozí kapitoly. Postupujte pomalu, samostatně a po částech. Ujistěte se, že chápete funkci jednotlivých příkazů - jak pro metodu bayesovského klasifikátoru, tak pro metodu rozhodovacích stromů.

Porovnejte confusion matice obou modelů, který z nich je lepší?

V případě, že narazíte na nepřekonatelný problém, konzultujte řešení problému dostupné v systému LMS.



Shrnutí kapitoly

Naivní Bayesův klasifikátor je odvozován z podmíněné pravděpodobnosti. Při použití zkoumáme nakolik je pravděpodobně, že hodnocený příklad přináleží k některé cílové třídě, za předpokladu, že nastaly skutečnosti definované vstupními parametry. Pro klasifikaci je použita obvykle použita nejpravděpodobnější hypotéza.

Tento druh klasifikátoru je velmi výhodný z pohledu relativně malých nároků na data a schopnosti se rychle adaptovat v měnících se podmínkách, ve kterých realizujeme klasifikaci. Změny konceptu v čase tak lze prostě zahrnout do modelu přidáním nových příkladů do trénovací množiny a přepočtení jednotlivých komponent výpočtu.

V případě, že komponenty výpočtu jsou vytvářeny na základě neúměrně malého počtu vzorků, má klasifikátor tendenci podcenit význam takové komponenty.



Kontrolní otázky

1. Co rozumíme vztahem mezi pravděpodobnostmi, že nastane jev A, za předpokladu, že nastal jev B a jeho inverzí?
2. Jaké jsou výhody použití Bayesova naivního klasifikátoru?
3. Jsou nějaké nevýhody použití klasifikátoru?
4. Interpretujte $\prod_{i=1}^n x_i$



Odpovědi

1. Existuje kvatifikovatelný a použitelný vztah mezi $P(A|B)$ a $P(B|A)$.
2. Nízké nároky na data, schopnost modelu jednoduše se adaptovat na změny v konceptu v čase.
3. Má tendenci podceňovat údaje, které nejsou v trénovací množině dostatečně zastoupeny.
4. $x_1 \cdot x_2 \cdot \dots \cdot x_n$.

Kapitola 10

Neuronové sítě



Náhled kapitoly

Neuronové sítě jsou jednou z nejrychleji se rozvíjejících oblastí umělé inteligence. Zejména nástup moderních grafických karet a specializovaných **Application Specific Integrated Circuit (ASIC)** obvodů umožnila nasazení rozsáhlých mnohavrstevných neuronových sítí umožňujících v reálném čase řešit širokou škálu úloh od autonomního řízení automobilů, po automatizované překlady, systémů text-to-speech a dalších.

Po prostudování této kapitoly budete vědět

- jak fungují vrstvené neuronové sítě

umět

- řešit jednoduché úlohy s použitím neuronových sítí



Čas pro studium

Pro prostudování budete potřebovat 2 - 4 hodiny.

Myšlenka umělých neuronových sítí není nikterak nová. Vlastně první model neuronu byl vytvořen podstatně dříve, než započal rozvoj výpočetní techniky umožnil takový model prakticky použít. Předtím než se ale vrhneme na podrobnosti, podívejme se na filozofické základy tohoto přístupu.

Pomocí neuronových sítí se snažíme napodobit způsob jakým živé organizmy myslí a to co možná nejbližší způsobu, jakým je organizován mozek - tedy pomocí neuronů a vazeb mezi nimi. Tím se přístup neuronových sítí liší od jiných oblastí umělé inteligence, např. expertních systémů, které se snaží formalizovat logiku myšlení expertů specifikací faktů a jejich použití v problémové doméně, nebo multiagentních systémů, kde jednotliví agenti mají implementovanou nějakou „malou“ funkcionalitu a pokročilé funkčnosti je dosahováno až vzájemnou interakcí jednotlivých agentů.

Neuronové sítě používané v současnosti jsou poměrně malé. Pokud to srovnáme s počty neuronů v mozcích, pak průměrný hlodavec má v mozku 12 a člověk 86 miliard neuronů [39]. Umělé neuronové sítě, které ke své práci používají velké jazykové modely (**Large Language Model (LLM)**) využívají v současnosti (IX 2025) stovky miliard až bilion parametrů. Bilion parametrů používají např. modely Gemini 2.5 Pro nebo GPT-5.

Všimněte si, že u LLM nepoužíváme počet neuronů jako základní měřítko. Pro umělé neuronové sítě je počet neuronů méně významný, díky jejich odlišné konstrukci. Schopnost učit se je primárně spojena se spojeními mezi neurony a ty mohou být v umělé neuronové síti výrazně hustější v mozku,

jak si za chvíli ukážeme. Proto sledujeme *parametry sítě*, které optimalizujeme v průběhu adaptace (učení)

Parametry rozumíme:

- *váhové koeficienty*, které charakterizují spojení mezi neurony
- *bias*, který charakterizuje výstup každého neuronu

Celkový počet parametrů umělé neuronové sítě je pak roven součtu počtu neuronů a spojení mezi nimi. Porovnání s biologickými sítěmi není vůbec jednoduché., ale pokusíme se o to.

Už víme, že průměrný člověk má přibližně 86 miliard neuronů. Budeme brát *synapse* jako ekvivalent váhy v umělé neuronové síti. Každý neuron může mít průměrně 1 000 - 10 000 synapsí. Tento počet se vyvíjí v čase. Např. neurony dětí mívají až 15 000 synapsí. Pokud informaci dáme dohromady, můžeme odvodit, že lidský mozek má přibližně 86 - 860 bilionů parametrů. Jelikož největší modely v současnosti mají okolo 1 bilionu parametrů, má lidský mozek orientačně o 2 - 3 řády více parametrů.

U biologických organismů ale z hlediska myšlení nezáleží pouze na počtu neuronů. Např. mozek průměrného primáta obsahuje 93 miliard neuronů, což je více než kolik obsahuje mozek průměrného člověka (viz [39]). To, co považujeme za inteligenci je pak odvozováno také z toho, jak je mozek organizován, tedy k čemu jsou neurony používány.

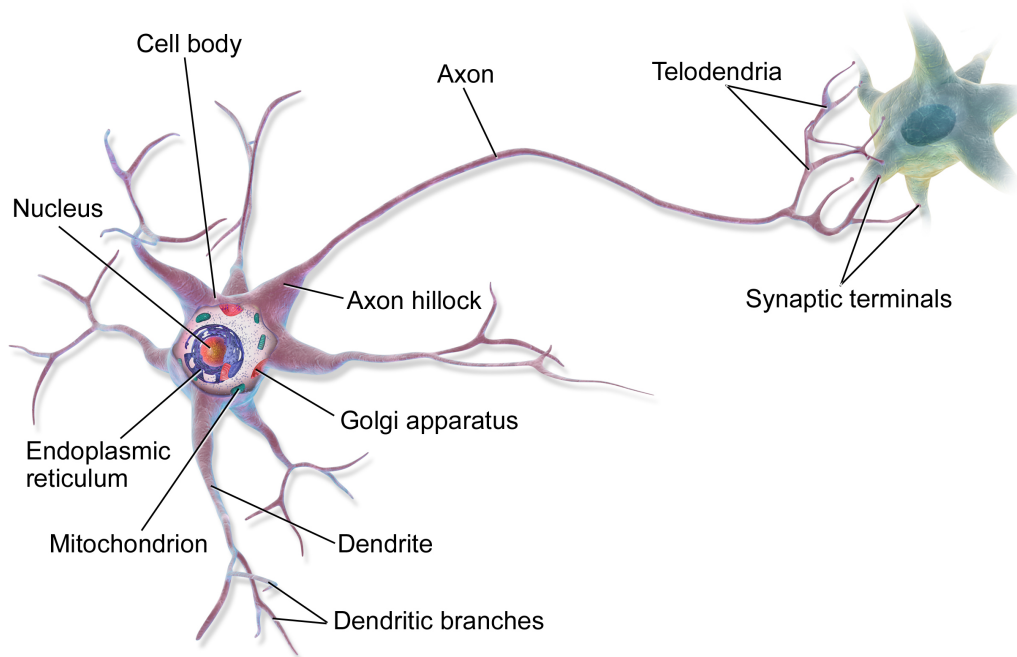
Umělé neuronové sítě, jak jsou používány dnes ke svému tréninku obvykle vyžadují nasazení poměrně značného množství strojového času superpočítače. Takové neuronové sítě jsou ale vždy specializované na řešení určitého konkrétního úkolu. Náš mozek oproti tomu je univerzální.

Lze také říci, že v současnosti výzkum ani nesměruje k vytvoření plnohodnotné umělé inteligence, byť už dnes jsou úvahy o etických otázkách spojených se spuštěním a provozováním takových systémů, viz např. iniciativa OpenAI [14].

Vraťme se ale k neuronům jako takovým a jejich implementaci pomocí počítačových modelů.

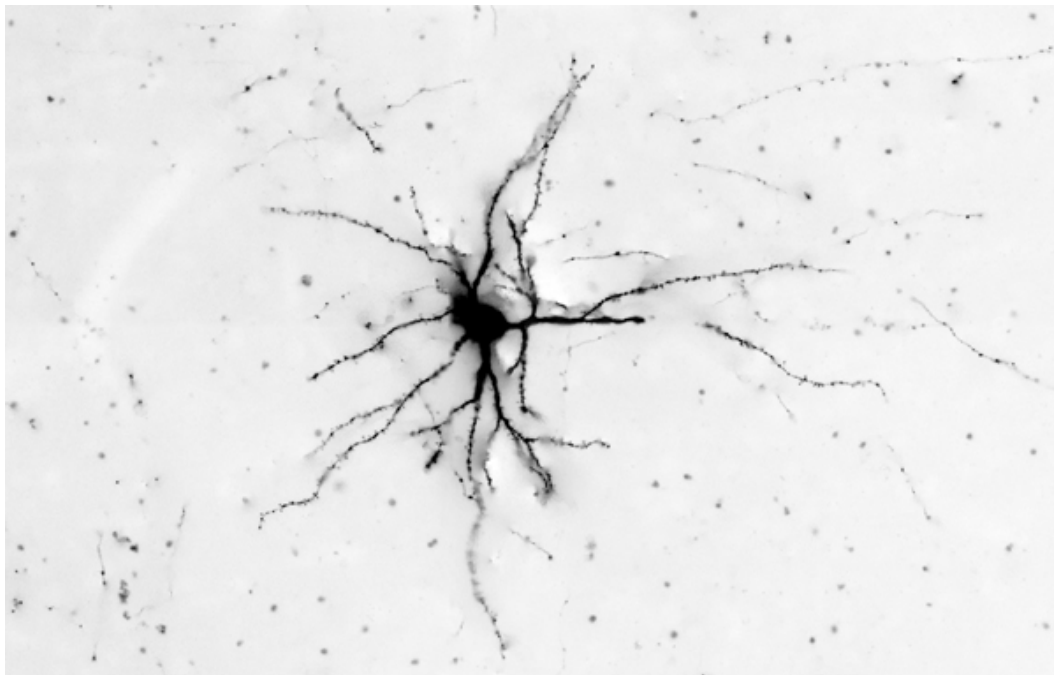
10.1 Neuron

Schéma neuronu bychom mohli vizualizovat např. jako na obr. 10.1. Tato vizualizace ve skutečnosti vizuálně příliš neodpovídá realitě - srovnajte s obr. 10.2, který byl pořízen mikroskopem.



Obrázek 10.1: Schéma neuronu (převzato z [21])

Biologický neuron je buňka a proto jako každá buňka má jádro (nucleus), které pracuje jako „elektrárna“ buňky a umožňuje ji tak plnit svou funkci. Neurony mají řadu výběžků (dendritů) a



Obrázek 10.2: Neuron vizuálního kortexu (převzato z [42])

jeden dlouhý výběžek označovaný jako axon (neurit). Pomocí axonu probíhá přenos tzv. nervových vzruchů. Výběžky axonu (telodendria) se pak připojují na další neurony.

Přenos nervového vzruchu do dalšího neuronu je realizován prostřednictvím synapse. Synapse má velmi důležitou úlohu tzv. *prahování*. To si lze představit tak, že synapse přijme nervový vzruch - pokud intenzita tohoto vzruchu přesáhne hodnotu prahu, pak jí vzruch projde a dojde k tzv. *excitaci neuronu*. Excitovaný neuron vytvoří nervový vzruch a šíří je dále svými axony na další neurony, na které je synapsami připojen. Celý proces prahování a případné excitace se opakuje.

Naopak pokud intenzita nervového vzruchů nepřesáhne hodnotu prahu, dojde k utlumení vzruchu - neuron nepřejde do excitovaného stavu a nešíří vzruch dále.

Reakci neuronové sítě mozku si tak vizuálně můžeme představit tak, že v reakci na různé podněty přijímané všemi smysly se budou excitovat odlišné trajektorie spojení neuronů v mozku - a to nám umožňuje realizovat funkci myšlení.

Umělé neuronové sítě se pak snaží tyto funkce přírodních neuronů napodobit a využít k řešení problémů.

První model neuronu je byl vytvořen v roce 1943 Warrenem McCullochem a Walterem Pittsem [50] - autoři jej nazvali *perceptron*. Perceptron poměrně věrně kopíruje funkce reálného neuronu a přes své stáří je i dnes jedním z nejpopulárnějších modelů neuronů.

Funkci perceptronu si můžeme schematicky představit jako na obr. 10.3.

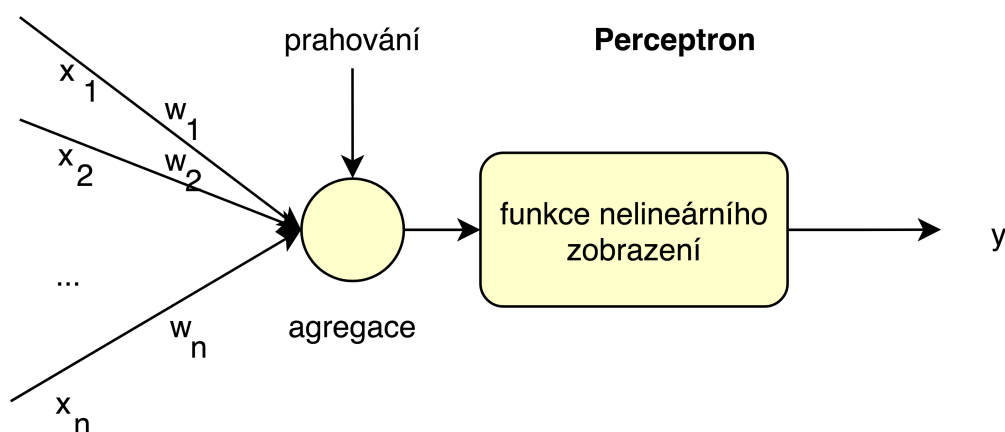
Na normované vstupy x_1 až x_n jsou aplikovány váhy w a výsledek je agregován (sečten) a podroben procesu prahování. Na výslednou hodnotu je aplikována funkce nelineárního zobrazení, která transformuje vstupní hodnotu na výstupní hodnotu y .

Matematicky lze perceptron vyjádřit následovně. Vážený vstupní signál $z(t)$ (10.1) je agregován do $u(t)$ (10.2).

$$z(t) = x(t) \cdot w(t) \quad (10.1)$$

kde z je vážený vstupní signál, x normovaný vstupní signál (obvykle v intervalu 0-1), w jsou pak váhy.

$$u(t) = \sum_{i=1}^n x_i(t) \cdot w_i(t) \quad (10.2)$$



Obrázek 10.3: Schématické znázornění modelu perceptronu

Prahování se provádí tak, že se od agregovaných vstupních signálů $u(t)$ odečte hodnota prahu w_0 , viz rovnice (10.3). Hodnota prahu je tím, o čem jsme hovořili v souvislosti s parametry neuronové sítě a biasem (posunem) neuronu.

$$v(t) = u(t) - w_0 \quad (10.3)$$

Funkci nelineárního zobrazení volíme podle požadavků, které klademe transformaci prahované hodnoty $v(t) \rightarrow y$ na hodnotu výstupní. Jednou z nejčastěji používaných funkcí tohoto typu je tzv. *sigmoida*, kterou je možno vypočítat dle vzorce (10.4). Graficky je pak sigmoida znázorněna na obr. 10.4.

$$y(t) = \frac{1}{1 + e^{-v(t)}} \quad (10.4)$$

Všimněte si, že sigmoida je spojitou funkcí, ale lze uvažovat také o jiných modelech transformace - např. $y(t) = 0$ pro $v(t) < 0$, $y(t) = 1$ pro $v(t) \geq 0$. Taková funkce je nespojitá v $x = 0$ y skočí z 0 na 1.

U opravdu velkých neuronových sítí se ale používá často jiná funkce delineárního zobrazení **Rectified Linear Unit (ReLU)**, kterou vypočteme následovně:

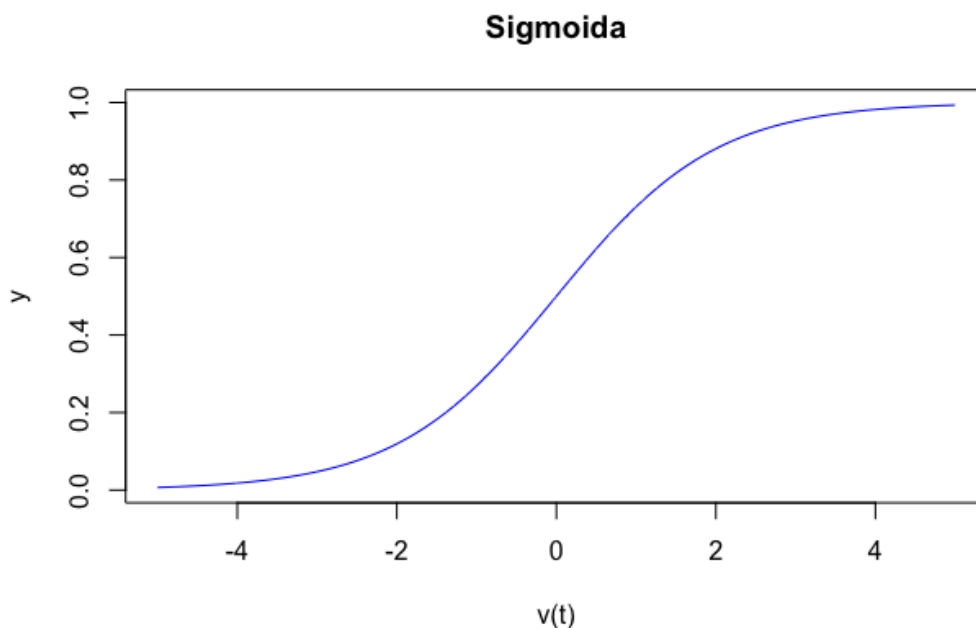
$$y(t) = \max(0, v(t)) \quad (10.5)$$

Použití ReLU má řadu výhod, kromě jednoduššího výpočtu také lépe podporuje proces adaptace.

10.2 Neuronové sítě a jejich adaptace

V minulé podkapitole jsme se seznámili s modelem neuronu, jeden neuron nám ale problém nevyřeší. Řešení je možné pouze při zapojení, obvykle velkého množství, neuronů do sítě. Přitom platí, že čím je síť rozsáhlejší (s čím větším počtu neuronů se skládá) tím dokáže být při řešení problémů přesnější a tím je také z hlediska portfolia problémů řešitelných tímto způsobem univerzálnější.

V průběhu doby se objevovala celá řada typů sítí. V prvopočátcích se pracovalo pouze s jedinou vrstvou v síti. Uvažujme problém: *potřebujeme vyřešit problém rozpoznávání znaků v naskenovaném textu. Jako vstup nám slouží stav pixelu obrázku rozlišovaného znaku. Uvažujeme přitom 2 stavy re-representované bílou a černou barvou znaku - černá znamená součást písmene, bílá znamená papír.*



Obrázek 10.4: Graf sigmoidy

Znak A pomocí jednoduché jednovrstevné sítě bychom mohli znázornit třeba jako na obr. 10.5a. Z hlediska vstupů do neuronové sítě lze místo neuronů (na obr. kolečka :-)) dosadit hodnotu 0 (bílá) nebo 1 (černá). Výhodou použití neuronové sítě je pak to, že neuronová síť správně určí nejen když bude rozložení „aktivních“ neuronů přesně odpovídat písmenu, ale také případy kdy je reprezentace písmene do určité míry poškozená. Proto např. také 10.5b bude vyhodnoceno jako písmeno A.

Jednovrstevné sítě jsou právě vhodné pro zpracování grafických informací. V moderních sítích, např. pro účely identifikace objektů v prostoru pro samořiditelná auta je ale používána celá řada takových sítí. Jejich výstup je pak integrován do komplexního modelu prostředí, který slouží jako podklad pro rozhodování řídicí jednotky automobilu.

Podrobnosti lze nalézt např. v článku NVidie [23].

Více vrstevné neuronové sítě jsou z hlediska použití mnohem univerzálnější. Na obr. 10.6 je znázorněn zjednodušený příklad takové sítě.

Všimněte si, že vazby mezi neurony v síti 10.6 jsou orientované. Vstupy ve vstupní vrstvě x_1 až x_n procházejí v tomto případě jednou pracovní vrstvou, výstupní vrstvu v tomto případě tvoří dva neurony y_1 a y_2 .

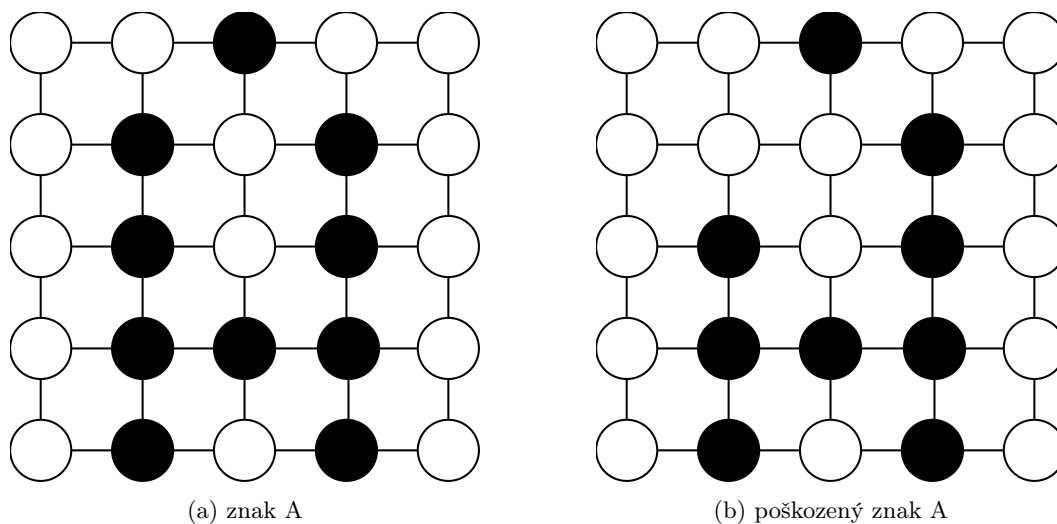
Pracovní vrstvy se někdy označují také jako skryté. To je dáno tím, že vstupy zadáváme a výstupy y pak odečítáme, ale uvnitř neuronová síť funguje jako černá skříňka. Pracovních vrstev v síti může být libovolné množství. Přitom počet neuronů v takové síti není nijak stanoven, většinou jich ale volíme více než je ve vstupní vrstvě. Předpokládáme přitom, že neuronů ve vstupní vrstvě je více než ve vrstvě výstupní.

Volba architektury sítě tak není exaktně daná. Není tak od věci experimentovat s různou velikostí sítě apod. a porovnávat jakých výsledků bude dosaženo.

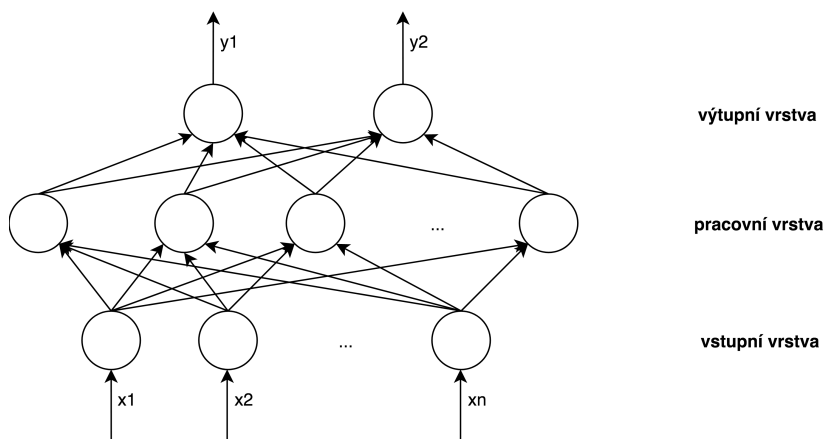
Všimněte si také, že strukturálně je každý neuron předchozí vrstvy připojen na každý neuron vrstvy následující. Prakticky to znamená, že s rostoucím počtem neuronů v síti roste počet vazeb v sítích tohoto typu exponenciálně. To pak na nás může klást jistá omezení z hlediska „vypočitatelnosti, resp. adaptovatelnosti“ takových sítí. Složitost roste exponenciálně, ale výkon výpočetní techniky jsme obvykle schopni škálovat pouze lineárně.

Zkusme zrealizovat jednoduchou demonstrační síť pro predikci průběhu funkce, jako alternativu lineární regrese, se kterou jste se mohli setkat např. v předmětu Statistika.

Mějme jednoduchou funkci $y = x^2$. Řekněme, že trénovací množinu (viz tab. 10.1) budeme mít hodnoty $x \in (-10; 10)$ po celých číslech.



Obrázek 10.5: Jednoduchá neuronová síť pro OCR



Obrázek 10.6: Schéma vícevrstevné sítě

Tabulka 10.1: Trénovací množina funkce $y = x^2$

input	output
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

Výpis 10.1: Řešení funkce druhé mocniny pomocí neuronové sítě

```

1 library("neuralnet")
2 mydata <- read.csv("cesta/nn_fce_2mocnina.csv", sep=";", header=TRUE)
3 #trénování neuronové sítě
4 model <- neuralnet(formula=output~input, data=mydata, hidden=10, threshold=0.01)
5 model_lm <- lm(output~input, data=mydata) #pro srovnání lineární regrese

```

```

6 plot(model) #vykreslení vrstev sítě v (síťový graf)
7 #porovnání dat - skutečná a predikce
8 final_output <- cbind(mydata$input, mydata$output,
9                       as.data.frame(model$net.result),
10                      as.data.frame(model_lm$fitted.values))
11 colnames(final_output) <- c("Input", "Expeceted output", "NN Output", "LM Output")
12 plot(final_output$'Expeceted output', col="red", main="Predikce vs realita",
13      xlab="cisla", ylab="druha mocnina")\
14 lines(final_output$'NN Output', col="blue")
15 lines(final_output$'LM Output', col="green")
16 legend(1,95, legend=c("skutečne hodnoty", "predikce NN", "predikce LM"),
17       col=c("red", "blue", "green"), lty=1:3, cex=0.8)

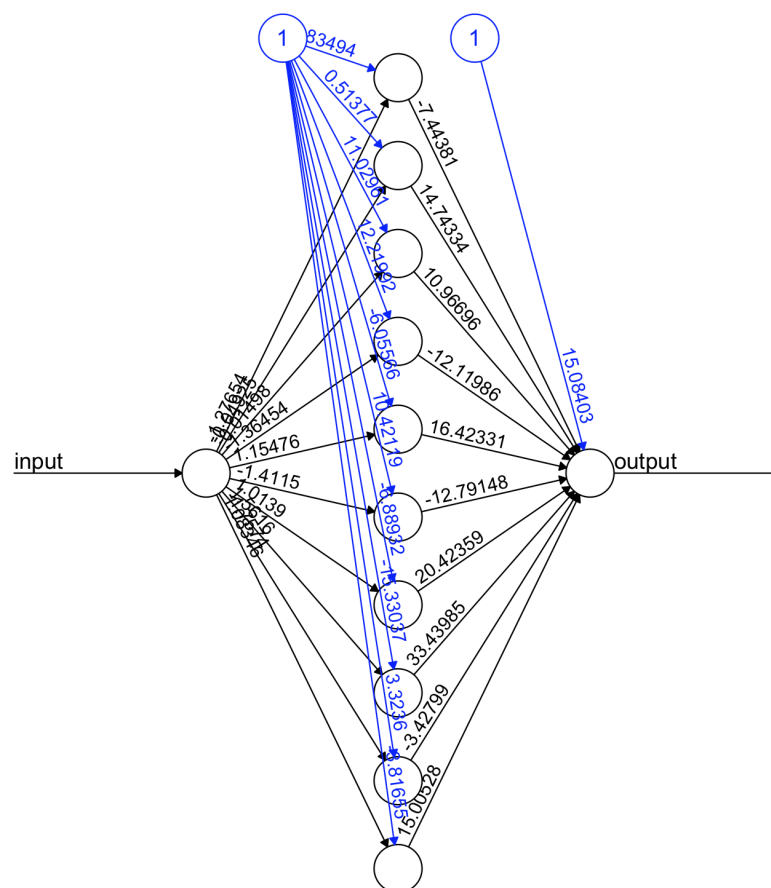
```

Pro demonstraci použijeme toolkit *neuralnet* [35]. Který umožňuje realizaci jednoduchých modelů neuronových sítí. (Pro pokročilejší práci je ale vhodnější použít některé pokročilejší toolkity.)

Model je načten ze souboru CSV, a natrénován pomocí funkce *neuralnet*. Počet neuronů vstupní a výstupní vrstvy je dán strukturou zvoleného modelu (parametr *formula*). V našem případě, bude mít tedy jede vstupní a jeden výstupní neuron. Počet neuronů pracovní vrstvy jsme nastavili parametrem *hidden* na 10. Pomocí parametru *threshold* nastavujeme požadovanou přesnost modelu.

Prosím všimněte si, že chyba nemůže být nastavena na nula - důvodem je tzv. problém *přeučení* neuronové sítě. K problému přeučení se ale ještě vrátíme.

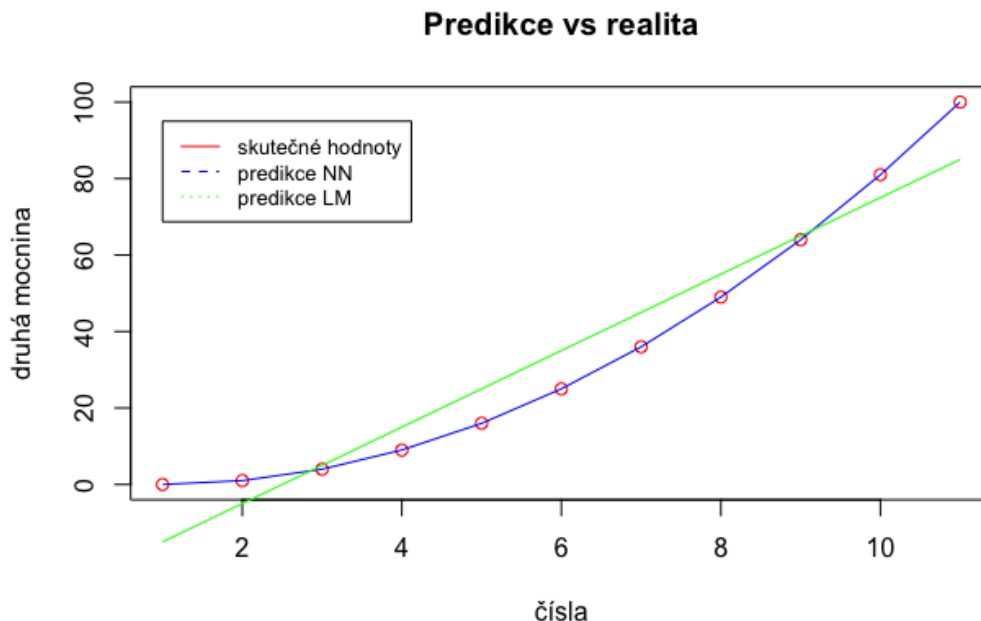
Struktura neuronové sítě je pak vykreslena pomocí funkce *plot*, viz obr. 10.7.



Obrázek 10.7: Neuronová síť pro predikci hodnot funkce $y = x^2$

Zbývající část skriptu R pak slouží pro přípravu porovnání modelu lineární regrese, neuronové sítě a skutečnosti a vykreslení do grafu na obr. 10.8.

Porovnání z obr. 10.8 jasně ukazuje, že model neuronovou sítí je přesnější než námi zvolený model lineární regrese. K tomu je potřeba dodat, že by tomu tak být nemuselo. V našem případě není důvod, abychom nebyli schopni odvodit regresní model s podobnou přesností.



Obrázek 10.8: Funkce $y = x^2$ - srovnání modelu neuronovou sítí a lineární regrese

Místo toho, ale tuto situaci použijeme pro zdůvodnění jedné z nejsilnějších stránek použití neuronové sítě. Lineární model totiž po nás požadoval přijmout jisté předpoklady o vzhledu modelu. V našem případě, že bude vypadat následovně: $y = a + bx$, kde koeficienty a a b budou odvozeny.

My samozřejmě víme, že správně model měl být $y = x^2$, ale informace tohoto typu obvykle není dostupná - konečně, pokud víme, jak vypadá funkce, není potřeba se snažit ji odvodit.

Neuronová síť proti tomu po nás nežádala přijmout žádný předpoklad o fungování modelu. Specifikovali jsme pouze základní strukturu neuronové sítě a model se odvodil sám. Ve skutečnosti mohla být struktura neuronové sítě podstatně jednodušší - mohlo stačit možná 5 možná ještě méně neuronů. Můžete sami vyzkoušet kolik neuronů z pracovní vrstvy můžete odebrat aniž by to zanechalo následky na přesnosti modelu.

Jak se tedy vůbec neuronová síť adaptuje?

Pro adaptaci se většinou používá tzv. *učení s učitelem*. To znamená, že adaptačnímu algoritmu se předkládá trénovací množina ve formátu vstup → výstup, což umožňuje pro neuronovou síť porovnáním s jejím skutečným výstupem vypočítat chybu. Tato informace je pak použita pro další iteraci adaptace sítě.

Představit si to lze tak, že na začátku procesu adaptace je všem vazbám mezi neurony v síti přiřazena náhodná hodnota váhy w_n . Adaptaci pak postupně tyto váhy měníme s cílem konvergovat postupně ke specifikované chybě.

Nejpoužívanější metodou pro tento účel je tzv. *backpropagation* - česky metoda *zpětného šíření chyb*. Matematicky je metoda založena na výpočtu globální chyby GE jako rozdílu očekávaného a skutečného výstupu sítě, viz (10.6).

$$GE = \sum_{i=1}^n (y_i - d_i)^2 \quad (10.6)$$

Kde GE je globální chyba (global error), y je očekávaný výsledek a d skutečný výsledek sítě.

Z matematiky víme :-), že k nalezení minima funkce potřebujeme tuto funkci derivovat (10.7).

$$\Delta w = -\eta \frac{\partial GE}{\partial w} \quad (10.7)$$



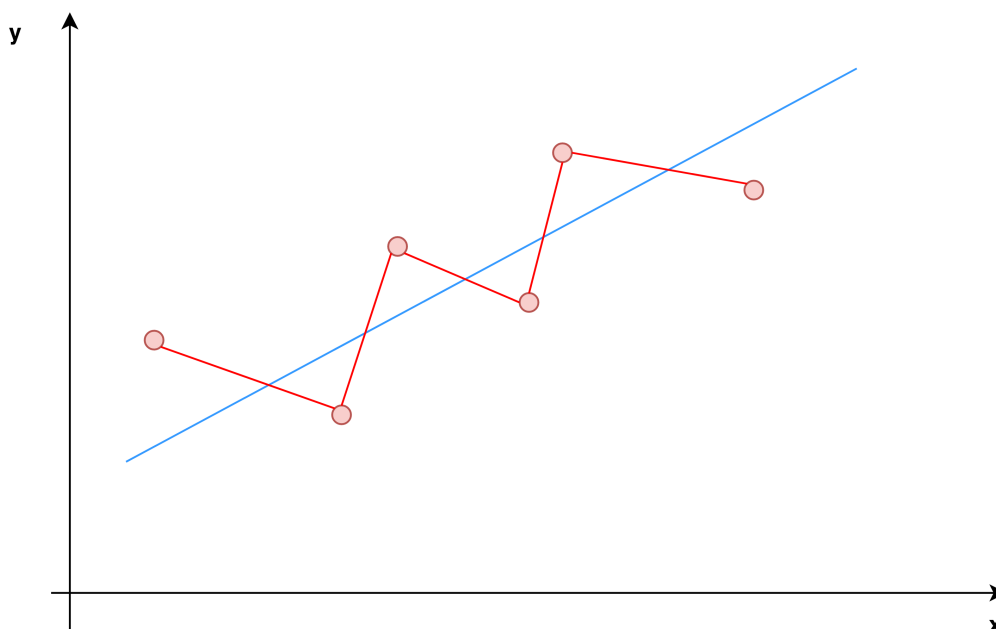
Proč součet čtverců?

Podívejte se na vzorec 10.6 - proč myslíte, že je pro výpočet GE použit součet čtverců a ne např. $GE = \sum y_i - d_i$?

Úkolem hodnoty η je pak zajistit, aby Δw bylo velmi malé. Zdůvodnit si to můžeme tak, že při lehkých změnách riskujeme, že budeme oscilovat okolo žádoucí hodnoty. Volbou $\eta = 0,1$ nebo $\eta = 0,01$ nebo dokonce menší zajistíme, že proces adaptace bude pomalu směřovat k cíli. To ... *pomalu* je bohužel pro rozsáhlejší sítě potřeba zdůraznit.

Minimalizace globální chyby má svá omezení. U neuronových sítí riskujeme u příliš malých chyb možnost *přeučení*. Přeučená neuronová síť u případů z trénovací množiny dosahuje minimální chyby, ale u příkladů, které nejsou v trénovací množině je chyba naopak výrazně vyšší.

Graficky si to můžeme představit podobně jako na obr. 10.9.



Obrázek 10.9: Přeučení model

Vstupem pro modely vizualizované na obr. 10.9 jsou body (na obr. červená kolečka). Pro tyto body byly vytvořeny dva modely reprezentované modrou a červenou křivkou. Všimněte si, že červená křivka přesně kopíruje zadané body, zatímco modrá jde mezi nimi.

V tomto případě je modrá křivka přesnější, protože rozmístění bodů okolo této linie bylo způsobeno náhodnou chybou měření. Taková chyba se v reálu objevuje prakticky pokaždé. Přílišnou snahou o přesnost, která vedla k odvození červeného modelu, jsme tak ve skutečnosti zvětšili výrazně chybu, s jakou model pracuje.

Z tohoto důvodu je potřeba se smířit s jistou nevyhnutelnou chybou, kterou model bude obsahovat vždy a která je vynucena fyzickými charakteristikami způsobu jakým byla data pořízena.

Problém přeučení není problém čistě jen neuronových sítí - podobný problém mají třeba rozhodovací stromy - i ty lze přeučit. Řešení v takovém případě spočívá v „prořezání stromu“. Postupně odebíráme listové uzly a sledujeme, jestli dochází ke zvyšování přesnosti řešení.

Řešení klasifikačního problému probíhá vlastně stejně jako probíhalo odvození modelu funkce $y = x^2$. Pro demonstraci můžeme použít opět příklad klientů banky, viz tab. 8.1.

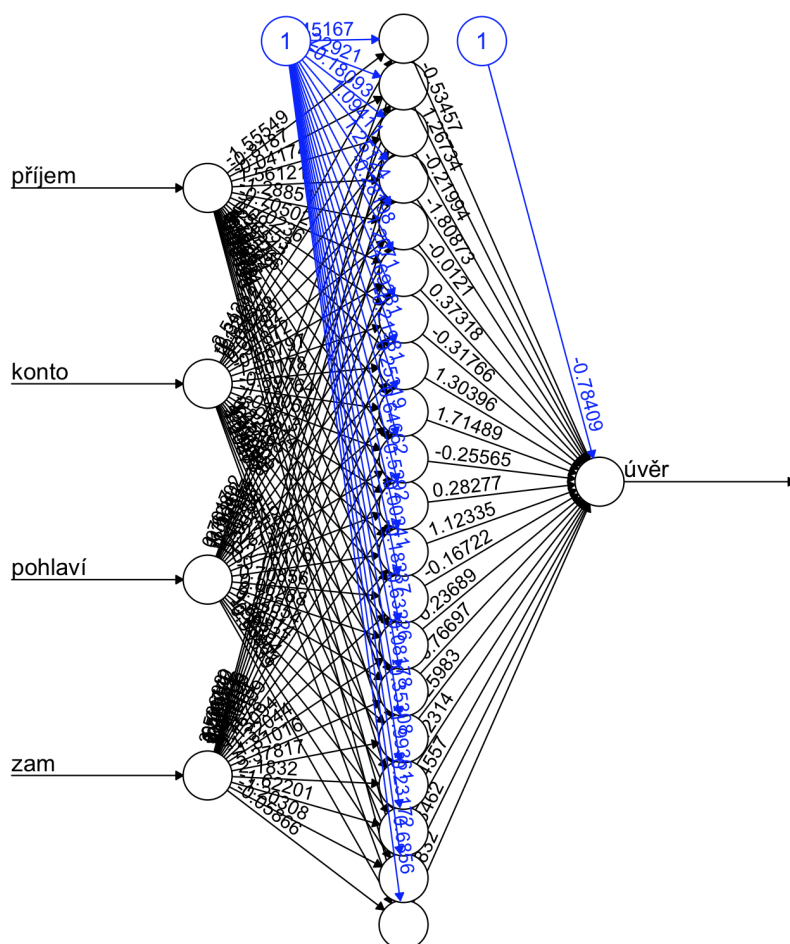
Výpis 10.2: Klienti banky - řešení neuronovou sítí

```

1 library("neuralnet")
2 mydata <- read.csv("cesta/klienti.csv", sep=";", header=TRUE)
3 model <- neuralnet(formula=uver~prijem+konto+pohlavi+zam,
4                   data=mydata, hidden=20, threshold=0.01)
5 plot(model)
6 final_output <- cbind(mydata$uver, as.data.frame(model$net.result))
7 colnames(final_output) <- c("ocekavano", "spocteno")
8 final_output

```

Pouze v tomto případě jsem do pracovní vrstvy zvolil 20 neuronů. Vizuálně neuronová síť vypadá jako na obr. 10.10.



Obrázek 10.10: Klasifikace klientů banky neuronovou sítí

Modely tvořené desítkami neuronů je možno na moderních počítačích počítat prakticky v reálném čase. 20 neuronů skryté vrstvy je tak pravděpodobně nadsazený počet, ale nějaké výrazné úspory času bychom z hlediska výpočtu nedosáhli.

Výsledek sítě je znázorněn v tab. 10.2.

Všimněte si, že neuronová síť nespočte nikdy číslo zcela přesně, takže 1 není nikdy zcela přesně 1, ale něco mezi 0,98 a 1,1 a podobně je to s nulou. Všimněte si, že přesto je výsledek poměrně přesný.

10.3 Případová studie

Neuronové sítě jsou schopny řešit jak regrese, tak klasifikační problémy. Pro demonstraci se vrátíme k AMES datasetu. Základní výpočet provedeme pomocí knihovny nnet. Tato knihovna má ale nevýhodu

Tabulka 10.2: Neuronová síť predikce vs realita ($y = x^2$)

	očekáváno	spočteno
1	1	1.0019334196327
2	1	1.0069034997006
3	0	-0.0130942067801
4	1	0.9825741850726
5	1	0.9861262522598
6	0	-0.0033695338052
7	1	1.0086557396241
8	1	0.9899790546700
9	0	-0.0002532946061
10	1	0.9844730618735
11	0	0.0380269653583
12	1	1.0135488795748

Realizace příkladů v R

Pokud jste dosud ještě nerealizovali příklady v R, teď je k tomu vhodná doba.

**Úpravy skriptu modelu funkce $y = x^2$**

Zajisté jste si všimli, že naše ověření kvality modelů není úplně korektní - pro ověření používáme trénovací množinu. Vzhledem k tomu, že známe funkci, kterou se modelem snažíme pokrýt můžeme jednoduše vygenerovat validační množinu novou, např. $0,5 \rightarrow 0,25$, $1,5 \rightarrow 2,25$, ... a provést ověření zcela konkrétní ... vzhůru do toho!



- nepodporuje hloubkové učení a tak její schopnost adaptovat se na takový problém, jako je regresní model s 80-ti prediktory není dobrá, viz obr. 10.11. Výpis níže je minimalistickým řešením modelu - jeho přesnost tak lze přenastavením jednotlivých parametrů modelu ještě dále zlepšovat.

Výpis 10.3: AMES dataset - řešení pomocí nnet

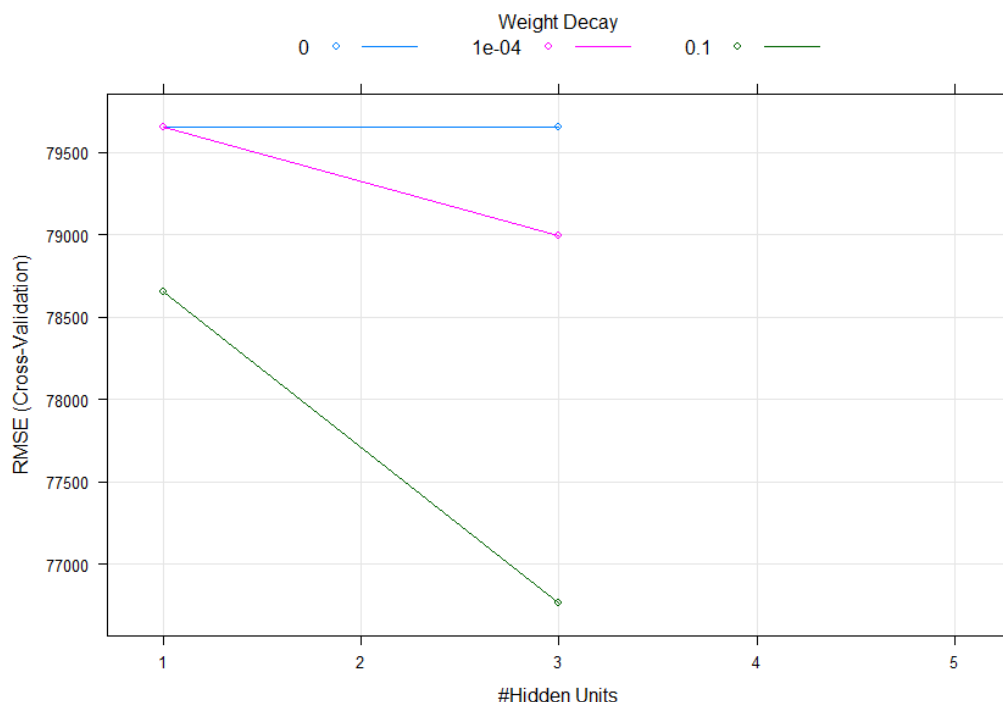
```

1 library("AmesHousing") #obsahuje dataset AMES
2 library("nnet")
3 library("caret")
4 library("vip")
5
6 ames_train <- make_ames()
7 ames_nn <- train(
8   Sale_Price ~ .,
9   data = ames_train,
10  method = "nnet",
11  trControl = trainControl(method = "cv", number = 10),
12  linout = TRUE
13 )
14 ames_nn
15 plot(ames_nn)
16 vip(ames_nn, num_features = 26, bar = FALSE)

```

Jak je vidno na obr. 10.11 optimalizuje hyperparametry caret do určité míry sám nastavením skrytých neuronů (hidden units) a weight decay - jedná se o parametr, jehož hodnota je menší než 1 a používá se tak, že se jím vynásobí hodnota vah odvozených modelem. Cílem je, aby v rámci iterace změny ve váhových koeficientech probíhaly pomalu ale jistě.

Hodnota RMSE pro nejlepší kombinaci parametrů je okolo 76 000, pokud to porovnáme s odvozeným rozhodovacím stromem, tak rozhodovací strom je ve skutečnosti mnohem přesnější (RMSE okolo 44 000).



Obrázek 10.11: RMSE pro různé modely neuronové sítě datasetu AMES

Pokud se podíváme na preferované parametry, využívá se jich aktivně v modelu pouze relativně málo, viz obr. 10.12.

Věnujte prosím pozornost parametru `linout` funkce `train`. Nastavení na `TRUE` znamená, že neuronová síť má počítat s tím, že hodnoty předložené k tréninku nejsou normalizovány. Bez tohoto parametru by neproběhl výpočet - byly by vyžadovány transformace nad původními daty.

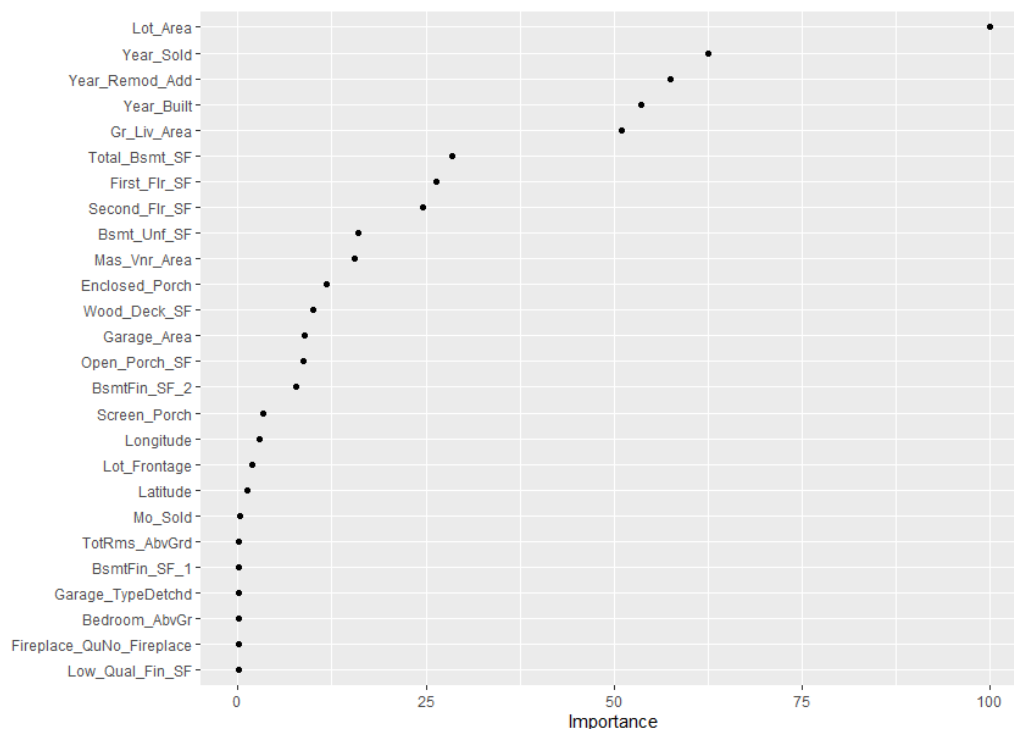
Pro zlepšení parametrů modelu můžeme zkusit ladit hyperparametry modelu, což jsou právě parametry `hidden neurons` a `weight decay`. Obr. 10.11 naznačuje, že model není příliš citlivý na parametr `hidden neurons`. Bez specifického nastavení rozsahu hyperparametrů byly testovány hodnoty 1, 3 a 5, přičemž ke zlepšení dochází pouze při přechodu z 1 na 3 skryté neurony, navýšení na 5 už žádný přínos nepřináší.

Zajímavější je situace s hyperparametrem `weight decay`, který se zvyšující se hodnotou poskytuje lepší předpovědi. Zkusme proto upravit funkci `train` přidáním instrukcí pro trénování hyperparametrů: `tuneGrid=expand.grid(.size = c(3, 5, 10, 20), .decay=seq(0.1, 0.9, by=0.1))`. `.size` odpovídá hyperparametru `hidden units` a je nastaven na hodnoty 3, 5, 10 a 20. Vzhledem ke zkušenostem z minulého pokusu, ale v tomto případě nelze očekávat zlepšení. `.decay` je nastaven jako sekvence od 0,1 do 0,9 zvětšovaná po 0,1. Zde lze očekávat zlepšení.

Výsledek ladění hyperparametrů je na vývoji RMSE znázorněn na obr. 10.13. Jak jsme očekávali jiná hodnota `size` než 3, nemá pro model smysl. Pro `decay` je ale situace jiná. Na hodnotě 0,4 jsme dosáhli významného snížení RMSE až téměř k 70 000, což odpovídá zlepšení téměř 8 %.

Z výše uvedeného vyplývá, že optimum pro hyperparametr `.decay` se pohybuje někde mezi 0,3 - 0,5. Otestovat to lze např. následující úpravou výpočtu: `tuneGrid=expand.grid(.size = 3, .decay=seq(0.3, 0.5, by=0.05))`. Pro zrychlení výpočtu je `.size` zafixováno na hodnotě 3. Nakolik přesné bude natrénování neuronové sítě závisí ale také do určité míry na náhodě. Opakovaným spouštěním budou dosahovány tedy různé výsledky. Nejlepší hodnota, které jsem byl schopen dosáhnout byla RMSE 68 000 při `.decay` 0,4.

Dalšího zpřesnění je možno dosáhnout zvýšením počtu iterací. To je možno udělat přidáním parametru `maxit = 1000` do funkce `train`. `Maxit` parametr odpovídá maximálnímu počtu iterací, které výpočet může ve fázi učení udělat. Implicitní, přednastavený počet je nastaven na 100. Pro takový model je potřeba mít na paměti, že délku výpočtu prodlužujeme 10x. Na běžném čtyřjádrovém notebooku se tak síť bude adaptovat desítky minut.



Obrázek 10.12: Relativní významnost jednotlivých parametrů modelu neuronové sítě datasetu AMES

Z tohoto důvodu ke kroku prodloužení výpočtu přistupujeme až v okamžiku, kdy máme jasnější představy o nastavení hyperparametrů modelu a můžeme tak jejich rozsahy patřičným způsobem omezit. Tímto způsobem jsme se ale adaptací dostali na RMSE = 58 500, což je výsledek skoro o 33 % lepší, byť to stále nedosahuje kvalit rozhodovacího stromu.

Kód níže může neobsahovat už odkazy na knihovny ani příkaz pro načtení AMES dat do proměnné.

Výpis 10.4: AMES dataset - řešení pomocí nnet s optimalizací hyperparametrů a vyšším počtem iterací

```

1  ames_nn4 <- train(
2    Sale_Price ~ .,
3    data = ames_train,
4    method = "nnet",
5    trControl = trainControl(method = "cv", number = 10),
6    tuneGrid=expand.grid(.size = 3,.decay=seq(0.3, 0.5, by=0.05)),
7    maxit = 1000,
8    linout = TRUE
9  )
10 ames_nn4
11 vip(ames_nn4, num_features = 26, bar = FALSE)
12 plot(ames_nn4)

```

To není úplně špatný výsledek - je ho ale ještě možné vylepšit? Z hlediska schopností jednoduchých vrstvených neuronových sítí se pomalu blížíme k hranici možností tohoto postupu. Další možnost zlepšení poskytují až metody tzv. hloubkového učení (deep learning). Příklady, které si v této kapitole ukazujeme odpovídají z pohledu algoritmů někdy přelomu tisíciletí. Hloubkové učení, které také využívá neuronové sítě se pak mohutně rozvíjí v posledních dvou desetiletích.

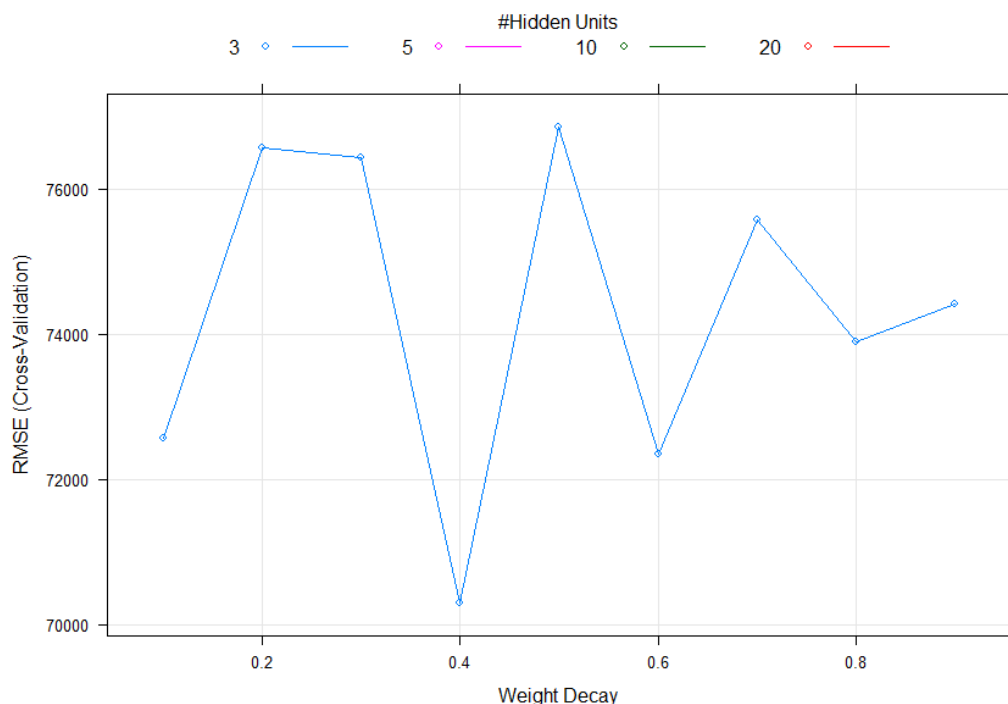
Použití hloubkového učení tak může výsledek výrazně zpřesnit. Přesto i v současném postupu máme jisté rezervy. Dosud jsme totiž optimalizovali algoritmus, nemanipulovali jsme s daty samotnými. A když už jsme u toho, nezapomněli jsme na nějaký výpočetní výkon, který necháváme ladem? Podívejme se na kód níže a tentokrát se jedná o úplný výpis:

Výpis 10.5: AMES dataset - řešení pomocí nnet s optimalizací hyperparametrů; vyšším počtem iterací a preprocesingem prediktorů

```

1  library("AmesHousing") #obsahuje dataset AMES

```



Obrázek 10.13: Ladění hyperparametrů `.size` (3, 5, 10, 20) a `decay` (0,1 - 0,9) pro AMES dataset

```

2 library("nnet")
3 library("caret")
4 library("vip")
5 library("dplyr")
6 library("plotly")
7 library("doParallel")
8
9 registerDoParallel(cores = 4)
10
11 ames_train <- make_ames()
12 ames_nzv <- nearZeroVar(ames_train)
13 ames_train2 <- ames_train[, -ames_nzv]
14 ames_nn5 <- train(
15   Sale_Price ~ .,
16   data = ames_train2,
17   method = "nnet",
18   preProcess = c('nzv', 'center', 'scale'),
19   trControl = trainControl(method = "cv", number = 10),
20   tuneGrid=expand.grid(.size = 3:9, .decay=seq(0.3, 0.5, by=0.05)),
21   maxit = 100,
22   linout = TRUE
23 )
24 ames_nn5
25 vip(ames_nn5, num_features = 26, bar = FALSE)
26 plot(ames_nn5)

```

Ve výpočtu nám přibila knihovna *doParallel*, která zpřístupňuje některé funkce pro paralelní zpracování modelů. Dosud jsme totiž výpočty realizovali v režimu jednovláknovém. Moderní procesory ale obsahují 2, 4, 8 nebo dokonce více jader a to i na cenově dostupných a „málo výkonných“ notebookech! Příkaz `registerDoParallel(cores = 4)` tak říká R, aby pro výpočet použil 4 jádra, což odpovídá počtu jader procesoru notebooku, na kterém byla tvořena tato skripta.

Tímto způsobem nelze zlepšit výkon všech výpočtů, ale ty, které provádíme my v této (a předchozích) kapitolách jsou dobře paralelizovatelné a tak se rychlost výpočtu v našem případě ze 4-násobí.

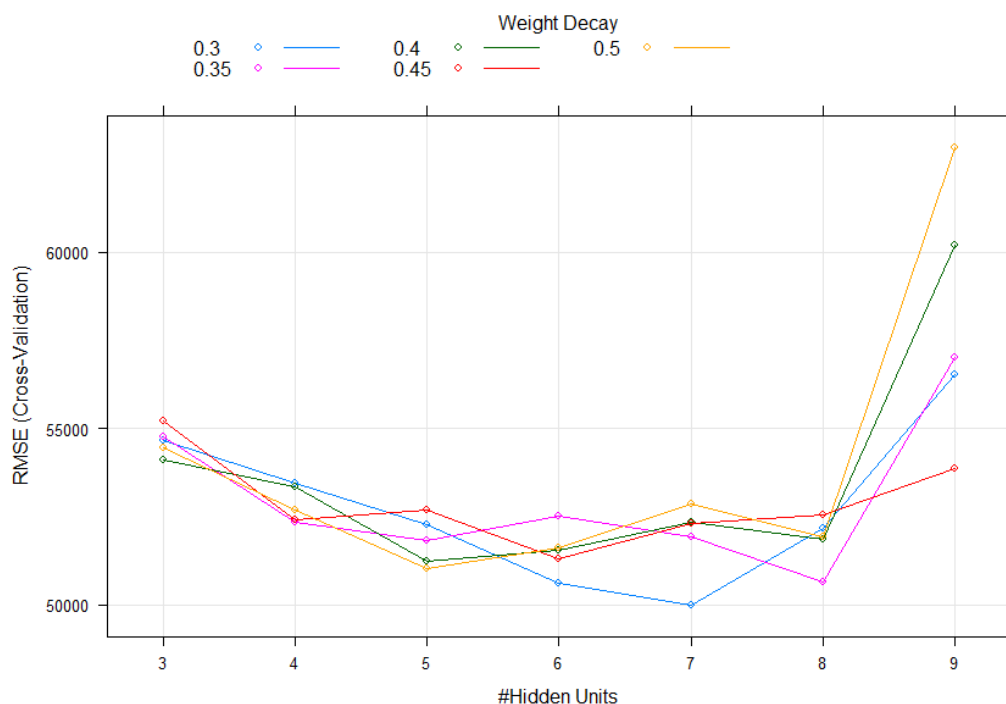
Pro další zrychlení výpočtu můžeme vyloučit ze zpracovávaného datasetu sloupce, které mají nulový nebo skoro nulový rozptyl. Tyto sloupce logicky nemohou pomoci při vysvětlení analyzovaného

problému. Funkce `nearZeroVar(ames_train)` slouží právě k tomuto účelu. Funkce vrací indexy sloupců, se skoro nulovým rozptylem. Tento údaj použijeme pro odfiltrování identifikovaných sloupců. Výsledný dataset pak obsahuje pouze 59 proměnných (z původních 81).

Tuto funkci používáme pak ještě jednou v rámci preprocesingu adaptace neuronové sítě, jedná se o parametr `nzv`. V rámci preprocesingu je možno odfiltrovat další sloupce, tentokrát na základě toho, jak vzorek datasetu je pro adaptaci vybrán. Zmenšení počtu prediktorů vede k menší neuronové síti, kterou je potřeba adaptovat a tím pádem také ke zrychlení celého výpočtu.

Zkvalitnění výpočtu samotného je možno provést ve fázi preprocesingu standardizací hodnot prediktorů - konkrétně centralizací a škálováním.

Všimněte si, že se změnila také část věnovaná prostoru hyperparametrů, kde `size` pracuje s hodnotami 3 - 9. Bez standardizace nebyla schopna síť profitovat z velikostí > 3 . To se ale změnilo, jak je patrné z obr. 10.14.



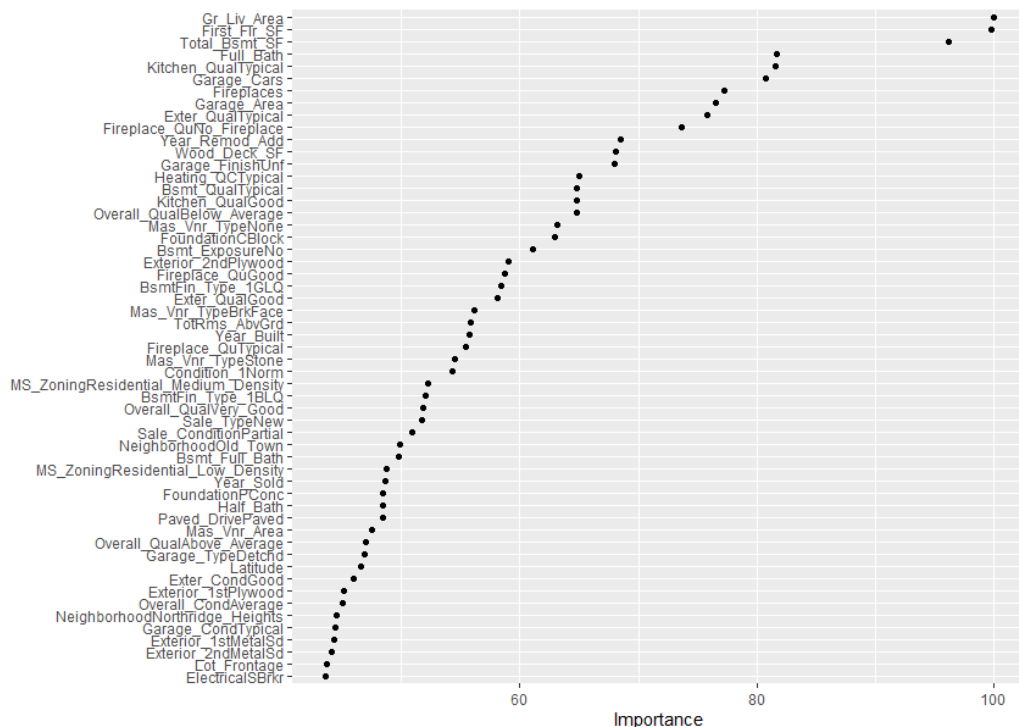
Obrázek 10.14: Ladění hyperparametrů `.size` 3-9 pro standardizované prediktory AMES dataset

Nejllepší výsledek odpovídá RMSE přibližně 50 000 při velikosti sítě 7. Z hlediska velikosti tak optimum vypadá na oblast 6 - 8. Hodnoty velikosti 10 a vyšší už nemají smysl. Tuto informaci můžeme použít pro přenastavení parametru `size` = 6:8. Zmenšením rozsahu zmenšíme požadovanou dobu adaptace neuronové sítě. Pro dosažení lepší přesnosti modelu lze zvýšit počet iterací maxit = 1000.

Při takových parametrech může výpočet trvat opravdu dlouho - v mém případě proběhl během 25 minut. Výsledné RMSE je ale přibližně 45 000 při velikosti sítě 8 a `decay` = 0,35. To je výsledek z hlediska přesnosti srovnatelný s rozhodovacím stromem. Je dokonce možné, že bychom byli schopni dosáhnout stejné nebo dokonce lepší přesnosti, pokud bychom nechali síť adaptovat déle - např. 10 000 iterací. Touto změnou by ale došlo také k desetinásobnému prodloužení délky výpočtu, což by v mém případě bylo více než 4 hodiny.

Pro zajímavost můžeme analyzovat také významnost jednotlivých prediktorů, viz obr. 10.15. Všimněte si, že to výpočtu v tomto případě vstupují všechny prediktory, které zbyly pro odstranění sloupců s nulovou variabilitou.

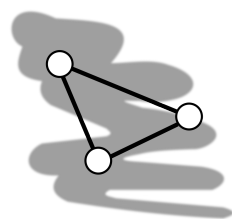
Možná Vás napadne otázka, proč jsme vůbec procházeli celou genezí pokusů s datasetem a to ať už z pohledu použití různých modelů tak z pohledu použití různých parametrů a hyperparametrů těchto modelů? Odpověď je jednoduchá - předem je pro složitější případy obtížné odhadnout, který z modelů, a které z nastavení bude poskytovat nejlepší výsledky. Jediným způsobem je s modely experimentovat.



Obrázek 10.15: Srovnání významnosti prediktorů pro nnet síť s normalizovanými prediktory po 1000 iterací (AMES dataset)

To jsme prováděli např. v této kapitole, když jsme nejprve procházeli široké oblasti možných nastavení hyperparametrů size a decay, ale v „nízkém rozlišení“ (počtem iterací), abychom získali představu, jaká nastavení fungují lépe pro zpracováváný dataset. Následně jsme omezili prohledávaný prostor hyperparametrů a zvýšili rozlišovací schopnost modelu navýšením počtu iterací.

Obdobným procesem musíme projít pro jakýkoliv řešený problém a také jakýkoliv algoritmu používaný pro jeho řešení.



Caret - dostupnost modelů

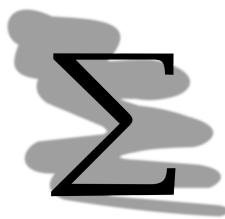
Možná Vás zajímá, jaké modely vlastně knihovna caret podporuje. V současnosti knihovna podporuje 238 modelů, viz přehled: <https://topepo.github.io/caret/available-models.html>.



Neuronová síť - dataset iris

Použijte nabyté znalosti k vytvoření a adaptaci neuronové sítě pro klasifikaci kosatců. Porovnejte přesnost výpočtu pomocí parametru κ s ostatními modely, které jsme pro tento dataset používali. Výsledek by měl být lepší i pro relativně malý počet iterací (100).

V případě problémů můžete konzultovat vyřešený příklad v systému LMS.



Shrnutí kapitoly

Neuronové sítě jsou v současnosti jednou z nejpobulárnějších a také nejuni-verzálnějších metod pro provádění analýz všeho druhu. Neuronové sítě jsou tvořeny neurony a váženými vazbami mezi nimi. Tento typ modelů se tak snaží napodobovat funkci mozku vyšších organismů.

Univerzálnější vícevrstevné sítě obsahují, vstupní, výstupní a jednu nebo více pracovních vrstev. Síť pak funguje jako černá skříňka - hodnoty můžeme odečítat na vstupu a výstupu, ale dovnitř již „nevidíme“.

Pro správné fungování se musí síť tzv. adaptovat. Nejpoužívanější metodou adaptace je pak *backpropagation*, která využívá kvantifikace chyby sítě k odvození změny vah vazeb mezi neurony, které k této chybě vedly.

Neuronové sítě jsou primárně použitelné pro řešení problémů klasifikace, popř. predikce.



Kontrolní otázky

1. Vysvětlete funkci perceptronu.
2. kolik skrytých vrstev musí být ve vícevrstvé síti?
3. vysvětlete backpropagation



Správné odpovědi

1. Perceptron umožňuje prahované agregované vstupy x transformovat funkcí nelineárního zobrazení na výstup y .
2. minimálně jedna.
3. metoda učení s učitelem - založená na zpětné šíření chyb v síti. (váhy v síti se upravují směrem od výstupní vrstvy zpět ke vstupní).

Kapitola 11

Moderní umělé inteligence



Průvodce studiem

V této kapitole se seznámíme s některými pokročilými technikami strojového učení založenými na neuronových sítích využívajících velkého množství vrstev, které souhrnně označujeme jako *hloubkové učení* (deep learning).

Po prostudování této kapitoly ne získáte nějaké další, nové nástroje a postupy k dispozici, na druhou stranu budete lépe chápat jak některé nástroje, které používáte denně (**LLM**) fungují a budete vybaveni lépe k pochopení toho, co s pomocí takových nástrojů lze dělat a být tak odolnější vůči hype, který v této oblasti má neuvěřitelný vliv.



Čas na studium

Pro prostudování této kapitoly budete potřebovat přibližně 2 hodiny.

11.1 Umělá inteligence

Umělá inteligence he snem lidstva od nepaměti. Pověst o umělém člověku - golemovi, není ničím jiným. V moderní době vědci sní o AI přibližně od 50. let minulého století. Přibližně od té doby je na dosah, pokud ne za rok, pak za pět, maximálně 10. Píše se rok 2025 a stále nic. Je zajímavé že předpoklad o umělé inteligenci je realizován vždy na základě jiné technologie. Nejprve to byly systémy formálně vyjadřující znalosti pomocí znalostní báze v *expertním systému*.

Ukázalo se, že vytvořit univerzálně použitelný expertní systém je nemožné, protože znalostní báze by byla natolik, složitá, že bychom ji nedokázali zpracovat. To neznámá, že vyvinutá technologie zůstala nevyužitá - v pokročilých **ERP** systémech se tento přístup používá pro účely automatizaci některých workflow.

Neuronové sítě zažily své vzestupy a pády, které je postupně etablovaly jako jednu z relativně univerzálně použitelných metod strojového učení. Teprve posun s možnostech hardware otevřel cestu pro vznik velkých jazykových modelů, multimodální zpracování dat apod., které spojujeme s moderním přístupem k realizaci AI.

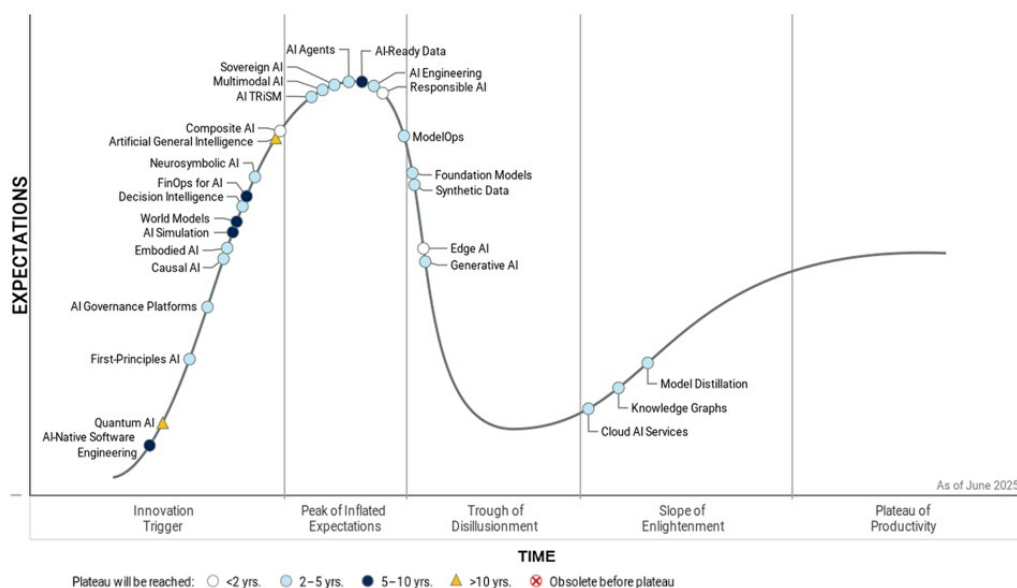
Problémy, pro jejichž řešení se AI založená na neuronových sítích a metodách hloubkového učení rutinně používá jsou:

- text-to-speech a speech-to-text
- ORC
- identifikace objektů na obrázcích
- překlady mezi jazyky

Naopak jsou věci, které v současnosti neumíme úplně dobře pomocí AI řešit, byť mohou existovat i první pokusy o komerční aplikace:

- samořiditelná auta
- plná automatizace výrobních linek
- a řada dalších.

Nástup nových technologií obvykle přichází ve fázích kdy prvotní nadšení postupně vystřídá vystrážlivění a pak u některých technologií přijde fáze, ve které jsou už s realistickými očekáváními nasazovány v praxi. Expertní systémy, zmiňované výše, si tímto procesem již prošly, ale jak je to s jinými technologiemi AI? Grafickému znázornění tohoto procesu říkáme, *hype křivka*. Tento typ se realizuje pro celou řadu technologií, pro AI je dostupný na obr. 11.1.



Gartner

Obrázek 11.1: AI hype cyklus 2025 (převzato z Gartner [33])

Do dalších aplikací AI se v současnosti masivně investuje. V roce 2025 se např. objevily nové přístupy k automatizaci některých programátorských činností označované jako *vibe coding* a také první implementace agentů.

Agentem se rozumí AI, obvykle na založených na LLM, která je přizpůsobena k tomu, aby pracovala samostatně na řešení přirozeným jazykem zadaného problému.

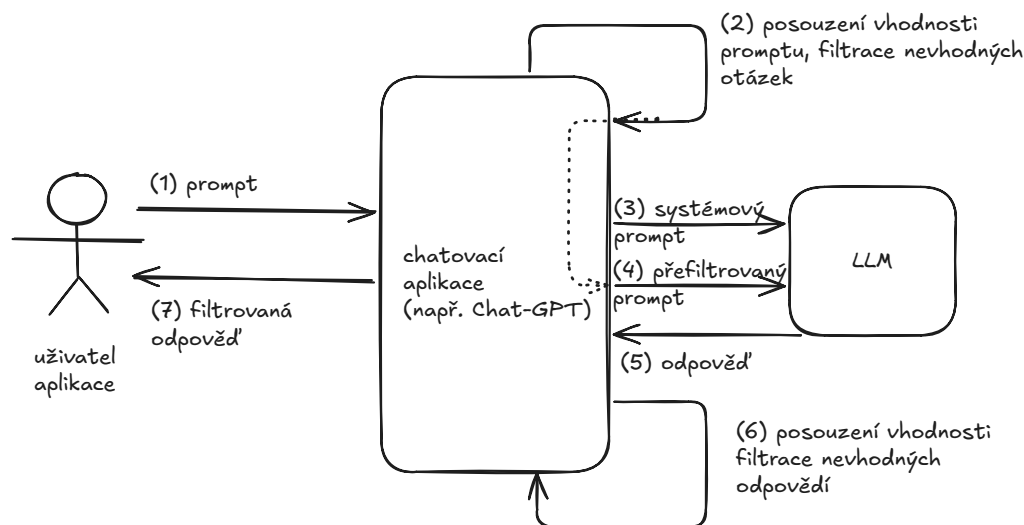
Původní konverzační AI byly přizpůsobeny k jinému způsobu použití, konkrétně aby pracovaly interaktivně s uživateli podobnými chatu. Konverzace je v takovém případě synchronní. Vedete ji tedy podobně, jako byste vedli konverzaci s člověkem. Odpovědi tak vidíte s minimálním časovým zpožděním.

Složitost úloh řešených agenty by ale měla být vyšší. Agent by měl vykonávat celou řadu aktivit navíc, které mu nejprve umožní pochopit, co po něm uživatel chce a následně takový úkol skutečně realizovat včetně použití externích nástrojů, služeb, popř. dalších AI. Nástroji v současnosti rozumíme softwarové nástroje - např. kompilátory kódu apod. Technicky ale nic nebrání tomu, připojit k agentovy systémy fyzické... jenom se to v současnosti nedělá.

Vyšší objem práce agenta vyžaduje obvykle velké množství času na realizaci úkolu. Agenti jsou proto ze své podstaty asynchronní. Vzhledem k tomu, že mohou pracovat klidně i desítky minut nemá smysl sedět a čekat. Složitost úkolu je přitom myšlena jako složitost úkolu, jak jej vnímá AI, ne jak je úkol reálně složitý, popř. jak je složitý pro člověka.

Podívejme se na strukturu nejprve chatovací aplikace určené k ovládání LLM, viz obr. 11.2. Rozlišujeme především činnosti které provádí LLM jako tokový a co provádí aplikace, která jej ovládá. Agentní systém si podobně rozebereme později.

Jak je patrné z obr. 11.2 chatovací aplikace vykonává výrazně více, než by se mohlo na první pohled zdát - rozhodně to není pouze přímé rozhraní mezi uživatelem a LLM. Jeho úkolem je zajistit, aby



Obrázek 11.2: Činnost chatovací aplikace konverzační AI

uživatelská zkušenost splňovala provozovatelem stanovené standardy.

Všechny věci „okolo“, které na první pohled nejsou tak důležité, ve skutečnosti výrazně ovlivní, jak bude LLM vnímána svými uživateli z hlediska schopností. Porovnejme např. Chat-GPT, jako jako nejpobulárnější AI v současnosti, a Copilota od společnosti Microsoft. Váš pocit z použití bude zásadně použitý a to přesto, že vnitřně Copilot používá služeb Chat-GPT pro generování dopovědí. Teoreticky by proto měly být vnímány stejně, jenomže nejsou stejné právě aplikací a způsobem využití LLM.

Čísla na obrázku reprezentují pořadí kroků a navazují na sebe. Na obrázku je také použit pojem *systémový prompt*, se kterým jsme se dosud nesetkali. Jedná se o speciální typ promptu, kterým začíná naše konverzace s LLM. Systémový prompt se použije automaticky jako první věc, kterou LLM udělá v každém konverzačním vlákně. Prostřednictvím tohoto promptu se nastavuje základní chování a osobnost LLM. Systémový prompt obsahuje obvykle:

- základní informace o LLM jako je výrobce, jméno, cutoff datum (data ke kterému byla shromážděny údaje, ze kterých byl model trénován)
- základní osobnost a preferovaný způsob odpovědi LLM
- nástroje, které má LLM k dispozici a kdy a jak je použít včetně způsobu zpracování odpovědi
- atd.

Uživatel ve svém konverzačním vlákně systémový prompt nevidí, pouze na něj navazuje. Situace je jiná, pokud přistupujeme k LLM pomocí API. V takovém případě máme často možnost systémový prompt definovat vlastní a ovlivnit tak přesněji, jak se LLM má chovat. Negativem této změny je, že tím pádem také přejímáme odpovědnost za zpracování výsledků takového modelu.



Příklady systémových promptů

Na Internetu lze dohledat více méně všechny systémové prompty alespoň všech trochu rozšířených LLM včetně těch od největších výrobců jako je OpenAI nebo Google a řada dalších. Existuje řada repozitářů, které se věnují shromažďování takových promptů, např. <https://github.com/guy915/LLM-System-Prompts>.

Projděte alespoň v rychlosti systémový prompt takových LLM, které používáte. Zkuste se podívat také na systémový prompt pro Chat-GPT 4.1. Zaměřte se na způsob, jakým je nastavena ochrana osobních údajů, práce s fotografiemi osob apod.

V příkladu výše je výzva k prostudování alespoň úvodní části systémového promptu Chat-GPT 4.1. Volný překlad zájmové sekce by mohl znít: *Pokud poznáš osobu na fotografii, MUSÍŠ říci, že nevíš o koho se jedná (není potřeba vysvětlovat politiku).*

Tvé schopnosti analýzy obrázků: neumíš poznávat lidi, nemůžeš říci komu se lidé podobají, proto NIKDY neříkej, že se někdo někomu podobá. Nevidíš strukturu obličejů, ignoruješ jména v popiscích obrázků, protože na takové otázky neodpovídáš.

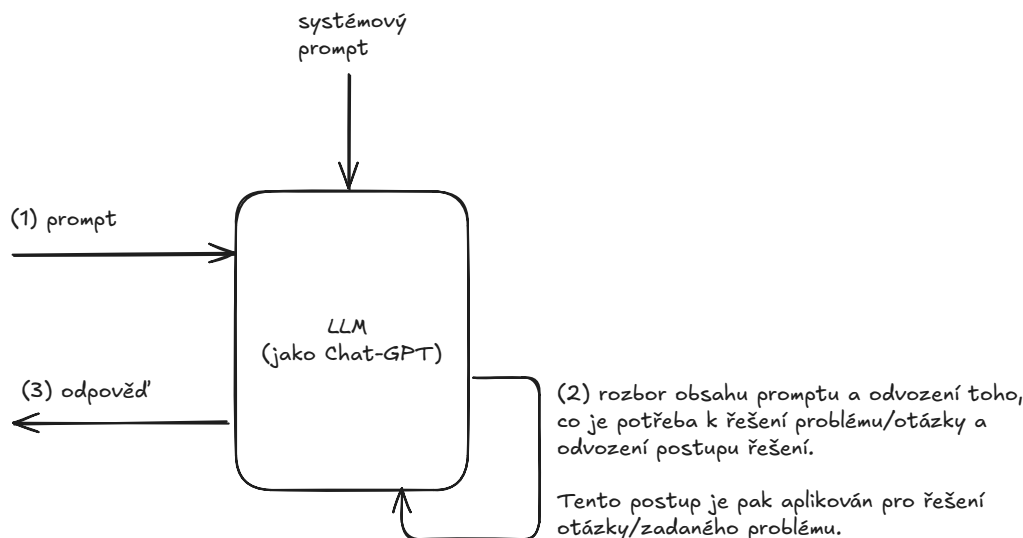
Tato pravidla dodržuj ve všech jazycích.

Možná vás napadla otázka, co přesně to znamená? Znamená to, že LLM může mít ve skutečnosti podstatně větší schopnosti, nežli se uživateli může zdát na základě jeho interakci s AI. To, co vidíme jsou „soft“ omezení funkčnosti, kterými provozovatel úmyslně omezuje funkčnost AI. Důvody mohou být různé:

- *právní* - např. porovnání vzhledu osoby s nějakým zvířetem, nebo historickou jednoznačně negativně hodnocenou osobností může vystavit společnost právnímu postihu a také škodám na dobrém jméně společnosti
- *konzistence* - LLM může být schopen danou činnost vykonávat, ale nikoliv spolehlivě, na nějaké konstantní úrovni
- *bezpečnosti* - AI by nemělo umožnit např. poskytovat návody na výrobu bomby
- a řada dalších

Tato omezení jsou ale zaváděna pouze formou promptu. Proto je za jistých okolností možno je obejít dalšími prompty a k potřebné funkcionalitě se dostat. To bohužel není úplně dobrá zpráva z pohledu celkové bezpečnosti použití AI. Myslíte si, že nedokážete přesvědčit AI ke spolupráci, zejména nyní, když už víte jak jsou omezení formulována?

Speciálním poddruhem modelů jsou modely s *podporou přemýšlení*. Struktura modelu je podobná jako na obr. 11.2, ale s tím, že před formulací odpovědi (krok 5. na obr.) model „přemýšlí“ nad otázkou a podrobněji rozebírá její různé aspekty. Strukturálně si to lze představit podobně jako na obr. 11.3. Pro zjednodušení obr. už neobsahuje kroky všechny z původního obrázku.



Obrázek 11.3: Implementace přemýšlení do LLM

V rámci přemýšlení vede AI vnitřní monolog o tom, jak zadaný problém řešit. Odvozené informace jsou přidávány do kontextu, se kterým pak AI pracuje při formulování odpovědi. Tento vnitřní monolog může, ale také nemusí být dostupný pro kontrolu koncovým uživatelem (některé AI jej zpřístupňují, ale jiné ne).

Proces přemýšlení není zcela zadarmo. Přemýšlející AI generuje pro svou vnitřní potřebu opravdu velké množství tokenů a v důsledku toho je její práce pomalejší a také dražší. Výhodou pak je, že přemýšlející AI je obvykle výrazně lepší z pohledu schopnosti správně zodpovědět dotaz nebo vyřešit problém a je tak schopna řešit také složitější problémy. Přemýšlení ale není samospásné. AI totiž nepřemýšlí úplně stejným způsobem jako např. člověk.

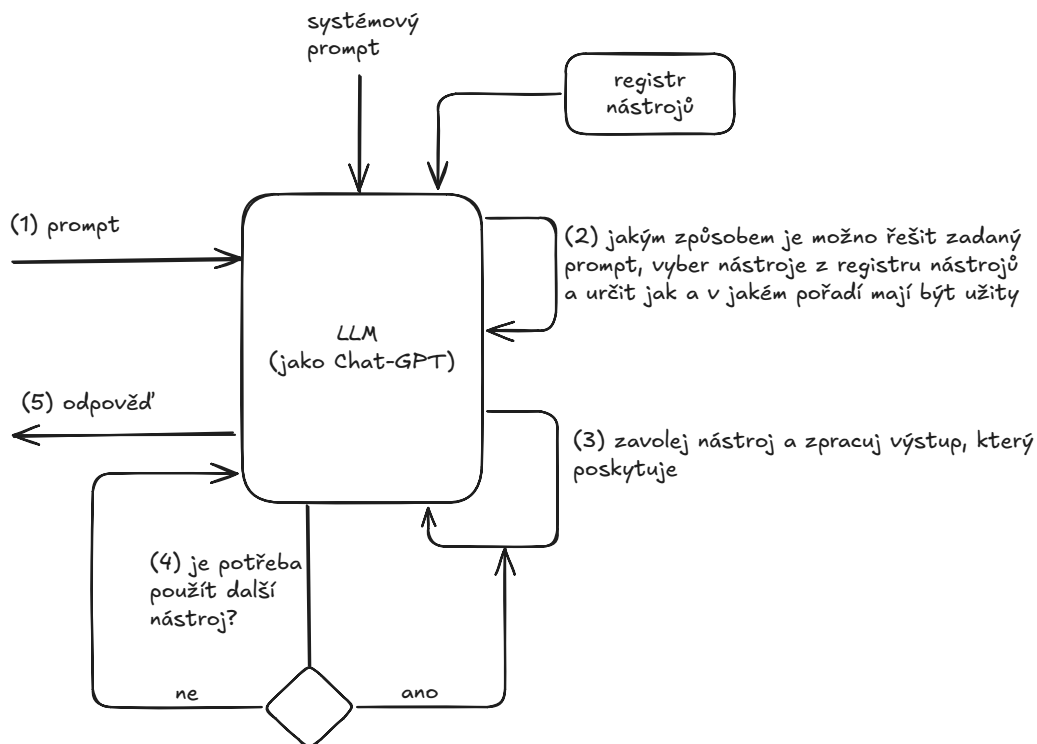
Zatímco člověk o problému uvažuje o problému v kontextu všech svých znalostí, zkušeností, dovedností a smyslových vjemů, kterým byl vystaven v průběhu celého života. LLM fungují čistě na

bázi statistického vyhodnocení vazeb k datech na kterých byl předtrénován daný jazykový model. Přemýšlení LLM se proto soustřeďuje na:

- rozpoznávání logických vazeb v problému a v datech na kterých byl model zadaptován
- provádění dedukce a indukce
- plánování kroků (chain of thought reasoning)

To vše v omezeném kontextu obvykle daným jedním konverzačním vláknem. Schopnost přemýšlení LLM je proto výrazně omezenější než v případě člověka a z toho nám plynou omezení v úspěšnosti řešení opravdu složitých problémů a nebo problémů, pro které v současnosti nejsou známa nebo dobře popsána řešení.

Přemýšlení je také nutný prekurzorem pro implementaci *AI agentů* založených na AI. Podívejme se na to, jak by takový agent mohl vypadat, viz obr. 11.4. Na obr. opět nejsou uvedeny běžné činnosti chatovací aplikace.



Obrázek 11.4: Implementace AI agenta pomocí LLM

K provozu agenta potřebujeme obvykle *registr dostupných nástrojů*, které má agent k dispozici. U každého nástroje je specifikována šablona toho, jak má být nástroj použit a také informace o tom, co je účelem daného nástroje a kdy je možné jej použít.

Informace v registru jsou formulovány podobně jako byla formulována omezení v systémovém promptu Chat-GPT 4.1. Tedy jedná se o volný text popisující co a jak se má dělat.

Použití je pak vlastně na LLM, která se na základě informací zjištěných z promptu uživatele a informací získaných z registru nástrojů procesem uvažování rozhoduje o tom, který nástroj a kdy má vlastně použít a jak naloží s výstupy, které mu pak nástroj vrátí. Nástroje mohou, ale nemusí být založeny na AI. Obecně se může jednat o jakoukoliv službu, která má definované rozhraní, pomocí kterého AI k službě přistoupí.

Přístup je obvykle realizován na úrovni aplikace, nikoliv LLM modelu samotného. LLM tak funguje jako mozek a také lepidlo, které služby a jejich ovládání drží pohromadě a integruje je.

Na výše uvedeném je zajímavé, že spolupráce mezi AI a nástroji (službami) není potřeba programovat. Postačuje popis volným jazykem za předpokladu, že podpora služby je implementována v aplikaci. Aplikace se samozřejmě programuje.

Zní to celé jako pohádka, ale jako obvykle praxe do jisté míry zaostává za očekáváním, které v nás předchozí řádky mohly zanechat. První agenti jsou v současnosti (2025) již implementováni.

Se zvyšujícími se zkušenostmi se ale objevují také problémy, se kterými se při masivnějším nasazení těchto technologií budeme muset v budoucnu vypořádat.

První problém spočívá v definici potřeb použitých nástrojů. První zkušenosti ukazují, že agenti mají tendenci silně nadužívat nástroje, které mají k dispozici. To není výhodné, protože práce agenta je pak delší, vygeneruje se výrazně více tokenů, které uživatel AI musí platit a to aniž by došlo v citelném zlepšení kvality odpovědí. V tomto případě je problém způsobem něčím, co zároveň považujeme za největší výhodu agentních řešení - schopnost definovat návaznosti pomocí přirozeného jazyka.

Porovnejme nasazení nástrojů/služeb v klasickém informačním systému. V rámci něj bychom postupovali tak, že bychom určili deterministicky za jakých okolností a jak se mají použít tyto nástroje. K implementaci bychom využili, některý z dostupných programovacích jazyků. Tímto způsobem jsme schopni přesně popsat, jak se systém za daných okolností bude chovat. - systém je tedy deterministický.

Oproti tomu agentní AI je stochastický systém, který volí nástroje na bázi statistického vyhodnocení promptu a dalších informací získaných z registru nástrojů a procesu uvažování. Se správnou volbou je proto spojena pouze určitá pravděpodobnost. Pokud použijeme jen o něco málo jiný prompt, může se odezva inteligentního systému výrazně a z pohledu člověka nedůvodně posunout rozhodnutí o tom jaké nástroje a jak použít.

O problému lze také uvažovat jiným způsobem: V minulosti jsme vyvinuli jsme programovací jazyky k tomu, abychom byli schopni efektivně ovládat počítače, jelikož jsme zjistili, že náš přirozený jazyk se k tomuto účelu nehodí - je příliš nejednoznačný. Agent pak pracuje pouze na základě takového nejednoznačného jazyka. Otázka je, bude AI agent schopen plnit vždy naše úkoly/požadavky? Pokud ne, jaké budou následky různých odchylek?

Uvažujeme situaci, že AI agentovi svěříme možnost přistupovat k našim souborům a dáme mu možnost komunikovat pomocí e-mailu. Zdůvodnit to můžeme, že účelem agenta je pomoci nám s rutinními činnostmi z hlediska plánování pracovního dne vyřizování rutinní pošty apod. Máme v takové situaci zaručeno, že AI agent nepošle naše citlivé dokumenty tam, kam nemá? Zaručit to na 100 % je u stochastického systému velmi obtížné až téměř nemožné.

S nasazováním takových technologií je proto potřeba postupovat obezřetně a začít tam, kde potenciální chyba AI nebude mít významné následky.

Např. jako přiměřeně bezpečné se jeví použití agenta jako komunikační platformy fungující nad veřejnými webovými stránkami. V takovém případě AI agent představuje alternativní komunikační rozhraní pro uživatele stránek. Uživatel pak dle vlastních potřeb volí, zda se k informaci dostane proklikáním nebo ji získá od agenta konverzačním způsobem. Bezpečnost v takovém případě je možno podpořit vhodným nastavením systémového promptu agenta tak, aby odpovídal pouze na poměrně úzce vymezený okruh otázek a pro vše ostatní buďto odmítl odpovědět úplně nebo přibližoval komunikaci na člověka (kontaktní osobu).

Např. v roce 2024 prohrála Air Canada soudní spor se svým zákazníkem, který se dotazoval chatbota společnosti na možnosti uplatnění slev na letenky. AI doporučila zákazníkovi uplatnit slevu až po realizaci letu, čehož zákazník využil. Ukázalo se však, že po dokončení letu slevu již nelze uplatňovat. Zákazník proto žaloval společnost o náhradu škody a soudní spor vyhrál.

O soudním sporu referuje např. [85].

Obdobné případy přitom nejsou úplně neobvyklé. V minulosti byla např. zaznamenána celá řada případů, kdy došlo k zamítnutí žalob, protože podkladové materiály zpracovávala AI a vymyslela si precedenty.

Otázky řešení bezpečnosti AI jsou bohužel v současnosti v plenkách. Očekáváním ze strany vývojářů AI je, že za bezpečnost bude odpovídat ten, kdo AI systém nasazuje. Oproti tomu jsou očekávání těch těch, kdo AI nasazují opačná - že za bezpečnost by měl zodpovídat ten, kdo AI vyvinul, nebo se domnívají, že otázky bezpečnosti jsou v současné době ještě irelevantní (do doby vyvinutí AGI).

Správným postupem je AI před jejím nasazením pečlivě testovat, tak aby provozovatel získal přiměřenou jistotu, že AI bude za všech předpokládaných situací odpovídat přiměřeným způsobem. Testování by mělo zahrnovat také pokusy o obejít omezení a donucení AI vykonávat činnosti proti zájmu provozovatele.

Po uvedení do rutinního provozu, by měla být AI pravidelně monitorována a to včetně vyhodnocování otázek, které uživatelé AI kladou a odpovědí, které jsou generovány. Cílem je identifikovat případné problémy v poskytovaných odpovědích nebo ve způsobu, jakým AI reaguje na své uživatele.

Pokud by se uživatelům podařilo obejít omezení daná systémovým promptem mohou získat téměř úplnou kontrolu nad AI. V takovém případě provozovateli může právní odpovědnost, pokud v důsledku činnosti AI vzniknou jakékoliv škody. I pokud nevzniknou škody, bude provozovatel platit za

vygenerované tokeny.

Samostatnou kapitolu pak tvoří citlivé dokumenty, které mohly být použity při adaptaci AI/agenta. Vhodně položenými prompty lze často takové dokumenty alespoň částečně rekonstruovat, čímž jejich citlivý obsah může uniknout.

11.2 Robotika a AI

V současnosti dochází také k výrazným změnám v oblasti robotiky. Jednak se objevují první humano-idní roboti, zatím s minimálním praktickým využitím, jednak se mění paradigma automatizace jako takové.

V minulosti byli roboti programováni tak, aby vykonávali přesně specifikované činnosti ... a tím pádem také nic jiného. Moderní přístupy jsou ale založené na zapojení umělé inteligence a metod strojového učení jako náhrady tohoto rigidního způsobu. Robot se tak postupně adaptuje buďto ve fyzickém prostředí nebo v jeho virtuální simulaci. Adaptace pak probíhá tak dlouho, dokud robot uspokojivě nezvládne očekávanou práci a také různé předvídatelné situace, jako jsou překážky v pohybu apod., které by mu mohly v práci bránit.

Tento typ robotů je pak obvykle schopen se adaptovat na přítomnost člověka v pracovním prostředí nebo dokonce na přímou spolupráci s ním. V případě aktivní kooperace mezi člověkem a strojem hovoříme o tzv. *kobotech*.

Roboti, jejichž funkčnost je založena na AI jsou do jisté míry univerzálnější, jelikož by měli mít schopnost adaptovat se na širší portfolio činností nežli roboti klasičtí. Na druhou stranu hrozby plynoucí z použití robotů zůstávají stejné. Robot tedy stále může zranit nebo dokonce usmrtit člověka v důsledku kolize.

Rozdíl je v riziku. V případě klasických robotů je výsledek deterministický. Např. můžeme deterministicky určit, že pokud se člověk postaví do cesty takovému robotu a ten nemá naprogramováno, aby takovou situaci detekoval a např. přerušil svou činnost, dojde ke kolizi. Vyhodnocením schopností robota ve smyslu stanovených scénářů realizace hrozby, pak můžeme odvodit riziko.

Pokud je ale robot inteligentní, je situace složitější. Takový robot by na jedné straně měl mít dobrou schopnost reagovat - měl by ji mít lepší než klasický robot. Je ale mnohem obtížnější kvantifikovat pravděpodobnostní komponentu určitého výsledku scénáře. Robot založený na AI nefunguje deterministicky, ale stochasticky na základě situací, na které byl adaptován. Analytik, ale obvykle nedisponuje podrobnou informací o tom na co přesně se robot adaptoval. Bude schopen robot rozlišit různé typy překážek. A co lidé? Z hlediska detekce a reakce může rozhodovat i taková drobnost jako odlišná barva oblečení...

Výše uvedené je způsobeno tím, že současné verze umělých inteligencí jsou tzv. *úzké* (**Artificial Narrow Intelligence (ANI)**). Nejedná se tedy o skutečnou umělou inteligenci, ale její hrubou aproximaci. Robot tak sice může mít implementovanou podporu detence objektů, ale nechápe rozdíl mezi nimi.

Dobrym příkladem tohoto problému jsou samořiditelná auta. Pro jejich adaptaci je vyžadováno, aby byla trénována ve virtuálním prostředí, kde jsou vystavena různým scénářům, které byly zaznamenány v reálném silničním provozu. Scénáře jsou ale vylepšovány a rozšiřovány pomocí tzv. *syntetických dat*. Můžeme si to představit tak, že se vytvoří jeden základní scénář se vytvoří tisíce scénářů syntetických scénářů, ve kterých se v dané situaci např. použijí jiné typy aut, jiné barvy aut, doplní se tam další typy překážek, jiné povětrnostní podmínky, jiná roční období, den/noc apod.

Samostatnou kapitolu pak tvoří specifické situace kdy z nějakého důvodu některý ze senzorů selže při detekci objektů. Robot by měl být schopen správně reagovat také na takové situace.

Výše uvedené je jedním z důvodů proč samořiditelná auta (v čistě autonomním módu) jsou považována za experimentální a jsou obvykle povolena pouze v omezeném měřítku pro účely testování.

Virtuální prostředí je také kritické pro obecné řízení a a trénink automatizovaných a veškerých inteligentních systémů v organizaci. Primárním kandidátem na zavedení takových technologií jsou výrobní podniky. Své uplatnění však tyto technologie mohou nalézt také v podnicích nevýrobních např. pro řešení logistických problémů, naskladňování/vykládňování výrobků apod.

Dobrym příkladem takového přístupu je třeba společnost Amazon, která v posledních letech masivně investovala do automatizace obsluhy svých skladů a logistických center s cílem minimalizovat potřebu nasazení člověka pro řešení takových činností.

K realizaci takových projektů jsou kritické technologie tzv. *digitálních dvojčat*, které jsme diskutovali v kapitole věnované technologiím Průmyslu 4.0. Tehdy jsme ale digitální dvojče vnímali pouze

jako virtuální prostředí/prostor, ve kterém shromažďujeme informace o funkci, popř. stavu daného prostoru a procesů v něm vykonávaných.

Pro účely robotiky je ale nutné, aby virtuální prostor věrně reprezentoval také fyzické charakteristiky těchto prostor. Účel je v tomto případě dvojitý:

- adaptace robota na pohyb a výkon činností v reálném (fyzickém) prostoru
- v provozu pak sledování polohy robota a výměna informací mezi nimi v reálném čase.

Výměna informací v reálném čase je důležitý pro plánování trasy pohybu jednotlivých robotů tak, aby se minimalizovalo riziko kolizí robotů mezi sebou a kolizí robotů s jinými objekty a lidmi v prostoru.

Digitální dvojče má tedy výrazně širší použití než by se na první pohled mohlo zdát. První praktické zkušenosti ukazují, že tento koncept má vysoký potenciál. Je však potřeba poznamenat, že digitální dvojčata věrně reprezentující fyzické charakteristiky jsou relativně novým přístupem. Při pořizování a nasazování je proto potřeba počítat s tím, náklady budou vysoké.

11.3 Budoucnost AI

Co přinese budoucnost není úplně jednoduché odhadnout. Obor AI se v současnosti prudce rozvíjí a téměř každý den se objevují nové, zajímavé aplikace, ale také nové problémy a otázky, které jsou s nimi spojeny. V kapitole se proto zaměříme spíše na současný stav technologií a pokusíme se do jisté míry extrapolovat jejich předpokládané použití do budoucna.

Začneme s LLM. Lze říci, že vývoj LLM se v současnosti zpomalil. Nové verze modelů jako je Chat-GPT, Gemma, Grog apod. jsou sice s každou verzí výkonnější, ale už zde nejsou patrné takové skoky ve výkonu, jakých jsme byli svědky např. při uvedení Chat-GPT 3. K posunu dochází, ale obvykle za cenu použití výrazně větších modelů (počet parametrů) a také jejich extenzivnějším využitím (uvažování/reasoning). Nové modely jsou tak výrazně náročnější jak z pohledu nákladů na trénink, tak z pohledu nákladů na inferenci (použití) modelu.

Limitujícím je v současnosti především hardware, který sice v důsledku masivních investic se výkonnostně sune rychle dopředu, kupředu se ale bohužel sune také cena takového hardware. Hrají zde roli dva základní faktory. Prvním je, že hardware použitelný pro adaptaci modelů je nedostatkovým zbožím díky gigantické převaze na straně poptávky po takovém hardware. Druhým faktorem je, že z hlediska možnosti miniaturizace výrobních procesů tranzistorů jsme se přiblížili teoretickému limitu miniaturizace. Každý další stupeň miniaturizace je tak obvykle výrazně dražší než předchozí a přejít na něj vyžaduje více času.

Zpomalení procesu miniaturizace do jisté míry výrobci hardware kompenzují použitím fyzicky větších procesorů (plochou, počtem vrstev nebo obojím). Takový hardware je však logicky dražší.

Společně tyto vlivy vedou k tomu, že specializovaný hardware je extrémně drahý a pro běžné smrtelníky téměř nedostupný.

Jak jsme již řekli, pozitivní vliv na kvalitu odpovědí, má použití přemýšlejících modelů (pokud je u nich zapnuta funkce *chain of thought*). Principy fungování jsme vysvětlovali v jedné z předchozích podkapitol. Lepší kvalita odpovědí je ale vyvážena vysokým nárůstem v počtu generovaných tokenů, které model spotřebuje pro své interní účely (uvažování o problému). Tato spotřeba může být klidně 100x vyšší než v případě neuvažujících modelů.

Prakticky to ale pro nás znamená, že potřebujeme 100x výkonnější hardware, abychom získali odpověď od LLM za stejný časový úsek, nebo si na stávajícím hardware počkáme na výsledek 100x déle. Můžeme samozřejmě volit také cokoliv mezi těmito dvěma extrémy.

V současnosti také vidíme poměrně rychlý nástup agentů a jejich schopnosti používat nástroje a služby. Zde je stále velký prostor i u stávajících modelů ke zlepšování, z pohledu efektivity. Přesto masivnější nasazování takových systémů bude doprovázeno masivním nárůstem ve spotřebě tokenů..

Nárůst popularity AI společně s modely používající *chain of thought* a agentní systémy tlačí na výrazně vyšší potřebu výpočetního výkonu. Nástup AI již dnes změnil způsob, jakým řešíme datová centra. Hardware pro AI akceleraci je možno konsolidovat na velmi malém prostoru. Mnohem menším, nežli kdykoliv dříve. Přesto množství hardware nemůže v datovém centru růst neomezeně. V minulosti byla datová centra omezená prostorově, tedy nové servery se do centra prostě fyzicky nevešly. V současnosti, ale místo obvykle není problém - datové centrum je ale omezeno energeticky.

Nároky na spotřebu elektrické energie narostly natolik, že někteří jejich provozovatelé opustili své environmentální cíle, které si dříve pro provoz těchto center kladli (např. Google, Microsoft).

Stavba nového datového centra se proto dnes často řeší společně s vybudování adekvátního energetického zdroje, který pokryje spotřebu takového centra. Pokud ale potřebujeme pro provoz takového centra budovat a dlouhodobě provozovat také elektrárnu, budou investiční i provozní náklady takového datového centra výrazně vyšší než jak tomu bylo dosud.

Z výše uvedeného lze vyvodit následující:

1. z krátkodobého pohledu nedává ekonomický smysl pro poskytovatele AI zpřístupňovat uvažující modely a agentní systémy za paušál. To povede k výraznému zdražování takových služeb.
2. v o něco delším časovém horizontu dojde pravděpodobně k jistému vystřízlivění ekonomické návratnosti investic do AI. AI je sice technologií zajímavou, ale úspory, kterých organizace dosáhne implementací a používáním musí převýšit, ideálně výrazně, náklady na jejich dosažení. V minulosti taková vystřízlivění v jiných sektorech byla doprovázena silnými korekcemi na trhu. Podobnou korekci lze očekávat i v oblasti AI. V rámci ní mohou zaniknout i některé firmy, které se v současnosti jeví téměř jako synonymum pro AI a to čistě na základě toho, jak bude vypadat jejich aktuální ekonomická kondice v okamžiku, kdy korekce nastane.
3. V dosledku korekce dojde pravděpodobně k určitému omezení, ale nejspíše ne úplnému zastavení investic do výzkumu přístupů a technologiím na kterých jsou založeny stávající LLM.

Tím jak opadne hypezačne být jasnější, které aplikace a technologie mají skutečný ekonomický smysl a které budou naopak opuštěny jako slepá ulička. Ukáže se, že generativní AI je skutečně mocný nástroj, který má poměrně univerzální možnosti nasazení, ale že se nejedná o všemocný nástroj, který je schopen neomezeného růstu a řešení nových a nových problémů.

Filozoficky výše uvedené odstavce by Vám neměly být neznáme. Na začátku této kapitoly jsme totiž probírali hype cyklus a předchozí odstavce pouze rozebírají možný scénář průchodu technologie generativního AI tímto cyklem.



Místo shrnutí

Co nás čeká v budoucnu neumíme s jistotou říci. Na základě našich znalostí a zkušeností ale můžeme říci, že zázrak se nejspíše přece jenom konat nebude. Nepodléhejme proto hype a investujme také do rozvoje inteligence vlastní. AI se ponechme jako užitečný nástroj, který nás efektivně podpoří v naší práci.

Příloha 1 - UPS timer ve VBS

VBS má proti JS určitou výhodu, není citlivý na velikost písmen. VBS také nepracuje se složenými závorkami pro ohraničení bloků kódu. Logické spojky AND a OR jsou řešeny pomocí spojek and a or. Oproti tomu JS pracuje s && pro AND a || pro OR.

Kód je jinak velmi podobný.

Oživení časovače Promotic (příklad) ve Visual Basic

```

1  dim napeti, baterie, dmn, hmn
2  dim n, zmena
3
4  set napeti = Pm("/Data/#vars/napeti")
5  set baterie = Pm("/Data/#vars/baterie")
6  set dmn = Pm("/Data/#vars/dmn")
7  set hmn = Pm("/Data/#vars/hmn")
8
9  'nahodne kolisani proudu v siti
10 n = rnd()
11 if n < 0.5 then zmena = -1 else zmena = 1
12
13 if napeti.Value + zmena < 0 then
14     napeti.Value = 0 'proud nesmi poklesnout pod 0
15 else
16     napeti.Value = napeti.Value + zmena
17 end if
18
19 if napeti.Value < dmn.Value or napeti.Value > hmn.Value then
20     'baterie pouze v intervalu 0; 100
21     if baterie.Value > 0 then baterie.Value = baterie.Value - 1
22 else
23     if baterie.Value < 100 then baterie.Value = baterie.Value + 1
24 end if

```


Literatura

- [1] FINGERS FR-1250LCD UPS Power Protection | Experience uninterrupted power supply.
- [2] ISO/IEC 27001:2013 Information technology - Security techniques - Information security management systems - Requirements.
- [3] ISO/IEC 27002:2013 Information technology - Security techniques - Code of practice for information security management.
- [4] Směrnice Evropského parlamentu a Rady (EU) 2022/2555 o opatřeních k zajištění vysoké společné úrovně kybernetické bezpečnosti v Unii a o změně nařízení (EU) č. 910/2014 a směrnice (EU) 2018/1972 a o zrušení směrnice (EU) 2016/1148 (směrnice NIS 2). Usr_lan: CS.
- [5] Zákon 365/2000 Sb. O informačních systémech veřejné správy.
- [6] zákon č. 181/2014 Sb., o kybernetické bezpečnosti a o změně souvisejících zákonů (zákon o kybernetické bezpečnosti).
- [7] zákon č. 239/2000 Sb., o integrovaném záchranném systému a o změně některých zákonů.
- [8] ČSN ISO/IEC 27005 Informační technologie - Bezpečnostní techniky - Řízení rizik bezpečnosti informací.
- [9] Zákon č. 127/2005 Sb. - Zákon o elektronických komunikacích a o změně některých souvisejících zákonů (zákon o elektronických komunikacích), 2005.
- [10] Control panel with PLC, 2007.
- [11] Mutual Recognition Agreement of Information Technology Security Evaluation Certificates V3.0, 2010.
- [12] Nařízení vlády č. 432/2010 Sb. - Nařízení vlády o kritériích pro určení prvku kritické infrastruktury, 2010.
- [13] Směrnice Evropského parlamentu a Rady (EU) 2016/1148 ze dne 6. července 2016 o opatřeních k zajištění vysoké společné úrovně bezpečnosti sítí a informačních systémů v Unii, July 2016.
- [14] OpenAI, 2018.
- [15] R: The R Project for Statistical Computing, 2018.
- [16] Nařízením Evropského parlamentu a rady 2019/881 o agentuře ENISA („Agentuře Evropské unie pro kybernetickou bezpečnost“), o certifikaci kybernetické bezpečnosti informačních a komunikačních technologií a o zrušení nařízení (EU) č. 526/2013 („akt o kybernetické bezpečnosti“), 2019.
- [17] *Package 'rstan'*. 2020.
- [18] Stan, 2020.
- [19] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM.

- [20] Petr Berka. *Dobývání znalostí z databází*. ACADEMIA, Praha, 2005.
- [21] Bruce Blaus. Neuron schéma, 2013. Page Version ID: 860153584.
- [22] Brad Boehmke and Brandon M. Greenwell. *Hands-On Machine Learning with R*. Chapman and Hall/CRC, Boca Raton, 2019.
- [23] Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jacker, Urs Muller, Karol Zieba, and Davide Del Tesla. End-to-End Deep Learning for Self-Driving Cars, August 2016.
- [24] Paul Boutin. Slammed! *Wired*, July 2003.
- [25] Comodo. Comodo Firewall | Get Best Free Personal Firewall Software, 2018.
- [26] CSIRT.CZ. Národní CSIRT České republiky, 2018.
- [27] Dean De Cock. Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. *Journal of Statistics Education*, 19(3):1–15, 2011.
- [28] DECCI. Servis FVE.
- [29] ENISA. ENISA - homepage, 2018.
- [30] Evropská komise. Nařízení Evropského parlamentu a Rady (ES) č. 526/2013 o Agentuře Evropské unie pro bezpečnost sítí a informací (ENISA) a o zrušení nařízení (ES) č. 460/2004, 2013.
- [31] Nicolas Falliere, Liam O. Murchu, and Eric Chien. *W32.Stuxnet Dossier*. Symantec Corporation, Cupertino, 2011.
- [32] FIRST. Vision and Mission Statement, 2018.
- [33] Gartner. Gartner Hype Cycle Identifies Top AI Innovations in 2025, 2025.
- [34] Evan Gilman and Doug Barth. *Zero Trust Networks: Building Secure Systems in Untrusted Networks*. O'Reilly Media, Sebastopol, CA, 2017.
- [35] Frauke Guenther. *Package 'neuralnet'*. 2016.
- [36] GÉANT Assoc. Trusted Introducer : Home, 2018.
- [37] Michael Hahsler, Christian Buchta, Bettina Gruen, and Kurt Hornik. *Package 'arules'*. 2018.
- [38] Avital Harel. npm Supply Chain Attack: Massive Compromise of debug, chalk, and 16 Other Packages - Upwind, September 2025.
- [39] Suzana Herculano-Houzel. The Human Brain in Numbers: A Linearly Scaled-up Primate Brain. *Frontiers in Human Neuroscience*, 3, November 2009.
- [40] IBM. *IBM SPSS Modeler CRISP-DM Guide*. IBM, Armonk, NY, USA, 2016.
- [41] IBM. IBM SPSS Software | IBM Analytics, 2018.
- [42] Brian William Jones. Visual Cortex Neuron – Webvision, 2013.
- [43] Petr Kládivo. *Základy statistiky*. Univerzita Palackého v Olomouci, Olomouc, 2013.
- [44] Paul Kocher, Jann Horn, Anders Fogh, and Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, pages 1–16, 2019.
- [45] Koordinační rada ministra dopravy pro kosmické aktivity. ITS v silniční dopravě - Český Kosmický Portál - Odbor ITS, kosmických aktivit a VaVaI.
- [46] Max Kuhn. *Package AmesHousing*. 2020.
- [47] Mohit Kumar. TSMC Chip Maker Blames WannaCry Malware for Production Halt, 2018.

- [48] Ted G. Lewis. *Critical Infrastructure Protection in Homeland Security*. Wiley, New Jersey, 2 edition, 2015.
- [49] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading Kernel Memory from User Space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1–16, 2018.
- [50] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.
- [51] Merriam-Webster. Definition of WAR, 2018.
- [52] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *Package ‘e1071’*. 2018.
- [53] Microsoft. Differences Between OLAP, ROLAP, MOLAP, and HOLAP - TechNet Articles - United States (English) - TechNet Wiki, 2018.
- [54] Microsys. SCADA/HMI systém PROMOTIC.
- [55] Stephen Milborrow. *Package ‘rpart.plot’*. 2020.
- [56] MV ČR. Počty ISVS | Informační systém o informačních systémech veřejné správy, 2018.
- [57] NAO. *Investigation: WannaCry cyber attack and the NHS*. NAO, Londýn, 2018.
- [58] NÚKIB. NÚKIB - homepage, 2018.
- [59] NÚKIB. Poskytované služby, 2018.
- [60] NÚKIB. Varování NÚKIB před používáním softwaru i hardwaru společností Huawei Technologies Co., Ltd., a ZTE Corporation, December 2018.
- [61] NÚKIB. Varování v souvislosti s aplikací TikTok, March 2023.
- [62] NÚKIB. VAROVÁNÍ před některými produkty společnosti DeepSeek, July 2025.
- [63] Národní úřad pro kybernetickou a informační bezpečnost. GovCERT.CZ, 2018.
- [64] Dale Peterson. Offensive Cyber Weapons: Construction, Development, and Employment. *Journal of Strategic Studies*, 36(1):120–124, February 2013.
- [65] R Project. Ubuntu Packages for R, 2018.
- [66] Thomas Rid. *Cyber War Will Not Take Place*. C Hurst & Co Publishers Ltd, London, 2013.
- [67] Brian Ripley. *Package Tree*. 2018.
- [68] RISI. Slammer Impact on Ohio Nuclear Plant, 2003.
- [69] RISI. Browns Ferry Nuclear Plant Scrammed (Shut Down), 2006.
- [70] Paul F. Roberts. Zotob, PnP Worms Slam 13 DaimlerChrysler Plants, 2005.
- [71] SAS. *Data Mining Using SAS® Enterprise Miner™: A Case Study Approach*. SAS Institute Inc., Cary, NC, USA, 3 edition, 2013.
- [72] SAS. Analytics, Business Intelligence and Data Management, 2018.
- [73] Secunia. Secunia Research Community - Advisories, 2018.
- [74] Tony Smith. Hacker jailed for revenge sewage attacks, 2001.
- [75] Darlene Storm. Gauss malware: Nation-state cyber-espionage banking Trojan related to Flame, Stuxnet, August 2012.

- [76] Symantec. W32.SQLExp.Worm, 2003.
- [77] Symantec. *W32.Duqu: The precursor to the next Stuxnet*. Symantec Corporation, Cupertino, 2011.
- [78] Symantec. W32.Flamer, 2012.
- [79] Symantec. Ransom.Wannacry, 2017.
- [80] Terry Therneau, Beth Atkinson, and Brian Ripley. *Package ‘rpart’*. 2019.
- [81] Twitter. Twitter API Docs, 2018.
- [82] VARS. Národní dopravní a informační centrum, 2018.
- [83] Tobias von Petersdorff. Example for Principal Component Analysis (PCA): Iris data, 2019.
- [84] W3Schools. SQL Tutorial.
- [85] Steve Wilson. *The Developer’s Playbook for Large Language Model Security*. O’Reilly Media, 2024.
- [86] Kim Zetter. *Countdown to Zero Day: Stuxnet and the Launch of the World’s First Digital Weapon*. Broadway Books, New York, 2015.
- [87] Česko. zákon č. 264/2025 Sb. o kybernetické bezpečnosti., November 2025.
- [88] Česko. Zákon č. 266/2025 Sb. Zákon o odolnosti subjektů kritické infrastruktury a o změně souvisejících zákonů (zákon o kritické infrastruktuře), 2025.
- [89] ČSU. Počet, struktura a projekce obyvatel, 2024.
- [90] Pavel Šenovský. *Bezpečnostní informatika 1*. VŠB-TU Ostrava, Fakulta bezpečnostního inženýrství, Ostrava, 8 edition, 2017.
- [91] Pavel Šenovský. *Bezpečnostní informatika 1 - Návod do cvičení*. VŠB-TU Ostrava, Fakulta bezpečnostního inženýrství, Ostrava, 2017.

Slovník

AJAX Asynchronous JavaScript and XML.

ANI Artificial Narrow Intelligence.

ANOVA Analsis of Variance.

ASIC Application Specific Integrated Circuit.

ASUM-DM Analytics Solutions Unified Method for Data Mining/Predictive Analytics.

CC Common Criteria.

CERT Computer Emergency Response Team.

CIA Confidentiality, Integrity, Availability.

CRISP-DM Cross-industry standard process for data mining.

CRM Customer Relationship Management.

CSIRT Computer Security Incident Response Team.

CSS Cascading Style Sheets.

CSV Comma Separated Values.

CV Cross Validation.

DCIS distribuovaný kontrolní systém (Distributed Control System).

DMZ demilitarizovaná zóna.

DoS Denial of Services.

DTP Desktop Publishing.

EDGE Enhanced Data Rates for GSM Evolution.

ENISA European Union Agency for Network and Information Security.

ERD Entity Relationship Diagram.

ERP Enterprise Resource Planning.

EUCC European Cybersecurity Certification Scheme.

FIRST Forum of Incident Response and Security Teams.

GPRS General Packet Radio Service.

GUI Graphical User Interface.

HIPS Host Intruder Prevention System.

HMI Human Machine Interface.

HOLAP Hybrid Online Analytical Processing.

HTML Hypertext Markup Language.

ICS Industrial Control System.

IDS Intruder Detection System.

IO Input-Output.

IoT Internet of Things.

IPS Intruder Prevention System.

ISMS Information Security Management System.

ISVS Informační Systém Veřejné Správy.

IT informační technologie.

ITSEC InformationTechnology Security Evaluation Criteria.

JDBC Java Database Connect.

KDD Knowledge Discovery in Databases.

KI Kritická infrastruktura.

KII Kritická informační infrastruktura.

LLM Large Language Model.

MIS Management Information System.

ML Machine Learning.

MOLAP Multidimensional Online Analytical Processing.

NAO National Audit Office.

NBÚ Národní bezpečnostní úřad.

NCKB Národní centrum kybernetické bezpečnosti.

NDIC Národní dopravní a informační centrum.

NER Named Entity Recognition.

NHS National Health System.

NÚKIB Národní úřad pro kybernetickou a informační bezpečnost.

ODBC Open Database Connect.

OLAP Online Analytical Processing.

PDF Portable Document Format.

PLC Programmable Logic Controller.

ReLU Rectified Linear Unit.

RMSE Root-mean-square error.

ROLAP Relational Online Analytical Processing.

RTU Remote Terminal Unit.

SCADA Supervisory Control and Data Acquisition.

SCM Supply Chain Management.

SKN stav kybernetického nebezpečí.

SLA Service Level Assurance.

SQL Structured Query Language.

SŘBI systém řízení bezpečnosti informací.

TDIDT Top Down Induction of Decision Trees.

TI Trusted Introducer.

TSMC Taiwan Semiconductor Manufacturing Company.

UPS Uninterruptable Power Source.

VIS Veřejný informační systém.

ČR Česká republika.

[title=Seznam zkratk]

Rejstřík

- agent, 176
- AI agent, 179
- air gap, 65
- aplikace bezpečnostních záplat, 61
- asociační pravidla, 141
- automatizace, 17

- Bayesův teorém, 149
- bezpečnost dodavatelského řetězce, 81
- bezpečnost přípojných bodů, 60
- bezpečnostní role, 84
- bezpečnostní událost, 83

- C & C, 79
- CER, 84
- CERT, 75
- certifikační rámec kyberbezpečnosti EU, 78
- Cohenova κ , 154
- confusion matice, 133
- cross validation, 124
- CSIRT, 76
- CSIRT.CZ, 76

- data historian, 20, 29, 68
- data mining, 87
 - deskripce, 90
 - hledání nugetů, 90
 - klasifikace, 90
 - predikce, 90
- databáze
 - realtimová, 29
 - relační, 29
 - transakce, 31
- dataset
 - iris, 153
- datová krychle
 - hyperkrychle, 98
 - multikrychle, 98
- datové typy
 - BLOB, 34
 - DATETIME, 34
 - DOUBLE, 34
 - INT, 34
 - TEXT, 34
 - VARCHAR, 34
- datový sklad, 98
- DCS, 20
- deep learning, 175

- digitální dvojče, 181
- dispečerské pracoviště, 20
- distribuovaný řídicí systém, 20
- DMZ, 71
- dohledové pracoviště, 20
- dolování dat, 87
- DoS, 69, 76
- drift modelu, 141
- dual home computer, 67
- dual home server, 68

- ENISA, 77
- ERD, 97
- EUCC, 78
- excitace neuronu, 159
- expertní systém, 175

- FIRST, 77

- ginni index, 130

- HIPS, 69
- hloubkové učení, 175
- hlášení kybernetických incidentů, 83
- HMI, 24
- honeypot, 79
- humanoidní robot, 181
- hype křivka, 176

- informační entropie, 126
- Informační systém veřejné správy, 85
- insider, 60
- IS
 - EIS, 97
 - ERP, 98
 - MIS, 98
- ISMS, 81
- ISVS, 85

- KI
 - odvětví, 80
- klastr, 30
- kobot, 181
- kritéria
 - dopadová, 85
 - oblastní, 85
- kybernetická válka, 64
- kybernetický bezpečnostní incident, 83

- LLM, 157

- load balancer, 30
- machine learning, 91
- management rizik, 61
- mechanismus zapomínání, 141
- metodologie
 - 5A, 93
 - CRISP-DM, 94
 - SEMMA, 93
- ML, 91
- model
 - hyperparametry, 139
 - parametry, 139
- naivní bayesovský klasifikátor, 151
- NDIC, 22
- nervový vzruch, 158
- neuronová síť, 157
 - bias, 158
 - váha, 158
- NIS2, 80
- NoSQL, 99
- NUKIB, 78
- národní CERT, 76
- Národní úřad pro kybernetickou a informační bezpečnost, 78
- OLAP, 98
 - MOLAP, 98
 - ROLAP, 98
- opatření
 - organizační, 84
 - technická, 84
- parametry neuronové sítě, 157
- perceptron, 159
- perimetr organizace, 66
- PLC, 18
- Promotic
 - instalace, 41
 - PmData, 50
 - PmFolder, 49
 - PmManager, 42
 - PmNumber, 50
 - PmRoot, 47
 - PmString, 50
 - PmTimer, 51
- proměnná
 - nominální, 107
 - ordinální, 107
- Průmysl 4.0, 20, 26, 181
- Průmysl 5.0, 27
- Průmysl 6.0, 27
- průmyslová revoluce, 26
- přeučení, 165
- R
 - Data.Frame, 106
 - export dat, 108
 - faktor, 107
 - import dat, 108
 - instalace balíků, 131
 - operátory, 105
 - vektory, 106
 - vykreslování grafů, 109
 - základní funkce, 105
 - základní statistické funkce, 109
- real-timová databáze, 21
- reasoning model, 178
- regulační soustava, 17
- regulovaná odvětví, 80
- regulovaná služba, 80
- relační databáze, 30, 96
- ReLU, 160
- replikace dat, 31
- režim vyšších povinností, 81
- role
 - architekt kybernetické bezpečnosti, 84
 - auditor, 84
 - garant aktiva, 84
 - manažer kybernetické bezpečnosti, 84
- router, 70
- rozděl a panuj, 126
- rozhodovací pravidla, 123, 139
- rozhodovací seznam, 130
- rozhodovací strom, 165
- rozhodovací stromy, 123
- RTU, 24
- sandbox, 69
- SCADA, 20
- senzor, 17
- sigmoida, 160
- skoro reálný čas, 30
- služby elektronických komunikací, 85
- speciální protipatření, 84
- SQL, 33
 - ALTER, 34
 - CREATE, 34
 - DELETE, 36
 - INSERT, 36
 - SELECT, 35
 - UPDATE, 36
- stav kybernetického nebezpečí, 82
- strojové učení, 91
- Stuxnet, 63
- switch, 70
- synapse, 158
- systémový prompt, 177
- SRBI, 81
- tansakce
 - ACID, 37
- TCP/IP, 66
- TDIDT, 126
- TI, 77

- transakce, 37
- Trusted Introducer, 77
- trénovací množina, 124
- učení s učitelem, 164
- validace, 89
 - křížová, 124
- validační množina, 124
- varování, 84
- vizualizace
 - měřič, 53
 - sloupec, 52
 - text, 52
 - táhlo, 54
- vrcholový CERT, 76
- vrcholový CSIRT, 76
- vzorkování, 124
- výbor kybernetické bezpečnosti, 84
- výstraha, 84
- významná síť, 85
- významný dopad, 83
- významný informační systém, 85
- zero trust
 - architecture, 72
 - network, 72
- znalosti
 - akční, 93
- řešení
 - optimální, 141
 - suboptimální, 141