

doc. Ing. Pavel Šenovský, Ph.D.

Bezpečnostní informatika 3

skripta
3. vydání



Bezpečnostní informatika 3

3. rozšířené vydání

tento text neprošel jazykovou úpravou

©Pavel Šenovský, Ostrava, 2018

Vysoká škola báňská - Technická univerzita Ostrava, Fakulta bezpečnostního inženýrství

Obsah

Seznam obrázků	6
Seznam tabulek	7
Úvod	9
1 Základy průmyslové automatizace	13
1.1 Automatizace	13
1.2 PLC	14
1.3 SCADA	16
1.4 Další zařízení využívána v automatizaci	17
2 Real-timeové databáze	23
2.1 Real-time databáze vs relační databáze	23
2.2 Transakce	25
2.2.1 Organizace dat v databázi	25
2.2.2 Jazyk SQL	26
2.2.3 Transakce	30
3 SCADA - příklad v Promotic	33
3.1 Promotic - instalace a projekt	33
3.2 UPS - specifikace podkladů pro tvorbu aplikace	40
3.3 Příprava obrazovek aplikace	42
3.4 Časovač - programování	45
4 Automatizace - otázky bezpečnosti	49
4.1 Případové studie bezpečnosti automatizace	50
4.2 Způsoby izolace automatizační sítě	54
5 Ochrana ICS	63
5.1 Typy CERT a CSIRT	63
5.2 Legislativa kybernetické bezpečnosti	67
6 Data mining	73
6.1 Postup data miningu	74
6.2 Metodiky dolování dat	77
6.3 Zdroje data miningu	80
7 Úvod do R	85
7.1 Instalace prostředí	85
7.2 RStudio	86
7.3 Základy jazyka R	88

8 Pravidlové metody	99
8.1 Příprava dat	99
8.2 Rozhodovací stromy	101
8.3 Rozhodovací pravidla	108
8.4 Asociační pravidla	108
9 Bayesovská klasifikace	115
10 Neuronové sítě	121
10.1 Neuron	122
10.2 Neuronové sítě a jejich adaptace	124
Literatura	135
Seznam zkratk	138
Rejstřík	139

Seznam obrázků

1.1	Obecné schéma regulační soustavy	14
1.2	Příklad PLC a řídicího systému (převzato z [10])	15
1.3	Zapojení SCADA systému, databáze, PLC a řízené technologie	17
1.4	Dohledové pracoviště Národní dopravní a informační centrum (NDIC) (převzato z [40])	18
1.5	Řešení monitoringu fotovoltaické elektrárny (převzato z [25])	19
1.6	Možná struktura webové aplikace zajišťující rozhraní člověk - stroj	20
2.1	Databázový klastr	25
2.2	ERD diagram metadat příspěvků v síti Twitter (adaptováno z [70])	26
2.3	Výsledky spojení dvou tabulek	29
2.4	Změna hodnoty transakce v čase	31
3.1	Základní rozložení aplikace Promotic - Nový objekt	35
3.2	Promotic - nastavení workspace s nástrojovou lištou	35
3.3	Promotic - vývojové prostředí	36
3.4	PmWorkspace - vlastnosti MainFrame	38
3.5	PmWorkspace - vlastnosti toolbar	38
3.6	PmWorkspace - vlastnosti Main	39
3.7	Promotic - nový objekt	39
3.8	Promotic - PmNumber - objekt typu číslo	40
3.9	Promotic - PmString - objekt typu textový řetězec	40
3.10	Promotic - PmData - sdružující proměnné různého typu	41
3.11	Promotic - objekty projektu UPS	42
3.12	Editor grafického obsahu - paleta prvků	43
3.13	Kruhový měřicí přístroj - připojení proměnné	44
3.14	Kruhový měřicí přístroj - nastavení vlastností	44
3.15	Kruhový měřicí přístroj - doplnění textových popisků	45
3.16	Vizualizace baterie	45
3.17	UPS - vizualizace, celek	46
4.1	Bezpečnost řešená na vnějším perimetru sítě firewallem	55
4.2	Dual home computers	56
4.3	Dual home server	57
4.4	Oddělení sítí pomocí zařízení různých typů	58
4.5	Důvěryhodná vs demilitarizovaná zóna	59
4.6	Důvěryhodná vs demilitarizovaná zóna - řešení s použitím dvojice firewallů	60
6.1	Technický pohled na postup data miningu (adaptováno z [19])	74
6.2	Manažerský pohled na postup data miningu (adaptováno z [19])	75
6.3	Úlohy data miningu - pokrytí konceptu (adaptováno z [19])	77
6.4	Proces data miningu v SAS Enterprise Miner (převzato z [61])	79
6.5	Životní cyklus data miningu (převzato z [35])	80
6.6	Příklad struktury zjednodušené objednávky	81
6.7	Struktura informačních systémů podniku	82
6.8	Datová krychle v OLAP (adaptováno z [19])	83

7.1	Rozhraní RStudio v OS X	87
7.2	Data Viewer - proměnná <code>tornado_data</code>	88
7.3	Párové srovnání vlastností kosatců v datové sadě iris	93
7.4	Délka vs šířka listu kosatců v datasetu iris (bodový graf)	94
7.5	Dostupné typy vykreslování ve funkci <code>plot</code>	95
7.6	Rozložení délky listu dataset iris (histogram)	96
7.7	Délka listu dataset iris (boxplot)	96
7.8	Rozložení počtu prodejů podle jejich úspěšnosti (pie)	97
8.1	Filozofie použití trénovací a validační množiny	100
8.2	Klienti banky -ručně vytvořený rozhodovací strom	104
8.3	Klienti banky - strom vygenerovaný toolkitem <i>Tree</i>	107
8.4	Asociační pravidla - dataset Titanic (27 základních pravidel)	113
8.5	Asociační pravidla - dataset Titanic (zaměřeno na přeživší podle věku a třídy)	113
10.1	Schéma neuronu (převzato z [20])	122
10.2	Neuron vizuálního kortexu (převzato z [37])	123
10.3	Schématické znázornění modelu perceptronu	124
10.4	Graf sigmoidy	124
10.5	Jednoduchá neuronová síť pro OCR	125
10.6	Schéma vícevrstevné sítě	125
10.7	Neuronová síť pro predikci hodnot funkce $y = x^2$	127
10.8	Funkce $y = x^2$ - srovnání modelu neuronovou sítí a lineární regresí	128
10.9	Přeučení model	129
10.10	Klasifikace klientů banky neuronovou sítí	130

Seznam tabulek

8.1	Přidělování úvěrů banky (převzato z [19])	102
8.2	Příjem vs úvěr	103
8.3	Přidělování úvěrů - 2. větvení	103
8.4	Přidělování úvěrů -3. větvení	104
8.5	Párek vs hořčice	109
8.6	Rozložení datasetu Titanic	110
10.1	Trénovací množina funkce $y = x^2$	126

Úvod

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

Ačkoliv název předmětu má na svém konci číslovku 3, text skript je samonosný a jeho pochopení nevyžaduje předchozí znalosti předmětů *Bezpečnostní informatika* a *Bezpečnostní informatika 2*, ačkoliv určitá znalost zejména v oblasti šifrování 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, ale např. také řízení klimatu v budovách.

Bezpečnost 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.

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.

Pokud studujete *Bezpečnostní informatiku 1* v rámci celoživotního vzdělávání, tak v průběhu studia narazíte také na tzv. *korespondenční úkoly*. Tyto úkoly je potřeba vypracovat a v termínech daných Vaším studijním harmonogramem je odevzdat. Tyto korespondenční úkoly poslouží k Vašemu závěrečnému hodnocení.

Pokud jste studenty řádného studia v denní nebo kombinované studijní formě, pak i Vy narazíte na korespondenční úkoly, ale můžete je v klidu ignorovat – Vaše hodnocení bude provedeno na základě písemné zkoušky.

Pro zjednodušení orientace v textu jsem zavedl systém ikon:

V novém (už šestém!) vydání skript jsem se rozhodl pro trošičku jiný způsob přípravy skript a celá jsem je přepsal v **Desktop Publishing (DTP)** systému **L^AT_EX**. Důvodem jsou některé schopnosti, které je s běžnými textovými procesory je možné dosáhnout pouze stěží a také to, že řada z vás bude

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**

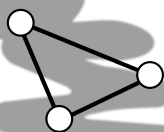
U této ikony najdete vysvětlující text, poznámku k probíranému tématu, která problém uvede do širších souvislostí, popřípadě důležitou definice.

**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.

studovat tento text přímo v počítači (tabletu, čtečce elektronických knih nebo mobilním telefonu) a v takovém případě budete chtít využít nejspíše všech schopností, která Vám tato zařízení poskytují.

Kolikrát jste si pomysleli - „jaké by to třeba bylo, kdybych mohl klepnout na jednu z těch divných

**Přestávka**

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

zkratk (které informatici tak milují) a ona by mě přesměrovala automaticky na seznam zkratk“? Nebylo by lepší kdyby na daný literární pramen bylo možné se dostat přímo klepnutím na jeho číslo v textu, nebo aby jste nemuseli vybranou pasáž hledat přes čísla stránek, ale postačovalo by kliknout na jméno kapitoly v obsahu?

Mě jako studentovy by se to líbilo a proto doufám, že je oceníte i Vy, protože všechny výše uvedené možnosti skriptu ve formátu **Portable Document Format (PDF)** obsahují. 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 třetí 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 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.

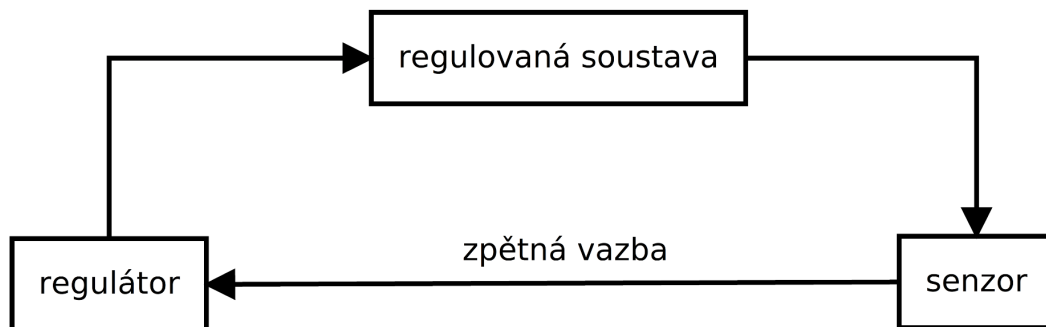
Nejprve se automatizace zaváděla do velkých výrobních linek, dnes se však automatizují prakticky všechny procesy od procesů výrobních, po systémy klimatizační ve velkých budovách až k regulaci kotle ve Vašem rodinném domě.

Automatizace se tedy stará o chod regulované soustavy tak, že automaticky aplikuje předem stanovené postupy na ovládání regulované soustavy a sleduje, zda změna v nastavení vedla k žádoucím změnám v chování soustavy. Uvažujme tři základní prvky automaticke:

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

Vizuálně je možné si výše uvedené prvky představit jako na obr. 1.1. Trojici těchto prvků označujeme jako *regulační soustava*. **Regulační soustavou** přitom rozumíme jakoukoliv soustavu, která může být regulována. Prakticky se je v regulační soustavě obvykle použito velké množství senzorů a často také regulátorů. Regulovaná soustava (v regulační soustavě) je obvykle pouze jedna, ale v praxi je často řadíme za sebe, takže na sebe logicky navazují.

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 klimatizace místnosti. 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 přesáhnuta 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 mez, zapne se přitápění pomocí tepelného čerpadla. Řídicí systém opět na základě informací podle údajů získaných ze senzoru teploty, 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 případě teplotním. 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 apod. 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 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 umísťována do prostorů, kde existuje určitá kontrola nad provozním prostředím - může se jednat o speciálně přizpůsobené skříně nebo místnosti (serverovny).



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

PLC také nejsou univerzální, alespoň ne stejným způsobem jako běžné počítače. Je sice možné je programovat, ale toto programování se děje na běžném počítači ve vývojovém prostředí výrobce PLC. Připravený program je následně nahráván na PLC pomocí přenosných médií nebo po síti. PLC pak následně bude vykonávat tento a pouze tento program.

To také ale znamená, že PLC různých výrobců (ani jejich řídicí systém) není možné zaměňovat - jedná se o ucelené řešení na míru požadovaného nasazení (řízení 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 nekompatibility. U velkých technologických celků organizace řeší tuto situaci tak, že se prostě předzásobí dostatečným počtem náhradních PLC. Následně pak v případě selhání PLC v technologii 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.

1.3 SCADA

Z tohoto důvodu se pro technologie připravují **dispečerská/dohledové/kontrolní pracoviště**, která umožní detekovat a také řešit takové problémy. Pro provoz takového pracoviště je potřeba, aby dispečer měl přístup k aktuálním datům popisujících současný stav technologie a to v takové formě, aby byl schopen:

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

To není zcela triviální problém - i jednoduché řízené technologie poskytují 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 řízené technologie) a připojených senzorů je obvykle velké množství. Zobrazení těchto dat v tabelární formě proto obvykle nepřichází v úvahu.

Z tohoto důvodu se pro řadu technologií instaluje také 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 (čase, který se blíží reálnému času, ale už je zde jisté drobné zpoždění).

SCADA systém informace obvykle nebere přímo ze senzorů (prostřednictvím PLC), ale z nějakého k tomu určeného místa, obvykle výkonného databázového serveru. Technicky je možno zajistit přímou komunikaci SCADA - PLC, ale takové řešení obvykle není dobře škálovatelné.

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 a nikoliv nutně úplně. To je poměrně zásadní rozdíl oproti nárokům, které jsou kladeny na databáze používané v běžných informačních systémech, kde naopak je primárním úkolem právě úplnost uchovávaných dat.

Pro tyto specifické nároky jsou někdy tyto databáze označovány jako *real-timeové*. Problematické realtimových databází se budeme věnovat v samostatné kapitole.

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

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

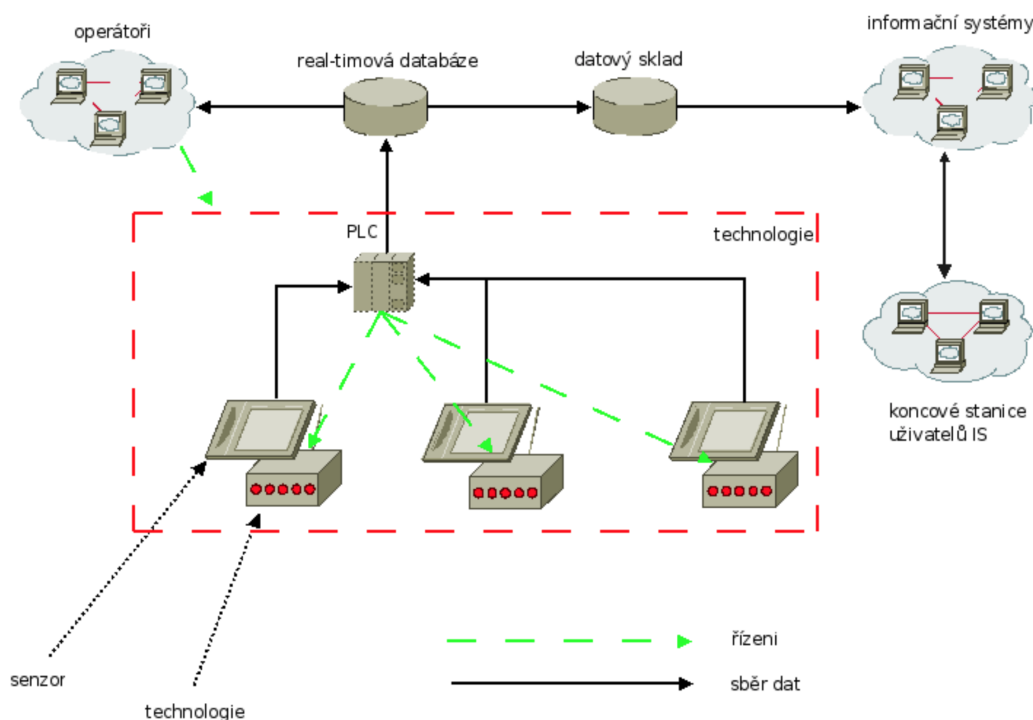
V případě, že dálkový zásah není z nějakého důvodu možný, zbývá možnost manuální zásahu, na místě.

Všimněte si 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.

NDIC zpracovává údaje z řady různých zdrojů zejména pak [71]:

- 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



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

- 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 realizována na míru. SCADA systém jako takový je obvykle realizován 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 to vyžaduje čas a peníze.

Z tohoto důvodu je aplikace SCADA provozována na platformě, pro kterou byla vyvinuta a to i v případě, že již není nadále podporována výrobcem. K aktualizaci buď nedochází vůbec nebo 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 nemusí podporovat nejnovější verzi operačního systému).

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.



Obrázek 1.4: Dohledové pracoviště NDIC (převzato z [40])

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 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 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
 - rozhraní je v řídicí místnosti
 - rozhraní je přímo na řízené technologii
- přístupnost



Obrázek 1.5: Řešení monitoringu fotovoltaické elektrárny (převzato z [25])



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.

Uvedme si příklad: v samostatné kapitole těchto skript a také ve cvičení budeme pracovat se systémem Promotic [48] 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

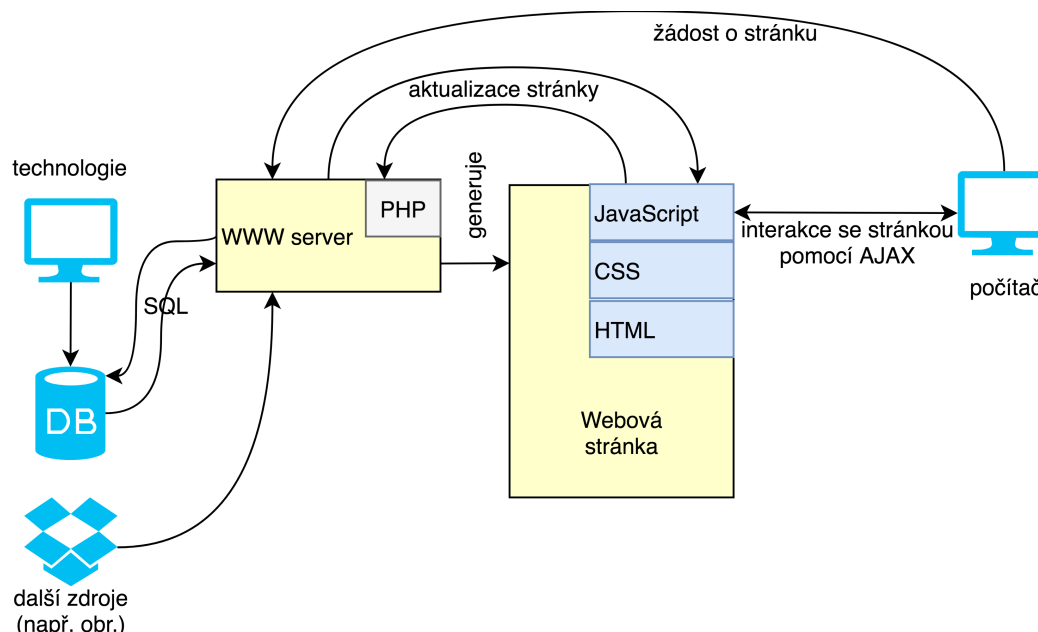
Ve výše uvedeném textu se pracuje poměrně extenzivně s životností, podporou apod. Co jsme nezmínili jsou otázky bezpečnosti. **Je možné považovat jednotlivé součásti řídicího systému za bezpečné i v případě, že už nadále nejsou podporované výrobcem?** Odpověď je poměrně složitá - *ne tak docela*. Co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

Možnostmi izolace řídicího systému se budeme zabývat později.

- 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.



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

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 k dalším zdrojům. Může se jednat o fragmenty souborů, ale třeba také data obsažené 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 ne 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 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 znova stránku celou.

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

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.

Důvodem, proč zkoumáme strukturu webových aplikací nebo proč se vůbec v tomto předmětu zabýváme problematikou automatizace jsou prosté: 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

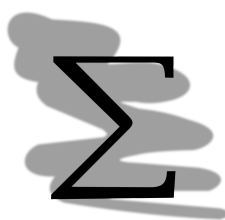
k dispozici velké množství. Podobně velké množství je ale také množství různých scénářů selhání technologie.

Z hlediska možných opatření lze jít po příčinách, tedy zabránit iniciaci nebezpečí nebo lze přerušit popř. minimalizovat následky takové posloupnosti. 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?

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



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 také 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. Vysvětlete 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. Sensory 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 na funkci databáze shromažďující informace o stavu technologie - tzv. *realtimové databáze*.

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 minut.

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. Očividně vyvíjí tlak na to, aby údaje ze senzorů byly ukládány co možná nejdříve od okamžiku jejich vzniku a řídicí systém s nimi pak mohl pracovat. Existuje řada možností, jak toto zajistit - řídicí systém může třeba pracovat se senzory přímo, ale takové řešení není dobře škálovatelné. Z tohoto důvodu jsou často takové údaje konsolidovány 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, pro složitější technologie se ale častěji používají databáze. Databáze jsou např. velmi jako řadou informačních systémů pro ukládání dat. Jenomže požadavky klasických informačních systémů na ukládání dat jsou odlišné než pro ukládání dat pro automatizaci. Zkusme je porovnat.

Požadavky běžného informačního systému

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

Požadavky na data historiana (real-timeová databáze)

1. hodnota dat v čase klesá
2. ne všechny data je nutné ukládat
3. vysoká rychlost manipulace s daty je nutnost.

Požadavky kladené na databázi 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 *reálné databáze* více méně protimluv. Každý proces měření a následného ukládání dat vyžaduje určitý čas. Tím v celém procesu vznikají latence. V zájmu efektivity řízení, je, aby tyto prodlevy byly co možná nejmenší - odtud tedy tlak na zpracování v reálném čase.

Technicky jsou ale reálné databáze, alespoň v současnosti, databázemi relačními. Otázka tedy je jakým způsobem lze práci databáze zrychlit. Metod, které lze použít je ve skutečnosti velké množství - volbou vhodné databáze a její správné nastavení, investicí do hardware, na kterém je databáze provozována.

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

- šířce přenosové kapacity
- rychlost operací **Input-Output (IO)**
- kapacita paměti

V případě software pak může být problém:

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

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

Z hlediska možných řešení ve výše uvedeném seznamu není obsažena možnost škálování výkonu pomocí použití *databázového klastru*. Hlavní myšlenka tohoto řešení je ta, že možnost navyšování databázového serveru jako jednoho zařízení je omezená. Je možno zvětšit paměť, použít výkonnější procesor, nebo procesory apod. Ve finále jsme ale vždy limitovaní technickými možnostmi rozšiřitelnosti daného systému.

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 představit 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.

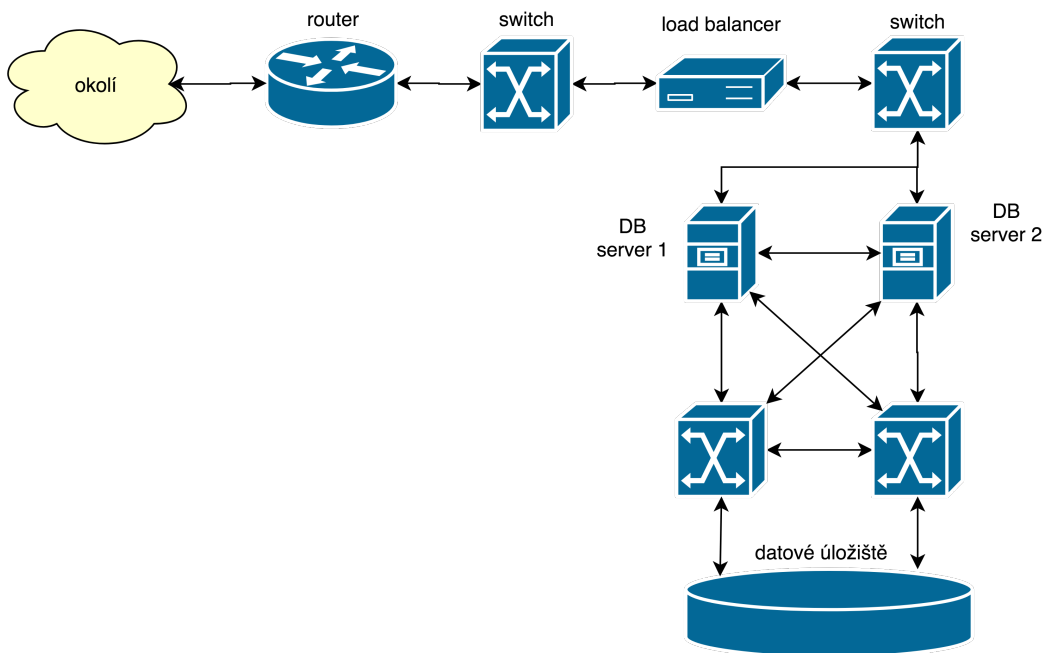
To ale znamená, že databázové operace nejsou prováděny zároveň na obou databázových serverech. To znamená, že load balancer představuje určitou úroveň abstrakce databázové architektury. Externí zařízení vnímají celý kluster jako jedno zařízení, ale vnitřní organizace klastru je složitější.

Synchronizace dat napříč databázemi pak probíhá procesem nazývaným *replikace dat* s tím, že replikovány jsou pouze změny v uložených datech.

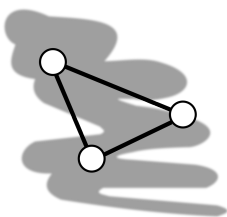
Kromě možnosti navýšení výkonu je také výhodou vzájemná zastupitelnost těchto zařízení i v případě výpadku. Např. pokud v klastru selže jeden z databázových serverů, load balancer rozdělí jeho zátěž na zbývající servery. Řešení pomocí klastru je tak z mnoha pohledů odolnější (ne jen výkonnější).

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 reálnou 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*.



Obrázek 2.1: Databázový klastř

**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 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.

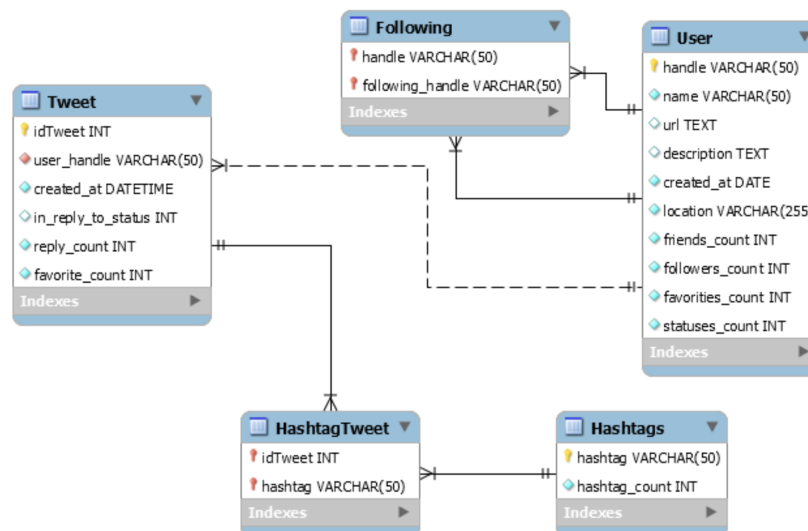
Databáze z obr. 2.2 by mohla být použita např. pro analýzu vazeb mezi jednotlivými uživateli. Na obr. hrají hlavní roli tabulky User a Tweet, vše ostatní se pak vztahuje k nim. Propojení tabulek se děje pomocí relací. Úkolem relace je specifikovat způsob, jakým spolu souvisí tabulky na obou stranách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

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 dat, jako fotografie)
- údaj o čase, kdy bylo provedeno měření
- případně údaj o senzoru jako takovém



Obrázek 2.2: ERD diagram metadat příspěvků v síti Twitter (adaptováno z [70])

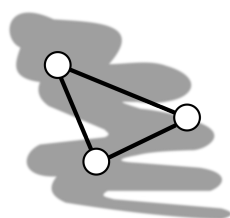
Lze uvažovat o různých přístupech. Jednotlivým sensorům můžeme např. vytvořit samostatné tabulky. Struktura takové tabulky by mohla být následující: *Senzor1*(DatumAČas, hodnota). Jméno tabulky pak identifikuje senzor, ze kterého bylo pořízeno měření.

Jinou možností by bylo zobecnění výše uvedeného řešení pro větší množství sensorů 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 sensorů evidovány samostatně v jednotlivých sloupcích tabulky. Tato konfigurace ale předpokládá, že interval měření bude konstantní (pro všechny v tabulce zavedené senzory stejný).

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.



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 skript.

2.2.2 Jazyk SQL

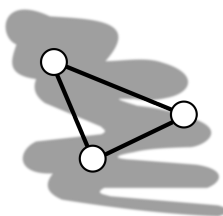
Jakým způsobem lze s databází manipulovat? Většina databází pracuje s jazykem **SQL**. Příkazy (dotazy) jazyka lze rozdělit podle toho, co dělají:

- Dotazy definující nebo měnící strukturu databáze (CREATE TABLE ..., ALTER TABLE ...)
- Dotazy výběrové (SELECT FROM ...)
- Dotazy modifikující data (INSERT INTO ..., UPDATE ... WHERE ...)

- 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 kapitolu skript *Bezpečnostní informatika 1 - Návod do cvičení* [75], 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.

```

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 tabulky 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 objektů 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.

```

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.

```
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 popsaným 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

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

Všechny Tweety uživatele @uzivatel:

```
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.

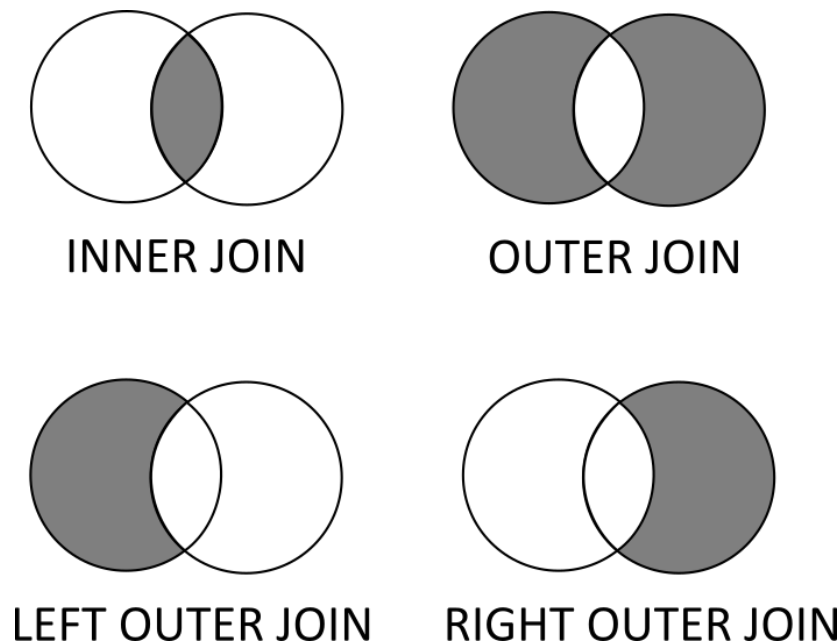
Výběry lze realizovat také napříč tabulkami. Řekněme, že bychom chtěli vybrat všechny tweety s haštagem #vyber. Řekněme, že nás ale budou zajímat pouze sloupce user handle a created at:

```
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.

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 množstvím spojovaných tabulek rostou národy na spotřebovávané zdroje databázového serveru.



Obrázek 2.3: Výsledky spojení dvou tabulek

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:

```
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, které jsou povinné. V tabulce Tweet jsou ale povinné 3 sloupce, tím třetím je idTweet. Tento sloupec je ale nadefinován jako automatické číslo, databáze jej proto vyplní sama.

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:

```
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.

```
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';
```

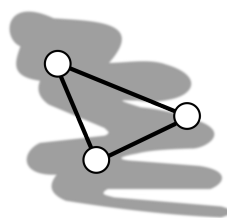
První příklad nastaví datum a čas vytvoření tweetu na 8. 8. 2018 5 hodin ráno. V druhém příkladu bude pro všechny tweety uživatele @uzivatel nastaven počet lajků na 0.

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).

```
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 [72]. Vhodným zdrojem informací je pak také manuál Vámi zvoleného databázového systému.

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. Na druhou stranu paralelně může databáze zpracovávat více (nezávislých) transakcí.

Souvisejícími operacemi pak máme na mysli to, že příkazy jazyka SQL, kterými manipulujeme s databází spolu mohou souviset, ale také nemusí - prováděné operace spolu nutně nemusí souviset. Technicky můžeme do jedné transakce vložit nezávislé příkazy, ale praktický smysl to nemá, protože se pouze prodlouží dobu vykonávání příkazů, které by se jinak vykonávaly paralelně.

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 „rozpojí“ a tím se naruší integrita databáze.

Správný postup je pak takový že musíme aktualizovat handle ve všech třech tabulkách. 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 selže? Důvodů může být celá řada - mohli jsme udělat chybu v příkazu, nebo zrovna v okamžiku provádění příkazu byl problém v databázi. Pokud bychom přijali pouze částečnou aktualizaci databáze narušili bychom její konzistenci.

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.

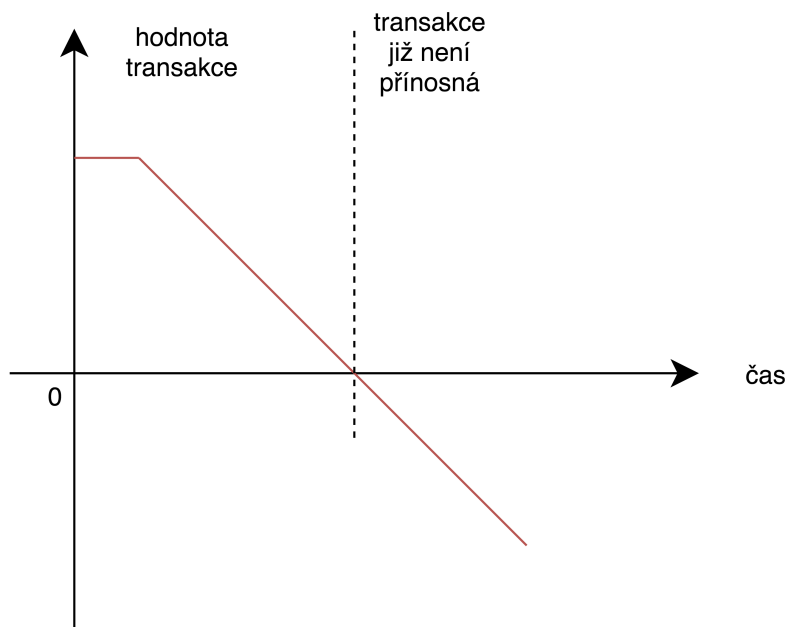
Nevratnost – pokud transakce byla úspěšně dokončena (příkazem COMMIT), není možné ji odvolat. Změny transakce je možné zvrátit pouze v průběhu transakce.

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:

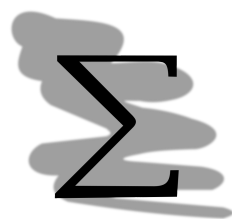
1. opožděné dokončení transakce je přínosné – hodnota přínosu obvykle se ale obvykle nerovná přínosu ze včas dokončené transakce (např. aktuální hodnota senzoru výrobní linky vs. hodnota téhož senzoru z minulého dne). V takovém případě se databázový systém snaží transakci dokončit

- obvykle ale opožděné transakci přiřazuje nižší prioritu tak, aby nebyla ohrožena aktuálnost nově zadávaných (čerstvých) údajů.
- 2. V okamžiku, kdy transakce ztratí hodnotu úplně nebo její hodnota klesne pod určitou úroveň, může databázový systém její dokončení stornovat, tak aby si uvolnil systémové zdroje pro vyřizování přínosných transakcí. Ne všechny typy transakcí lze stornovat.
- 3. Hodnota nedokončené transakce může dokonce poklesnout pod nulovou hodnotu – nedokončení transakce způsobuje další náklady. V takovém případě systém může naopak navýšit prioritu 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 v souvislosti se změnami v životě databázového serveru.

Dopad na výkon mohou mít také bezpečnostní aktualizace operačního systému. Např. záplaty na chyby v procesorech společnosti Intel známé pod jmény Spectre [39] a Meltdown [43] 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 *realtimeové*. 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 realtimeových 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 realtimeová 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 [48] Ostravské společnosti MycroSys. Jedná se o nástroj umožňující realizovat SCADA systém, popř. realizovat HMI.

Po přečtení této kapitoly budete vědět

1. jakým způsobem jsou realizovány SCADA systémy

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 déle než jiným.

K tomu Vám mohu pouze popřát, abyste u studia vytrvali ... látka není složitá, je pro Vás pravděpodobně pouze 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 konkrétně pro verzi XP nebo vyšší.

Společnost MicroSys poskytuje ke stažení jednak poslední verzi prostředí - v době psaní těchto skript (VIII 2018) se jednalo o verzi 8.3.26, z důvodu kompatibility poskytuje také verze starší.

Stažení je možno provést ze stránek <https://www.promotic.eu>. Preferovat byste měli poslední stabilní verzi. V případě, že se Vám ji ale nedaří nainstalovat (což by se vzhledem k poměrně široké podpoře různých verzí operačního systému Windows nemělo stát), můžete vyzkoušet některou z předchozích verzí.

Promotic je dopředně kompatibilní - to znamená, že projekty vytvořené starší verzí Promotic lze otevřít v novější - bez ztráty funkčnosti, ale pozor opačně to již nejde.

Na stránkách Promotic je ke stažení pro každou verzi dostupné samotné prostředí (označené např. Verze 8.3.26) a potom web klient. Web klient doplňuje do aplikací Promotic schopnost přístupu k výsledné aplikaci po síti, pomocí webového prostředí. My ale tuto funkčnost nevyužijeme - takže pro práci nám bude dostačovat základní vývojové prostředí.

Při stahování dejte pozor, abyste stáhli poslední stabilní verzi. MicroSys někdy pro testování uvolňuje také vývojové (nestabilní) verze prostředí - zejména v období před oficiálním uvedením nové verze do oběhu. V době psaní těchto skript ale žádná vývojová verze nebyla na stránkách prostředí dostupná.

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.

První projekt

V menu start instalátor přidal odkazy na:

- Demo aplikace PROMOTIC a
- Vývojové prostředí PROMOTIC

Demo aplikace slouží k demonstraci schopností prostředí Promotic. Může Vám tedy sloužit pro inspiraci, co je možné v Promoticu realizovat. Demo aplikace je také plně editovatelná - takže lze zkoumat, jak bylo funkčnosti dosaženo.

Vytváření a editace projektů je možná z *vývojového prostředí Promotic*.

Při spuštění vývojového prostředí se otevře základní menu umožňující:

- vytvořit *novou aplikaci*,
- spustit *demo aplikace* - toto demo je možno spustit také samostatně z menu start operačního systému,
- *otevřít existující aplikaci*,
- příklady v aplikaci obsahují sadu příkladů použití vybraných v Promoticu dostupných technologií (databáze, alarmy, reporty, trendy a další).
- *učebnice* - obsahuje podrobně vysvětlený postup realizace projektu kotelny (společně s projektem v této kapitole a menšími projekty realizovanými ve cvičení získáváte základní baterii příkladů, ze které můžete vyjít při návrhu vlastních projektů)
- *dokumentace* - obsahuje podrobnou dokumentaci ke všem komponentám dostupným pro vývoj.

Jelikož nemáme zatím žádný projekt vytvořen - vytvoříme si nový. **Pro nový projekt si vždy vytvořte samostatnou složku.** Každý projekt obsahuje soubor s příponou .pra. Tento soubor obsahuje základní strukturu projektu, ale také obrazy aplikace pro editaci a ve zkompilevané podobě a řadu dalších věcí.

Kromě .pra souboru může (ale nemusí) projekt obsahovat další zdroje, jako jsou např. externí obrázky použité ve vizualizaci, konfiguraci, log, dočasné soubory, apod. Některé z nich mohou být vyžadované pro běh námi vytvářené aplikace. Projekt v samostatné složce nám umožní jednoznačně identifikovat, které soubory jsou součástí projektu a případně je přenést, zabalit a odevzdat jako semestrální projekt apod.

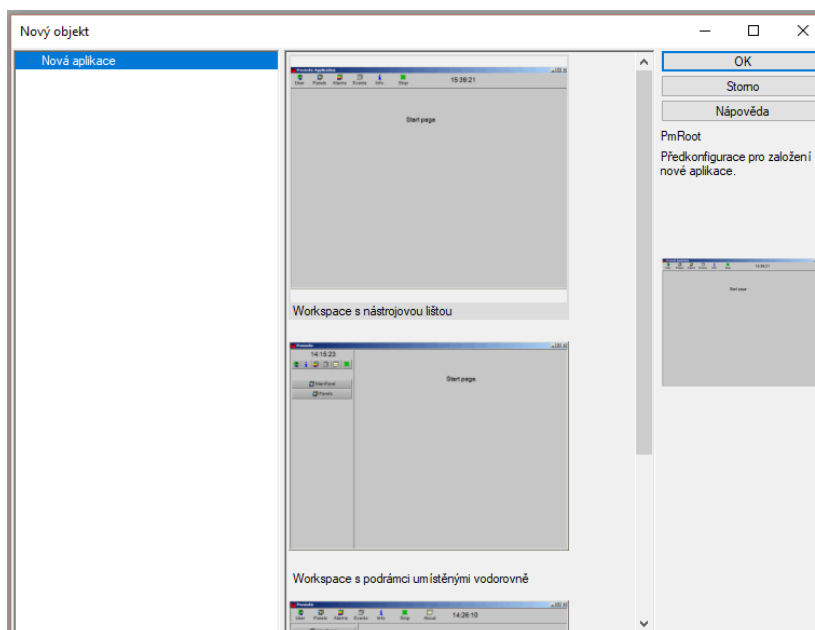
My budeme vytvářet jednoduchou vizualizaci fungování systému UPS a proto si soubor také tak pojmenujeme (UPS.pra).

Po specifikaci místa a názvu projektu se objeví dialogové okno *Nový objekt* umožňující nastavit základní vzhled budoucí aplikace, viz obr. 3.1.

Z hlediska funkčnosti jsou všechna rozložení nabízená v rámci dialogu na obr. 3.1 ekvivalentní - rozdíl je tedy pouze vizuální. Aplikaci vždy tvoří *workspace* (největší část okna aplikace s nápisem *start page*), zde se budou nacházet jednotlivé komponenty vizualizující něco - v našem případě činnost UPS.

Povinná je také *nástrojová lišta* - ta umožňuje základní manipulaci s vytvořenou aplikací, ukončit její běh, otevírat jednotlivé obrazy apod.

Některá rozložení navíc obsahují také tzv. *podrámce*, které mohou sloužit třeba pro zobrazení seznamu předpřipravených obrazů apod.

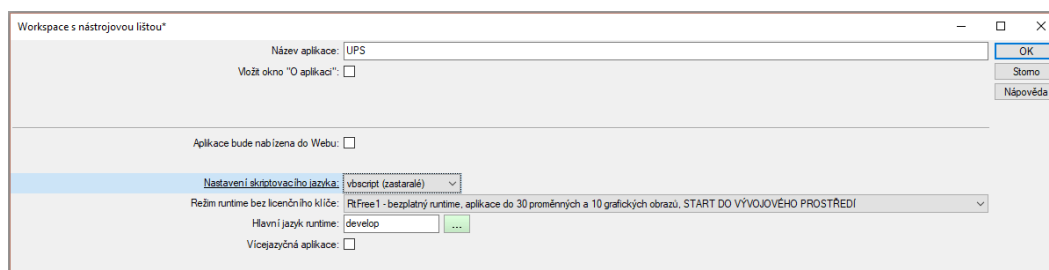


Obrázek 3.1: Základní rozložení aplikace Promotic - Nový objekt

V našem projektu budeme používat pouze jeden obraz pro vizualizaci, proto podrámce potřebovat nebudeme. Já osobně preferuji první rozhraní s nástrojovou lištou pod titulkem okna aplikace. Je to způsob, který používá většina aplikací a proto jej použijí také.

Kliknutím vyberu a výběr potvrdím kliknutím na tlačítko OK.

V dalším kroku nastavujeme vlastnosti projektu. Některé z nich jsou důležité, jiné jsou důležité méně - všechny je možno v případě potřeby změnit také později přímo uvnitř vývojového prostředí. Tady ale máme jednotlivé položky hezky pohromadě a můžeme tak provést nastavení jednou - a obvykle ho už pak nebudeme muset měnit. My nastavíme název aplikace na UPS, odklikneme volbu *Aplikace bude nabízena do webu* a změníme nastavení skriptovacího jazyka na *vbscript (zastaralé)*. Ostatní volby ponecháme, tak jak jsou - viz obr. 3.2.



Obrázek 3.2: Promotic - nastavení workspace s nástrojovou lištou

Zastavme se u některých voleb. Promotic umožňuje realizovat interaktivní části pomocí jednoho z podporovaných skriptovacích jazyků. V současnosti je podporován Visual Basic Script a JavaScript. V obou případech se jedná o plnohodnotné programovací jazyky. Promotic tradičně využíval Visual Basic Script. Doplnění podpory JavaScriptu tak přišlo později.

Paradoxně - vývojové prostředí označuje Visual Basic jako zastaralé prostředí, avšak podpora JavaScriptu v současnosti není úplná. Pro naše účely tak JavaScript není možné použít, protože nepodporuje skripty časovače, které budeme potřebovat.

Oba podporované programovací jazyky je možno extenzivně využívat také v dalších aplikacích (mimo Promotic). Visual Basic je využíván např. pro psaní maker v balíku MS Office, grafickém balíku CorelDRAW, Adobe Illustrator (ted podporuje také AppleScript a JavaScript) a řada dalších. JavaScript je pak jazykem zajišťujícím interaktivitu webových stránek a je implementován také v řadě programů pro účely vytváření maker, viz třeba Adobe Illustrator.



Programování?

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

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

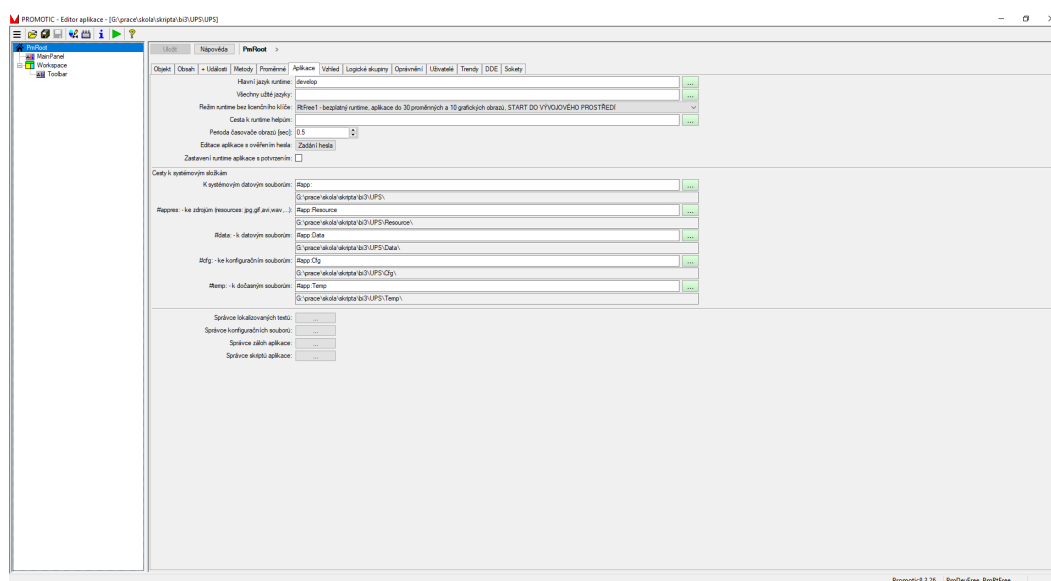
Pozor si také dejte na režim runtime bez licenčního klíče. Promotic, který MycroSys poskytuje ke stažení je ve skutečnosti plnou verzí programu. Pokud není přítomen hardwarový klíč nebo nainstalován licenční server, bude Promotic spouštěn v režimu bez licenčního klíče a jeho funkčnost bude podléhat omezením velikostí projektu, který lze v prostředí vytvořit a provozovat.

RtFree1 je předvolen a umožňuje vytvořit projekt s maximálně 30-ti proměnnými a 10-ti grafickými obrazy. Při nastavení *RtFree1* se projekt otevře v režimu pro vývoj, otevře se tedy ve vývojovém prostředí. Oproti tomu v režimu *RtFree2* se otevře do běhového prostředí - otevře se obraz a začne být vizualizována technologie.

Logicky je ale nejprve nutno aplikaci vyvinout a pak je možné ji spouštět, proto ponecháme režim *RtFree1*.

Po kliknutí na tlačítko OK se konečně vytvoří projekt a otevře se samotné vývojové prostředí. Po vytvoření projektu budete automaticky vyzváni k provedení kompilace obrazů - tu proveďte. Tímto způsobem se připraví nástrojová lišta a první prázdný obraz aplikace.

Co je to kompilace obrazů a proč je potřeba dělat si vysvětlíme později při výkladu editoru obrazů. Vývojové prostředí vypadá podobně jako na obr. 3.3.



Obrázek 3.3: Promotic - vývojové prostředí

Vývojové prostředí má tři základní části:

- panel nástrojů,
- průzkumníka projektu a
- největší část obrazovky pak zabírají vlastnosti právě zvoleného objektu.

Vlastnosti objektu se mění v závislosti na tom, jaký objekt je vybrán. Na obr. 3.3 je vybrán základní objekt projektu (kořen projektu) - *PmRoot*. Všimněte si, že vlastností je tolik, že je bylo nutné rozdělit do jednotlivých listů vlastností. Otevřen pro náhled je pak vždy ten, kde Promotic očekává, že se budou provádět změny nejčastěji.

My z prostorových důvodů vysvětlíme pouze základní vlastnosti. 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. Projekt dále vytvořil obrazy *Main Panel* a *Toolbat*. *Main panel* je

základní obrazovka aplikace, Toolbar pak obsahu 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.

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:

- zdroje (Resource)
- datové soubory (Data)
- konfigurační soubory (Cfg) a
- dočasné soubory (Temp)

Výše uvedené složky se implicitně vytvářejí v hlavní složce projektu - proto také bylo tak důležité, abychom pro projekt vytvořili samostatnou složku. V případě práce s více projekty z jedné složky by totiž mohlo dojít ke smíchání dat z těchto systémových složek, což obvykle není žádoucí.

Této informace můžeme také využít v případě, že potřebujeme např. přidat do projektu nějaký obrázek, který není dostupný ve standardní knihovně obrázků poskytovaných Promoticem. Obrázek získáme např. z Internetu a uložíme jej do složky Resources našeho projektu.

PmPanel

Objekt typu *PmPanel*, česky uživatelská grafika umožňuje do projektu přidávat obrazy. 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.

Pro obraz je možno nastavit také skriptovací engine - implicitní nastavení odpovídá jazyku zvolenému při vytváření projektu (viz obr. 3.2).

Hlavní část editace uživatelské grafiky je prováděna v samostatné vývojové aplikaci - *editoru obrazů*, který lze spustit kliknutím na tlačítko Editace grafického obsahu.

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

PmWorkspace

Pracovní plocha umožňuje zvolit základní rozložení aplikace. Nejdůležitější vlastnosti se nachází na záložkách Hlavní a Rámce.

V hlavní záložce je možno nastavit, zda tato pracovní ploch představuje hlavní okno aplikace, jaký má mít název a v jakém stavu se má nacházet při spuštění (implicitně v maximalizovaném).

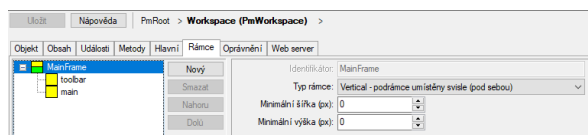
Z hlediska funkce výsledné aplikace je ale důležitější záložka Rámce. V ní máme možnost zvolit jak budou jednotlivé obrazy vizuálně poskládány na obrazovce.

Kořen pracovní plochy tvoří MainFrame. V rámci něj je možno nastavit rozložení. Předvoleno je rozložení, které jsme zvolili na obr. 3.1 při vytváření projektu.

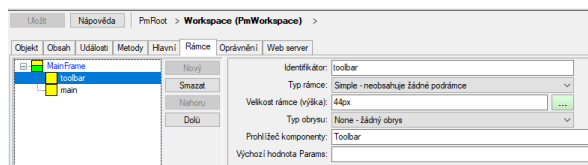
Pod MainFrame jsou pak pod zvoleného rozložení dva nebo tři další objekty, kam už lze mapovat určitou uživatelskou grafiku. Pro vertikální rozložení je nutno namapovat (viz obr. 3.4) pouze dva objekty a to toolbar (panel nástrojů, viz obr. 3.5) a main (viz obr. 3.6), což je hlavní obrazovka aplikace.

Hlavní obrazovka se automaticky načte při startu aplikace do běhového režimu. Případné další obrazy si bude muset uživatel buďto manuálně vyžádat nebo jejich zobrazení bude muset být vyvolané nějakým skriptem, který k tomuto účelu připravíme.

Mapování uživatelské grafiky se provádí ve vlastnosti prohlížeč komponenty. Velikost rámce se určuje nastavením vlastnosti velikost. Všimněte si, že v našem projektu je na obr. 3.5 nastavena výška na 44px, ale u hlavního panelu na obr. 3.6 je výška specifikována pomocí znaku: *. Hvězdička je



Obrázek 3.4: PmWorkspace - vlastnosti MainFrame



Obrázek 3.5: PmWorkspace - vlastnosti toolbar

zástupným znakem, který v tomto kontextu znamená, že výška bude nastavena na zbytek výšky celé pracovní plochy.

Pokud u rámce nastavíme vlastnost typ rámce ze simple na horizontal nebo vertical. Je možné do něj pak vkládat další rámce a tak podle potřeby rozdělit pracovní plochu do takového množství segmentů, které potřebujeme pro řešení našeho problému.

Toto byly objekty, které máme v projektu nyní, jaké ale další objekty lze do projektu přidat a jak se takové přidávání vůbec realizuje?

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) 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).

Objeví se paleta dostupných objektů, viz obr. 3.7.

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

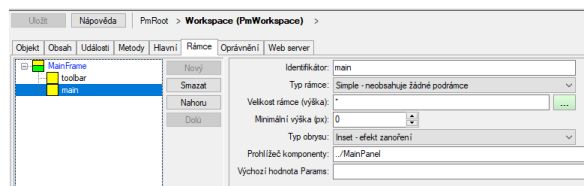
Tento objekt slouží pro management proměnných, se kterými bude Promotic pracovat. Proměnné obvykle odpovídají buďto měřeným veličinám nebo slouží pro interní potřeby aplikace.

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.

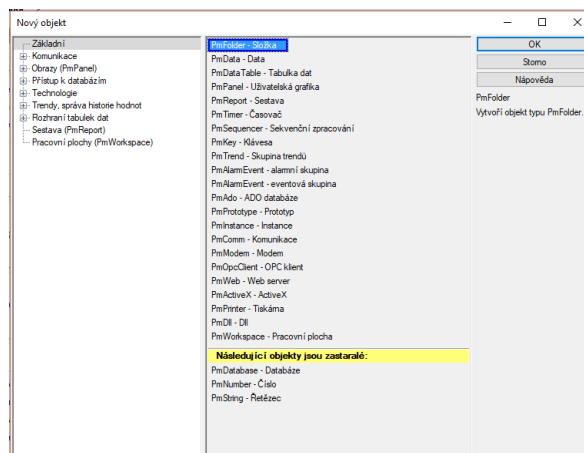
U zastaralých objektů lze očekávat, že v budoucích verzích bude odstraněna jejich podpora. V současnosti ale není zcela jasné jak dlouho budou tyto typy objektů ještě podporovány.

Na obr. 3.8 jsou zobrazeny vlastnosti objektu PmNumber, na obr. 3.9 jsou pak zobrazeny vlastnosti objektu PmString.

Jak je z obrázků 3.8 a 3.9 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.



Obrázek 3.6: PmWorkspace - vlastnosti Main



Obrázek 3.7: Promotic - nový objekt

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:

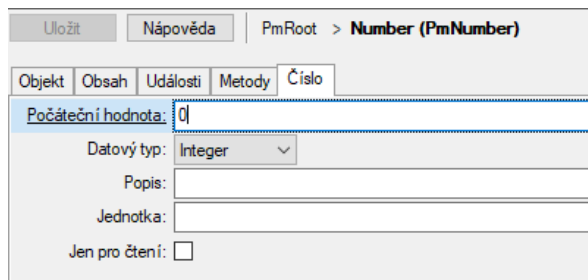
- boolean (0/1)
- byte (0 - 255)
- integer - celočíselný datový typ (16-ti bitový, -32768 - 32768)
- long - celočíselný datový typ (32 bitový, -2 147 483 648 - 2 147 483 648)
- single - desetinné číslo, jednoduchá přesnost (32 bitové $-3.4 \cdot 10^{38}$ - $3.4 \cdot 10^{38}$)
- double - desetinné číslo, 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?

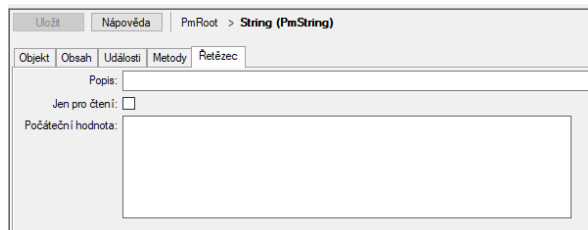
Vyplnění jednotky samo o sobě tento problém zcela neodstraní, umožní ale aby v průběhu vytváření aplikace jste klikli na proměnnou a měli ten AHA moment ... veličina je v té a té jednotce (a nemusíte o tom přemýšlet).

Rozhraní objektu PmData vypadá odlišně, viz obr. 3.10.

Nové proměnné do objektu PmData zadáváme kliknutím na tlačítko *Nová*. Rozhraní definice proměnné je univerzální. Kromě datových typů podporovaných v proměnných PmNumber a PmStrin



Obrázek 3.8: Promotic - PmNumber - objekt typu číslo



Obrázek 3.9: Promotic - PmString - objekt typu textový řetězec

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. 3.10 je cesta PmRoot/Data (objekt PmData jsem pojmenoval Data).

V případě, že bych v objektu Data nadeřinoval 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.

PmTimer - časovač

Časovač slouží pro spouštění určitých operací (definovaných skriptem) ve specifikovaném časovém 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. 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 pouze v jazyku Visual Basic Script (podpora JavaScript pravděpodobně bude doplněna v budoucích verzích Promotic).

Problematické oživení časovače budeme věnovat samostatnou podkapitolu.

3.2 UPS - specifikace podkladů pro tvorbu aplikace

UPS je zkratka pro Uninterruptible Power Source, tedy nepřerušitelný zdroj napájení. Tato zařízení obvykle obsahují baterii a logické obvody zajišťující vyhodnocování „kvality“ dodávek elektřiny v síti. V případě detekce problémů (nestability napětí, výpadku nebo naopak přepětí v síti) by přívod energie ze sítě měl být automaticky odpojen a elektřina by jednotlivým zařízeními měla být dodávána z baterie.

Naším úkolem bude vytvořit aplikaci, která bude funkci UPS vizualizovat.

První otázku, kterou musíme vyřešit jsou **proměnné**, tedy - jaké proměnné budeme potřebovat pro vyřešení problému:

- *napětí* - aktuální napětí v elektrické síti [V]



Obrázek 3.10: Promotic - PmData - sdružující proměnné různého typu

- *baterie* - stav baterie [%]
- *dolní mez napětí* a *horní mez napětí* - limity toho, co budeme považovat za normální [V]

Kromě výše uvedených proměnných bychom zcela jistě mohli vymyslet celou řadu dalších proměnných, jejichž použití by nám třeba umožnilo signalizovat některé stavy - třeba že baterie je takřka vybitá, že baterie se právě nabíjí, že UPS právě dodává elektřinu z baterie apod.

Volba proměnných vždy závisí na tom, co má výsledná aplikace přesně dělat. Z předpokládaného účelu je pak možno odvodit datové typy, počáteční hodnoty a také chování jako podklad pro návrh funkce časovače.

V našem případě všechny čtyři výše uvedené proměnné budou čísla. O typu čísla ale rozhodne způsob, jakým navrhne logiku simulátoru UPS - konkrétně s jakými kroky bude časovač pracovat. Pokud tyto kroky (změny za předem stanovený časový interval) budou celočíselné, pak mohou být také proměnné celočíselného datového typu.

Vzhledem k tomu, že pracujeme s napětím ve voltech a standardní napětí v elektrické síti je okolo 230 V a stav baterie bude v procentech, pak nám bude dostačovat typ Integer (nebudete potřebovat typ long).

Simulace

Pro účely simulace můžeme chtít, aby hodnota napětí v elektrické síti (s počátečním napětím 230 V) náhodně kolísala v každém časovém kroku o třeba 5 V.

Dolní a horní mez přípustného napětí jsou položky které můžeme nastavit třeba na 200 a 260 V, ale můžeme také chtít, aby případně tuto hodnotu mohl uživatel změnit sám v grafickém uživatelském rozhraní aplikace.

Dále předpokládáme, že v každém časovém kroku se bude baterie nabíjet nebo vyvíjet podle toho, zda hodnota aktuálního napětí v síti je v mezích přípustného napětí nebo ne. Řekněme, že změna bude ± 1 za časový krok.

Logickými omezeními by pak mělo být, že baterie se nemůže nabít více než na $> 100\%$ a vybit pod 0% . Podobně hodnota napětí v síti by nikdy neměla být menší než < 0 .

Výše uvedená omezení nám determinují to, jak by se měl chovat časovač. My už ale každopádně máme dostatek informací k tomu, abychom mohli v Promotic začít prakticky realizovat náš projekt UPS.

Realizace Projektu

Základní rozhraní aplikace včetně hlavní obrazovky Promotic už máme vytvořeno. Potřebujeme vytvořit jeden objekt typu PmData a jeden PmTimer.

Objekt PmData pojmenujeme Data a uvnitř vytvoříme čtyři proměnné:

1. *napeti* - počáteční hodnota 230
2. *baterie* - počáteční hodnota 50
3. *dln* (dolní mez napětí) - počáteční hodnota 200
4. *hln* (horní mez napětí) - počáteční hodnota 260

PmTimer nazveme timer. Na záložce časovač by měl být časovat povolen po startu, s periodou 1 s. Výsledek by měl vypadat podobně jako obr. 3.11.

... to nebylo tak hrozné, nebo ano?



První kroky návrhu aplikace

Podobný postup, jaký je uveden výše budete muset absolvovat pokaždé, kdy budete chtít něco implementovat a to nejen v systému Promotic.

Není dobrou strategií otevřít Promotic a začít v něm okamžitě něco tvořit. Šance, že výsledek Vaší práce bude za něco stát je v takovém případě minimální. Správný postup je, že si pořádně rozmyslíte celý projekt. Potřebujete zejména zodpovědět otázky:

1. Jaké hodnoty/proměnné potřebuji zpracovávat?
2. Jak proces nebo technologie ve skutečnosti funguje?
3. Jakou interakci od uživatele aplikace očekávám?

V první fázi si dělejte poznámky a náčrtky na papír. Zejména v začátcích Vás bude omezovat méně než samotný Promotic. Zodpovězení/rozmyšlení výše uvedených otázek Vám následně umožní, abyste se následně v implementační fázi soustředili pouze na to jak to implementovat (místo co, jak a proč implementovat).

PROMOTIC - Editor aplikace - [G:\prace\skola\skripta\bi3\UPS\UPS]

Obrázek 3.11: Promotic - objekty projektu UPS

3.3 Příprava obrazovek aplikace

Pro přípravu obrazovky potřebujeme spustit editor grafického obsahu. To je možné provést z objektu MainPanel kliknutím na tlačítko *Editace grafického obsahu*. Po kliknutí na toto tlačítko se editor spustí v novém okně. Editor obrazů a editor aplikace jsou nezávislé - můžete se tedy bez obav mezi oběma okny přepínat a podle potřeby je editovat (není potřeba a není ani žádoucí jedno okno zavřít).

Text *Start page* kliknutím označíme a stisknutím klávesy delete jej pak můžete vymazat.

Na obrazovce bude potřeba vytvořit nové objekty - to lze provést z kontextového menu plochy kliknutím na položku *Nový prvek* nebo stisknutím klávesy *N*.

Nové prvky se přidávají z *palety prvků*, viz obr. 3.12.

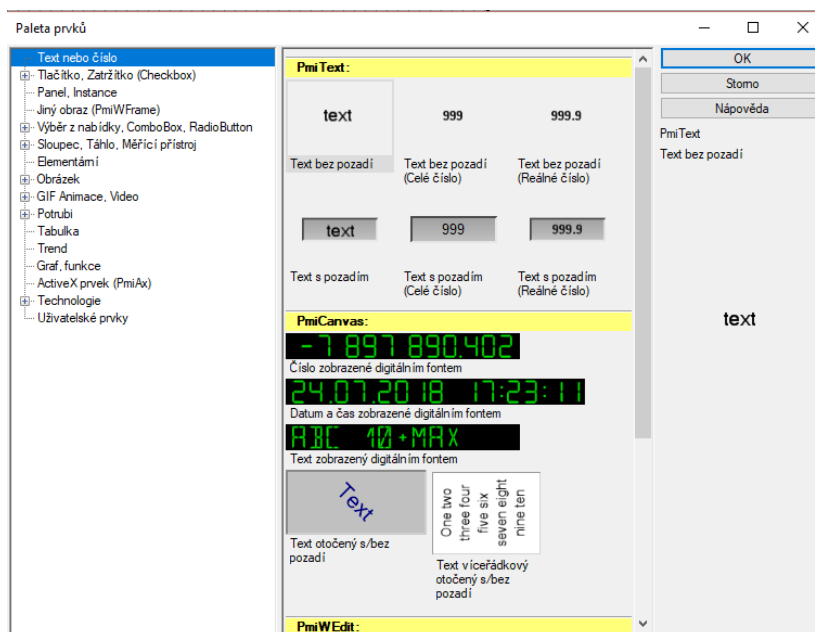
Jednotlivé prvky jsou seskupeny podle typů. Jelikož je dostupných prvků k dispozici opravdu velké množství nebudeme podrobně diskutovat všechny - zaměříme se pouze na ty, které bychom mohli chtít použít nejčastěji.

- text nebo číslo - dané prvky umožňují zobrazovat informaci v textové podobě
- tlačítka a zatržítka jsou v praxi často používaná, k jejich zprovoznění je ale potřeba programování, proto je v tomto textu používat nebudeme
- sloupec, táhlo, měřicí přístroj - nabízené prvky umožňují vizualizovat v grafické podobě číselnou veličinu a případně (pomocí táhla) měnit její hodnotu.
- obrázek - obrázky ve vektorovém formátu (PmiCanvas) nebo ve formě bitmapy (PmiRastr), v obrázky - uživatelsky definované je přístupná grafika uložená ve složce Resources projektu
- GIF, animace, video - podobně jako obrázky, ale s animacemi
- potrubí
- graf, funkce - umožňuje vizualizovat vývoj sledované veličiny v čase

Pro zpracování vizualizace činnosti UPS vytvoříme dva měřicí přístroje pro baterii a napětí v elektrické síti a dvě táhla pro dolní a horní mez napětí.

Měřicí přístroje je možno nalézt v panelu nástrojů v sekci Sloupec, táhlo, měřicí přístroj → Měřicí přístroj. Použijeme kruhový měřicí přístroj (hned ten první) pro napětí v síti a pro baterii pak měřicí přístroj (viz obr. 3.16).

PmiCanvas je vektorový, proto je možné jej libovolně škálovat beze změny v kvalitě zobrazení. Zároveň ale manipulace s tímto typem je náročnější - tedy pokud chcete nastavit nebo změnit funkci prvku, která není nabízena v průvodci vytvoření prvku.



Obrázek 3.12: Editor grafického obsahu - paleta prvků

Zkusme nadefinovat první přístroj pro měření napětí v síti. Dvojklikem vybereme měřicí přístroj (alternativně kliknutím vybrat a kliknout na tlačítko OK). Spustí okno s nastaveními Kruhový měřicí přístroj (viz obr. Y). Nastavení:

- měřená jednotka: V
- měřená hodnota: 230 - pozor, tady budeme připojovat proměnnou „napeti“
- rozsah od minima: 0
- rozsah do maxima: 300
- zobrazit technologické meze: zaškrtnuto
- technologické minimum: 200 - pozor, tady budeme připojovat také proměnnou „dmn“
- technologické maximum: 260 - pozor tady budeme připojovat také proměnnou „hmn“
- hlavní dílky stupnice: 15
- vedlejší dílky stupnice: 3

Připojení proměnných provedeme tak, že klikneme na oranžové tlačítko vedle dané hodnoty. Následně vybereme PP lvlastnost PmObjektu a v pravé části obrazovky v položce Objekt klikneme na tlačítko ... Teď už stačí vybrat pouze PmRoot → Data → #vars → napeti (viz obr. 3.13) a volbu potvrdit OK, potom znovu OK až bude viditelné pouze okno vlastností Kruhového měřicího přístroje.

Analogicky připojíme také proměnné technologického minima a maxima. Které proměnné budeme připojovat do kterých vlastností je patrné v odrážkách výše. Vyplněné vlastnosti by měly vypadat podobně jako na obr. 3.14 (podle toho, jak jste přesně pojmenovali proměnné a kam jste je umístili).

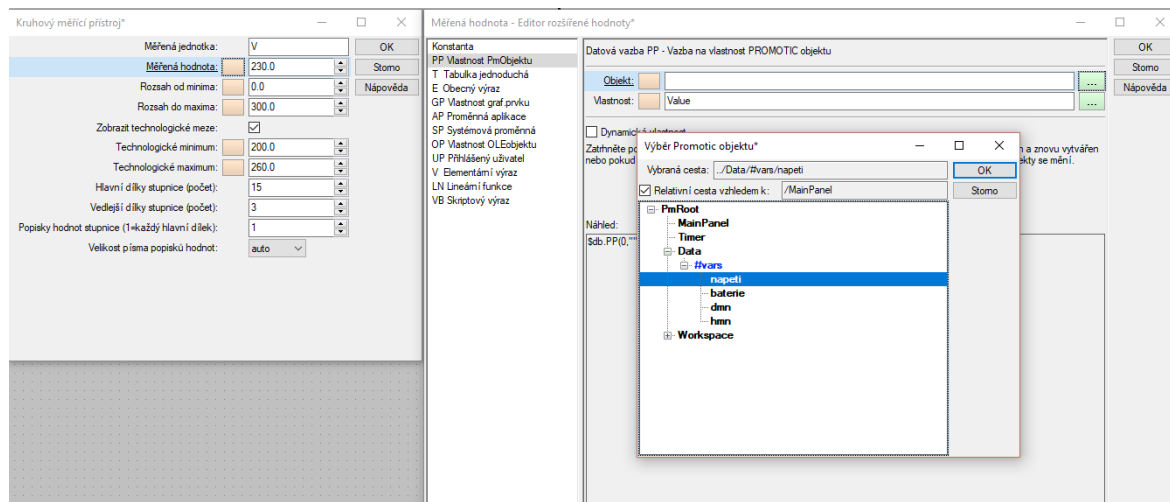
Pokud jste s vyplněním spokojeni - klikněte na OK. Prostor pro vytvoření prvku označíte na pracovní ploše kliknutím a tažením. Velikost prvku a jeho umístění je možno kdykoliv změnit. (Manipulace s prvky je stejná jako u kteréhokoliv grafického editoru).

Dvojitým kliknutím na vytvořeném objektu se dostaneme do jeho vlastností. Nejdůležitější jsou záložky Proměnné, obsahující připojené proměnné a Obsah - v obsahu je možné přenastavit vlastnosti z obr. 3.14. Obsah je ale v tomto případě dostupný pouze ve formátu XML a proto najít kde přesně se daná vlastnost nachází je poněkud náročnější.

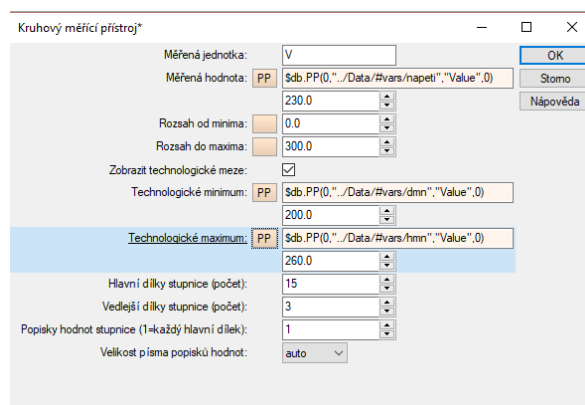
Z hlediska rychlosti návrhu vizualizace je proto žádoucí, aby objekty byly vytvořeny správně na první pokus.

Z hlediska vizualizace napětí v síti by také bylo výhodné vidět aktuální hodnotu proměnné „napeti“ i v textovém formátu a hodil by se také popisek specifikující jaká veličina je vlastně měřena. Obojí lze jednoduše realizovat pomocí prvku *Text nebo číslo*, pro obojí zvolím text bez pozadí.

Text má pouze omezené množství vlastností, které má smysl vyplňovat. Vlastnost text v případě popisku nastavíme na nápis „Napětí v elektrické síti“ a umístíme jej nad vytvořený měřicí přístroj.



Obrázek 3.13: Kruhový měřicí přístroj - připojení proměnné



Obrázek 3.14: Kruhový měřicí přístroj - nastavení vlastností

Případně lze změnit vlastnost Písmo a vybrat nějaké výraznější.

Pro vizualizaci hodnoty bude potřeba realizovat připojení na proměnnou „napeti“ a to konkrétně ve vlastnosti Text. Postupovat budeme stejným způsobem jako v případě nastavení měřicího přístroje. Původní vyplnění vlastnosti text můžeme ponechat, alespoň se nám bude s prvkem lépe manipulovat.

Prvek umístíme na měřicí přístroj, podobně jako na obr. 3.15 (nad V).

Z hlediska umísťování objektů platí, že novější objekty jsou více ve předu - což je v tomto případě využito.

Vytvoření měřidla pro baterii je již jednodušší. Nastavit je zde potřeba pouze měřenou hodnotu - připojíme k ní proměnnou „baterie“. Z hlediska estetičnosti můžeme ještě přerozdělit hlavní a vedlejší dílky stupnice. Já jsem pro výsledek na obr. 3.16 zvolil 5 hlavních dílků a 2 vedlejší dílky stupnice.

Nad měřidlo umístíme popisek - Baterie [%].

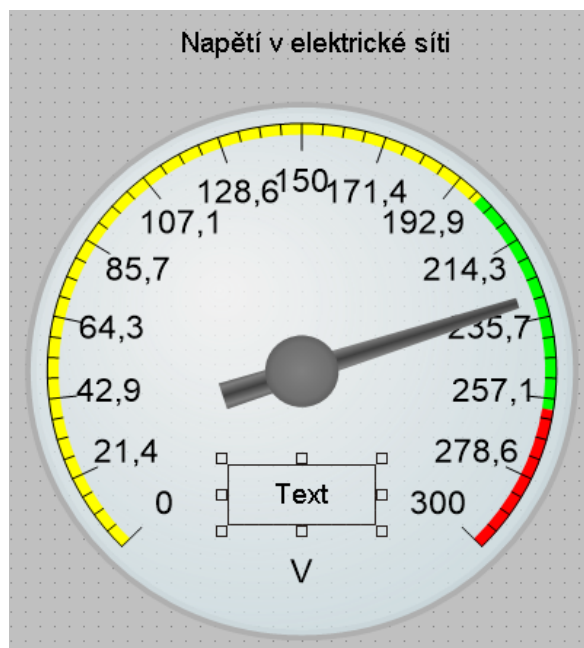
V obraze už zbývá pouze vytvoření dvou táhel pro realizaci dolní a horní meze napětí. Táhlá jsou dostupná Sloupe., táhlo, měřicí přístroj → Táhlo. Pro náš účel se hodí třeba táhlo se stupnicí, vodorovné.

Ani tady není potřeba nastavovat příliš velké množství vlastností. Nadpis nastavíme na Dolní mez napětí [V], do hodnoty připojíme proměnnou „dmn“, dolní mez dáme třeba 170 a horní 230.

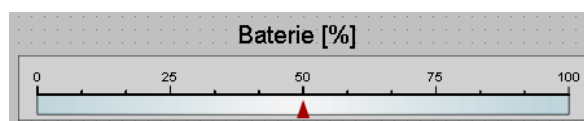
Analogicky pak vytvoříme i táhlo pro horní mez napětí. Správně nastavení a propojení byste již měli být schopni realizovat sami. Celá vizualizace by měla vypadat podobně jako na obr. 3.17.

Aby byl obraz funkční, je potřeba provést tzv. kompilaci obrazu. Kompilaci je možno provést z menu Projekt → Překlad ..., nebo klávesovou zkratkou CTRL + P a nebo pomocí tlačítka na panelu nástrojů (hned vedle ikony diskety).

Kompilovat je možné buďto pouze nekompilevané nebo změněné obrazovky, nebo všechny obra-



Obrázek 3.15: Kruhový měřicí přístroj - doplnění textových popisků



Obrázek 3.16: Vizualizace baterie

zovky. V našem případě pracujeme pouze s jednou obrazovkou, takže časový rozdíl obou operací bude zanedbatelný. U větších projektů tomu tak ale být nemusí.

V běhovém prostředí budou dostupné pouze kompilované obrazy. Běhové prostředí přitom spouští poslední kompilovanou verzi obrazu, takže pokud to co vidíte v editoru obrazů a v běhovém prostředí je odlišné, je potřeba provést opětovnou kompilaci obrazu.

Po spuštění překladu se Vám na obrazovce objeví výsledné hlášení, v našem případě by překlad obrazu měl proběhnout v pořádku. V případě ale, že bychom někde udělali chybu, mohl by překlad selhat. Chybové hlášení by Vám pak mělo říci, kde přesně se chyba nachází.

Podle mých zkušeností s projekty se nejčastěji chybí v propojování proměnných. Může se např. stát, že v základním rozhraní Promotic změníte názvy proměnných, ale změnu už třeba neprovedete v editoru obrazů - tím se naruší propojení mezi navrhovanou obrazovkou a systémem Promotic.

Nyní již můžeme editor obrazů zavřít. Aplikaci je možno spustit z hlavic aplikace Promotic kliknutím na tlačítko zeleného trojúhelníku v panelu nástrojů (tlačítko spustit aplikaci). Naše aplikace je plně funkční ... ale nic nedělá, protože jí chybí logika fungování technologie. Jednotlivé proměnné tak zůstávají statické.

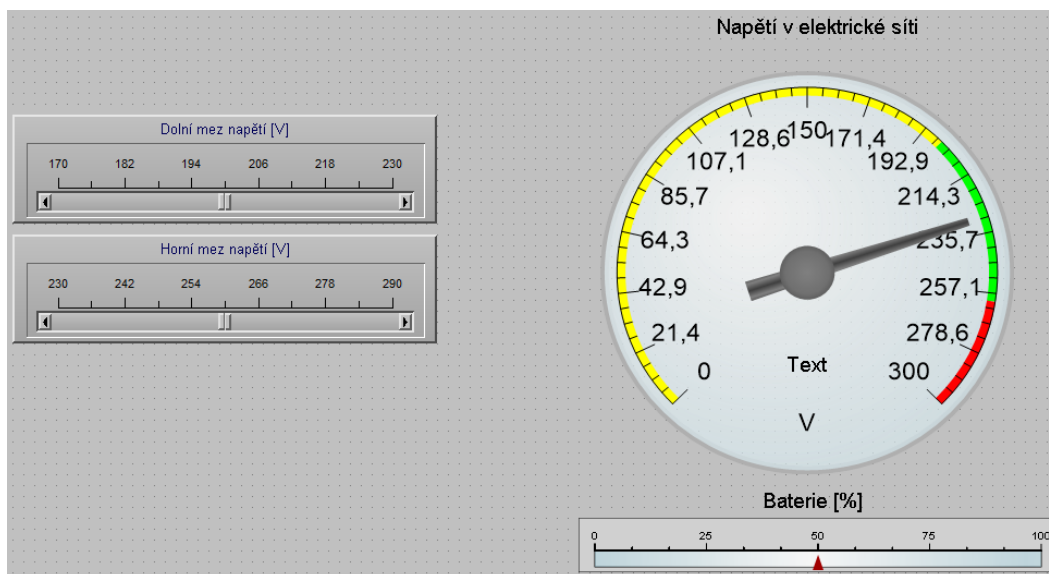
Tento problém vyřešíme v následující podkapitole vytvořením jednoduchého skriptu v časovači.

3.4 Časovač - programování

Logiku fungování již máme vymyšlenou, viz předchozí podkapitoly.

Skript vytváříme na záložce Události časovače, konkrétně pak v události onTick, tedy česky po tiknutí časovače. Časovač pracuje tak, že spustí v určeném intervalu (na záložce časovač, vlastnost perioda) skript v události onTick.

Proměnné které se zde ale použijí nejsou tzv. perzistentní - to znamená, že jejich hodnoty mezi jednotlivými spuštěními nebudou zachované.



Obrázek 3.17: UPS - vizualizace, celek

Z tohoto důvodu musíme provést mapování proměnných Visual Basic na Proměnné systému Promotic. Celý skript je přiložen níže:

```

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

```

Zkusme projít zdrojový kód řádek po řádku. Na začátku jsou nejprve deklarované proměnné pomocí příkazu **dim**. Za příkazem **dim** vždy následuje jeden nebo více proměnných. V případě použití více proměnných, používáme oddělovat čárku.

Visual Basic přitom nebere v úvahu velikost písmen, proto zápisy **dim** proměnná nebo **DIM** **PRO****MĚNNÁ** nebo **DiM** **ProMěNNá** (jakákoliv jiná kombinace velikosti písmen) jsou funkčně ekvivalentní.

Proměnné jsou deklarovány na dvou samostatných řádcích spíše z důvodu snadnější čitelnosti kódu než čehokoliv jiného. V prvním řádku jsou deklarovány proměnné, do kterých budou mapovány

proměnné Promotic, v řádku druhém jsou deklarovány pracovní proměnné, které vyžaduje ke své práci skript.

Mapování proměnných se děje pomocí příkazu **set**. Pomocí tohoto příkazu přiřadíme do proměnné odkaz na objekt, popřípadě vlastnost objektu. Řádek: `set napeti = Pm("/Data/#vars/napeti")` tak do proměnné `napeti` přiřadí proměnnou Promoticu nazvanou `napeti` z objektu `Data`.

Prosím při psaní vlastního kódu věnujte pozornost obsahu funkce **Pm** - pokud uděláte chybu v cestě k proměnné, nebude se jednat o syntaktickou chybu a samotné vývojové prostředí ji proto neodhalí (bude se tvářit, že je vše v pořádku). Po spuštění ale nebude aplikace pracovat správně.

Průvodním znakem této chyby je, že změny hodnot veličin měněných kódem se nezobrazují ve vizualizaci po spuštění.

Komentáře jsou označovány znakem apostrofu („‘“). Vše, co následuje za znakem apostrofu je považováno za komentář a nebude vykonáno.

Funkce **rnd** vygeneruje náhodné desetinné číslo v intervalu 0 a 1. Ve skriptu je tato funkce použita pro simulaci náhodného kolísání proudu v elektrické síti. Jako hraniční je použita hodnota 0,5. Vyhodnocení je pak realizováno podmínkou.

Podmínku lze ve Visual Basicu realizovat příkazem **if**. Podmínky mohou být buďto jednořádkové nebo víceřádkové. Příkladem jednořádkové podmínky může být řádek 11. Formát podmínky je potom následující: Pokud podmínka pak příkaz ... vše na jednom řádku.

V některých příkladech, ale takový zjednodušený zápis není dostatečný. V takovém případě přichází na řadu víceřádková podmínka:

```

1  if podminka then
2      ... prikazy ...
3  elseif jina_podminka then
4      'dodatecne podminky lze pomoci elseif pridavat
5      ... prikazy ...
6  else
7      ... prikazy ...
8  end if

```

Víceřádková podmínka je vždy zakončena `end if`.

Ve výpisech výše si všimněte také odlišného formátu použití proměnných - např. rozdílů v použití proměnné `n` a `napeti`. Zatímco proměnná `n` je používána tak, jak je, proměnná `napeti` je vždy používána v objektové notaci `napeti.Value`. To je dáno tím, že proměnnou `napeti` jsme nastavovali pomocí příkazu `set` a ve skutečnosti se proto jedná o objekt a objekty mají velké množství vlastností. Nás v tomto případě zajímá hodnota proměnné, proto voláme vlastnost objektu `Value` (hodnota).

K jednotlivým vlastnostem objektu je možno přistoupit pomocí tečky bezprostředně za objektovou proměnnou.

Náš kód je hotov. Před spuštěním celé aplikace je vhodné spustit test z kontextového menu. Test je schopen odhalit syntaktické chyby ve skriptu a označit místa, kde byly tyto chyby detekovány. Pokud byly zjištěny syntaktické chyby máte jistotu, že Váš skript nebude fungovat správně a proto je potřeba tyto chyby okamžitě opravit.

Pokud chyby detekovány nebyly bohužel nemáte jistotu, že Váš skript bude fungovat správně, protože může obsahovat také logické chyby, které automatizovaně detekovány nejsou.

Vhodným postupem pro začátečníka je zpracovávat kód po relativně malých částech, provádět pravidelně testování a spouštět aplikaci tak, aby bylo možno vizuálně zkontrolovat, že připravená část skriptu funguje.

Kód skriptu by bylo možné např. realizovat nejprve takto:

```

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  napeti.Value = 300
10 baterie.Value = 33
11 dmn.Value = 230
12 hmn.Value = 230
```

Provedli jsme deklaraci proměnných a provedli jejich připojení na proměnné Promotic, následně jsme ale do takto spojených proměnných zadali odlišnou hodnotu. Pokud takovou aplikaci spustíme vizuálně uvidíme, zda se přenastavení hodnot proměnných provedlo nebo ne. Pokud ne, pak chyba musela nastat buď v přiřazení objektu set nebo v přiřazení hodnoty.

Protože ale zároveň uvidíte, která hodnota nebyla přiřazena omezíte možný problém pouze na dva řádky. Když získáme jistotou, že objekty se přiřazují správně, je možné řádky 9 - 12 odstranit a pokračovat v další práci.

Postupujte po malých krocích (rozděľujte problém na malé, jednoduché dílčí problémy).

Dbejte také na použití tzv. bílých znaků. Jedná se např. o tabulátory. Všimněte si, např. jak jsou naformátovány víceřádkové podmínky. Tabulátor umožňuje jednoznačně určit „tělo“ podmínky. Bílé znaky nejsou povinné, ale jejich použití zlepšuje výrazně čitelnost kódu, protože se v něm dá lépe orientovat. Dobře čitelný kód se pak lépe ladí.



Místo shrnutí

Tato kapitola byla více praktická, doufám proto, že jste příklad úspěšně realizovali v Promoticu a pochopili základy fungování tohoto systému. Teď je vhodná doba se systémem dále experimentovat.

V kontrolních otázkách proto najdete některé náměty, které by Vás mohly motivovat



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.

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 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 - je proto zranitelný běžnými útoky
- typické je dálkové řízení
- některé informace získávané nebo zpracovávané v automatizační části sítě organizace je nutné sdílet i do běžné počítačové sítě organizace.

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 rozebrány bezpečnostní implikace, které z nich lze vyvodit. 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 dávné době - v roce 2000. Tehdy se do čističky odpadních vod v Maroochy Shire, Queensland - Austrálie [64] 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 prostředí.

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 řízení technologií. Technicky je takový síťový provoz pak obtížněji detekovatelný než pokus realizovaný bez takových znalostí a nástrojů.

Problematická je také rychlost provedení útoku - v případě, že se útočníkovi podaří k řízené technologii připojit, může ji začít přímo ovládat část nebo dokonce celou řízenou technologii. Běžný útoční 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 tak neměl mít možnost se vůbec připojit do automatizační sítě.

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 a takový pracovník by pak měl ztratit možnost se vzdáleně připojit.

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 [66] 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 zařazeny i jisté dopady, které rámec běžného IT přesahují. Vyřazena byla např. činnost několika aerolinek [22], 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.

V atomové elektrárně Davis-Besse v Ohiu (USA) [58] 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.

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].

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.

Ukazuje se, že člověk je tvor omylný a tak všechny jeho výtvořky je obsahují také. V IT se to týká jak software tak hardware. Hlavním poučením, které si lze z případu Slammer vzít, je že se nevyplácí podcenit **management rizik**.

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 [63] 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 [39] a Meltdown [43] 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.

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 [60]. Škody se v tomto případě odhadují poměrně obtížně, ale práci muselo přerušit okolo 50-ti tisíc zaměstnanců společnosti.

K infekci došlo být automatizační síť byla za firewallem. 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 [69] na Britský **National Health System (NHS)**. Podle šetření **National Audit Office (NAO)** [50] 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 přerušování útoku.

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.

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)** [41] 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) [59]. 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 ale také zabránil dálkovému zásahu operátora.

Projevila se tak chyba v zařízení, která za normálních okolností zůstávala latentní. Jelikož obě oběhová čerpadla byla pořízena ve stejnou dobu od stejného výrobce, byla také náchylná ke stejným problémům. Nepříznivá kombinace faktorů která zapříčinila výpadek hlavního čerpadla způsobila také výpadek jeho zálohy.

Řešení v tomto případě jsou poměrně problematická. Lze uvažovat o pořízení odlišných záloh, 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.

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 příčiny 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 rozsáhlejším výpadkům v síti jako celku.

Teoretické zdůvodnění toho, proč tomu tak je je možné nalézt např. v Lewis [42]. 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.

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 [29] byl specifický svým zaměřením. Kompromitoval počítače s operačním systémem Windows. K tomu využíval 4 v době útoku 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.

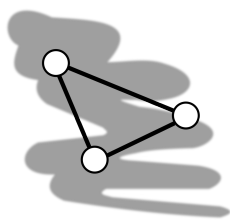
Pozdější analýzou binárního kódu červa a přítomných omezení bylo zjištěno, že hlavním cílem červa byla pravděpodobně zařízení v Natanz v Iránu, které sloužilo k obohacování uranu. Hlavním cílem bylo přitom pravděpodobně napadené PLC automatů v síti ovládající centrifugy na obohacování uranu. Změnou rychlosti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

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 asi 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* [73].

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, zda existují také jiní červi, kteří sdílejí podobné vlastnosti/moduly.

Stuxnet by pak mohl být považován za útočnou variantu červa, kterou by mohly doplňovat varianty zaměřené např. na špionáž. Ukázalo se, že takové varianty skutečně existují.

Červ DuQu [67] 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) [68] 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 [65] 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.



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 neznamena, 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 jednoduše definovatelná (slovník Merriam-Webster [45]) 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 [56] tvrdí, že kybernetická válka je ve skutečnosti protimluv. Argumentuje, že použití v současnosti známých nástrojů má tak 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 [54]. 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 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 vznikala prostor pro objevení již kompromitovaných systémů a jejich oprava do původní podoby. 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 nikdy nebyl realizován.

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 síť další 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) kontraktora.

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.

Ochrana pak logicky musí být založena na zajištění ochrany *vnějšího perimetru organizace*. Mezi opatření pak 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.

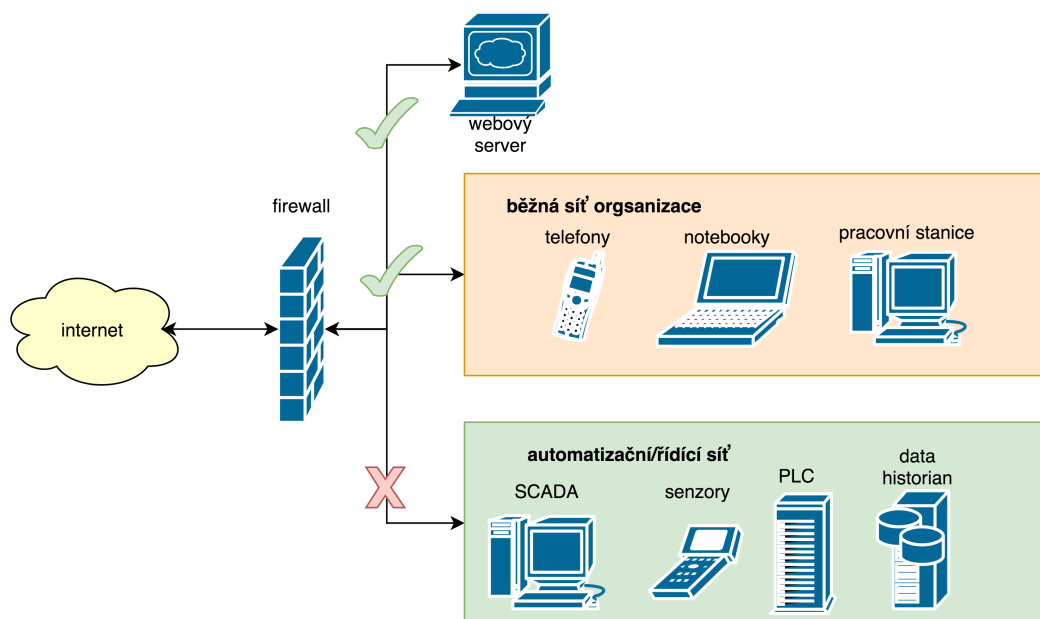
Výše uvedená opatření mají za cíl zvýšit náročnost provedení útoku, popř. prodloužit čas nutný pro jeho realizaci a tak zvýšit šanci, že dojde k jeho odhalení a následně přerušení. Cílem tedy není provedení útoku zcela zamezit - neboť úplné zabezpečení není technicky možné nikdy realizovat.

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 firewalllem 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 (řídicí) 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 a pak může pokračovat neomezeně v útoku do zbytku sítě i té, kde byla komunikace s vnějším světem na firewallu zakázána.

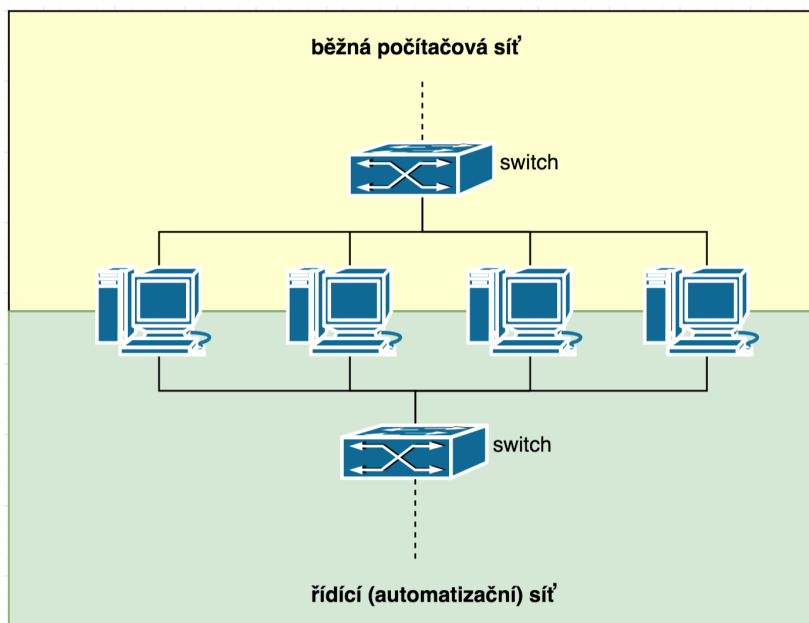
Úroveň ochrany, kterou toto řešení poskytuje tak nelze přeceňovat. Z toho důvodu je tento scénář ochrany volen samostatně pouze zřídka a to zejména v případech kdy oddělení sítí nemá vzhledem k velikosti smysl (např. mikro firmy, malé firmy).

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

Dual home computers

Tímto ne úplně dobře do češtiny přeložitelným označením rozumíme pracovní stanice, které jsou připojeny zároveň do řídicí 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.

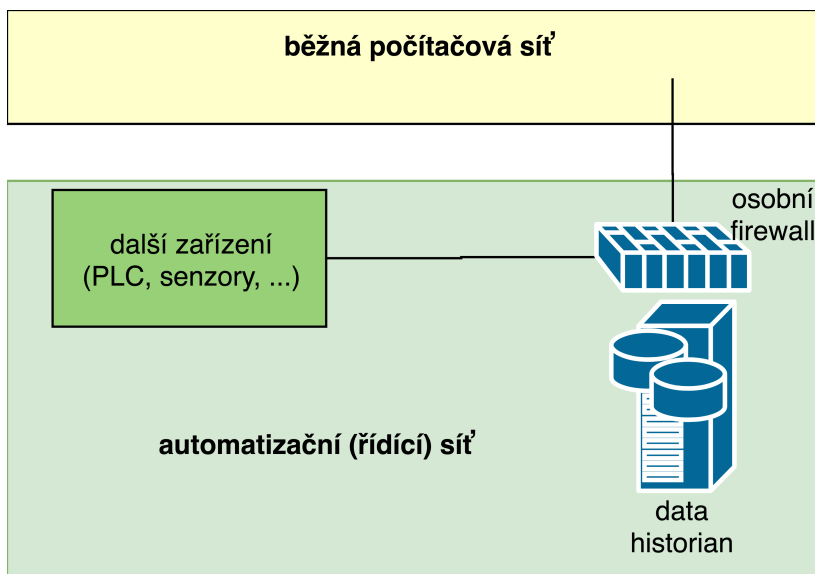
Podobná situace je s možností škálování řešení - např. přidávání dalších počítačů s touto konektivitou.

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 sítě podniku.

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 místo, kam se takové údaje často ukládají - *data historian*.



Obrázek 4.3: Dual home server

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

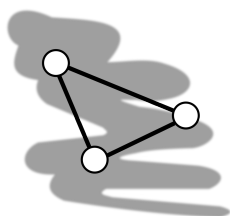
Toto řešení je ve srovnání s umisťováním jednotlivých počítačů do dvou sítí lépe škálovatelné. Databázový server bude nejspíše umístěn v serverovně, u které se dá očekávat, že její přesun nebude probíhat právě často. Dále se jedná pouze o jedno místo, jeho zasíťování není proto až tak problematické.

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. 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. whitelingu aplikací (specifikace aplikací, které mohou být spouštěny, popř. 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* [23] 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 také některé problémy. Asi největším problémem je, že osobní firewally jsou určeny pro „osobní použití“ - tedy předpokládají, že u chráněného zařízení bude sedět fyzická osoba, která v případě potřeby zasáhne do funkce firewallu.

Osobní firewally se 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 v této situaci dělat - např. povolit dané aplikaci možnost komunikovat po síti. Na základě interakce s uživatelem jsou pak doplňována pravidla.

U serveru ale za normálních okolností není fyzická obsluha - většina přístupů a to i administrátorských zásahů je realizována dálkové prostřednictvím na serveru provozovaných služeb. I osobní firewall lze nakonfigurovat tak, aby se na nic neptal. Sada použitých pravidel je ale v takovém případě omezenější. Plnohodnotné použití všech (zejména těch pokročilejších funkcí jako je sandbox nebo HIPS) pak často není možná.

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.

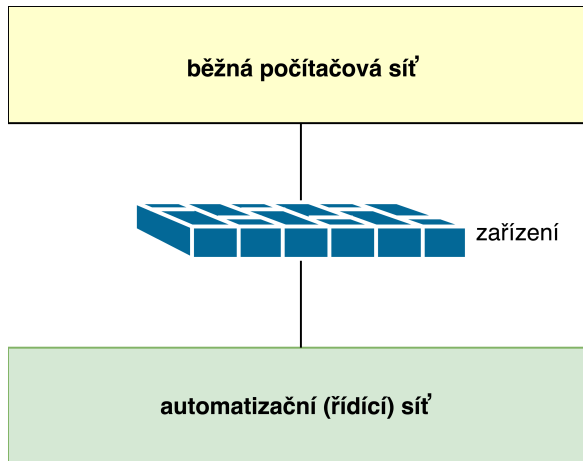
Poslední poznámka - z hlediska funkce tohoto řešení je v zásadě jedno, zda dual home server je umístěn v řídicí síti (jak je tomu na obr. 4.3) nebo by byl umístěn v běžné síti. Řešení by fungovalo analogicky - pouze v obráceném gardu než je znázorněno na obrázku.

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.

Switch se pak stará primárně o funkci přepínání - tedy zjednodušeně řešeno přeposílá síťovou komunikaci do cílových portů. V případě tzv. „spravovatelných“ switchů lze zasáhnout do konfigurace a ovlivnit způsob, jakým se toto přeposílání děje.

K tomu je však potřeba dodat, že switch pracuje je linkové vrstvě síťové architektury a proto nemá schopnost rozlišovat např. mezi jednotlivými službami. Povoluje se, popř. zakazuje tak komunikace identifikací IP adresy zařízení.

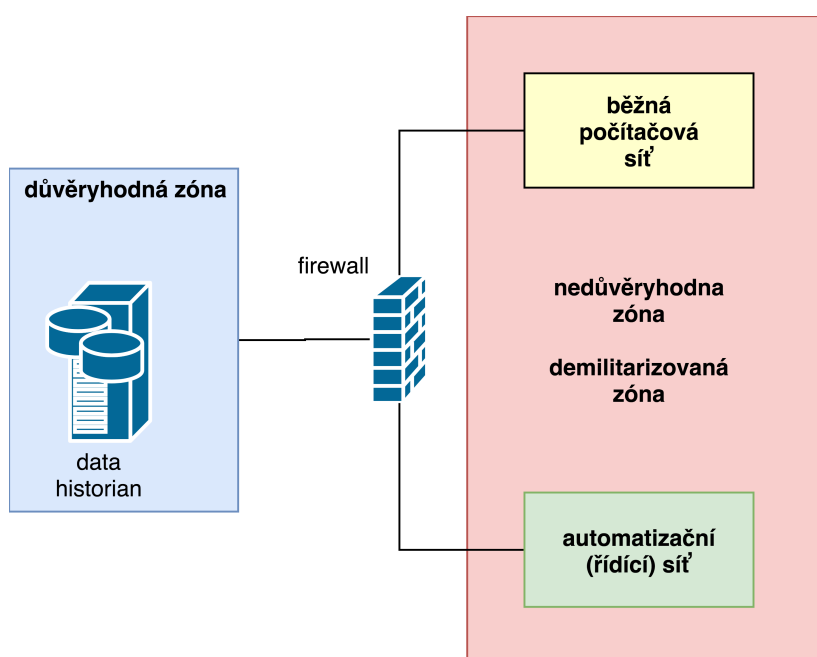
Ani router, ani switch přitom nejsou určeny primárně k filtraci síťového provozu - tento typ použití je tak pouze sekundární a lze jej použít tam, kde se nevyplatí (z různých důvodů) použití firewallu.

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.

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

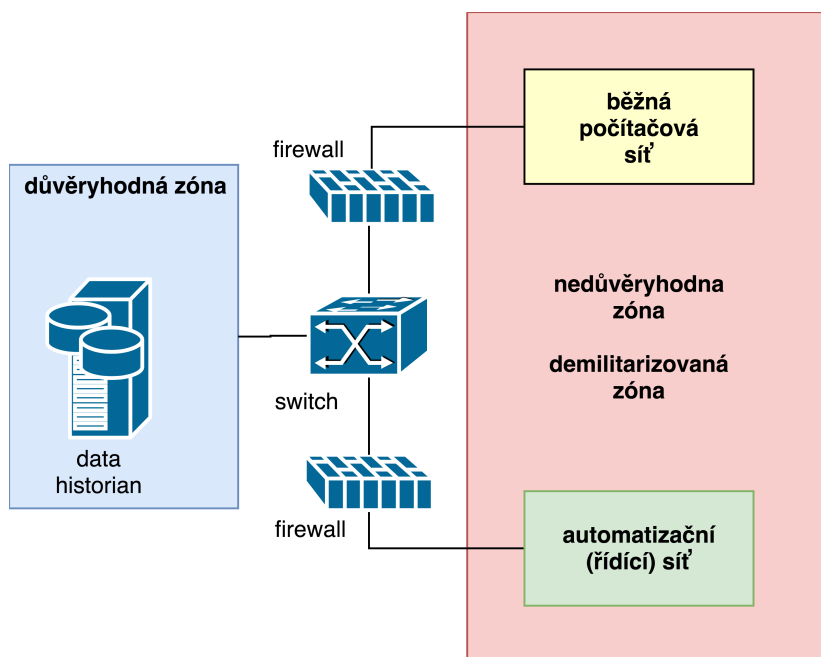
Řešení z obr. 4.5 předpokládá, že žádné ze sítí není možné bez výhrady věřit z toho důvodu je žádoucí obzvláště důležitá zařízení vyčlenit do důvěryhodné zóny a přesně stanovit nastavením pravidel firewallu, která zařízení s použitím kterých služeb mohou s ním komunikovat. Zbývající zařízení uvnitř sítě jsou tedy nedůvěryhodná. 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

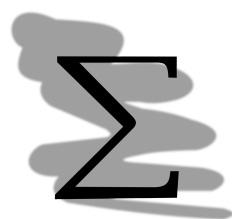


Obrázek 4.6: Důvěryhodná vs demilitarizovaná zóna - řešení s použitím dvojice firewallů

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 - 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.

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.



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 řešené 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 očekávaných 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 firewallem?
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 jí 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 podporu ochrany **Industrial Control System (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 alespoň státy sponzorované. Vzhledem k tomu, že infrastruktura samotná je pak v soukromých rukou vzniká zde nepoměr v dostupných zdrojích z hlediska možnosti financovat útok na infrastrukturu 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 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 zkratka pro Computer Emergency Response Team, tedy česky tým určený 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 než je tomu 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 byla obecná podpora. Podobný přístup a také označení takových týmů bylo později přejímáno také jinými organizacemi. Řada z nich se ale rozhodla nepoužít pro tým označení **CERT**, ale **CSIRT**.

CSIRT je zkratka pro Computer Security Incident Response Team - česky tedy 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 otevřelo firmám i jednotlivcům netušené možnosti, na straně druhé obdobné možnosti přinesla 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 **Denial of Services (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 [5] 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.

Řada států realizuje tzv. národní a vládní CERT týmy. V případě české republiky plní funkci národního CERT týmu tým CSIRT.CZ [24], funkci vládního CERT týmu plní pak GovCERT.CZ [53].

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.

Poznámka:

Všimněte si používání pojmů CERT a CSIRT. Ve výše uvedeném textu se může zdát, že jsou zaměño-

vány 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 [4], který explicitně hovoří o národním CERT týmu a vládním CERT týmu.

Hlavní cíle CSIRT.CZ jsou následující [24]:

- 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)** [32] a **FIRST** [30]. TI bylo zřízeno v roce 2000 evropskou komunitou CERT a CSIRT týmů [32]: *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 [32]:

- 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 [30]:

- 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)** [27], 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 [28] 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 není neomezená - její fungování je v pravidelných intervalech přehodnocováno, což může vést k poměrně razantním změnám nebo dokonce zrušení agentury. Další kolo hodnocení je připravováno na rok 2018.

Všeobecné očekávání však je, že role ENISy by se do budoucna měla spíše rozšířit. Některé spekulace vedou až k chápání ENISy jako vrcholového CERT týmu pracujícího na úrovni EU jako celku. Návrh nové regulace činnosti ENISy byl rozeslán jednotlivým členským státům k připomínkování [26]. V současnosti (léto 2018) jednotlivé státy stále vznášejí k návrhu připomínky, byť připomínkové řízení se pomalu chýlí ke konci.

Z dosavadních ohlasů lze usuzovat, že nová regulace bude přijata ve stavu velmi podobném návrhu, jelikož většina připomínek nezpochybňovala samotná opatření, ale směřovala spíše k vyjasnění některých pasáží textu.

Podle návrhu by ENISA měla získat permanentní mandát a rozsah jejích činností by se měl rozšířit. Nejdůležitějším novým úkolem 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.

Schémata samotná by pak měla odpovídat mezinárodně uznávaným normám. Hlavním cílem je tedy rozvoj společného trhu v této oblasti.

GovCERT.CZ má za sebou poměrně krátkou, avšak pestrou minulost. Původně GovCERT.CZ provozoval **Národní bezpečnostní úřad (NBÚ)** v rámci **Národní centrum 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 hlavním náplní práce NÚKIB patří [51]:

- 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 [52]:

- 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, prostředkování 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 síť, 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 laboratoř 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

Zákon o kybernetické bezpečnosti (181/2014 Sb. [4]) vešel v platnost 1. 1. 2015. Během svého relativně krátkého života však již byl jednou poměrně významně novelizován (v roce 2017) 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.

Předtím než začneme rozebírat samotný zákon podívejme se na celkové legislativní řešení kybernetické bezpečnosti. Její základ tvoří *zákon o kybernetické bezpečnosti*. Kromě něj existuje i řada dalších prováděcí vyhlášek, popř. dalších sovisejících vyhlášek a nařízení vlády:

- Vyhláška č. 82/2018 Sb. o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních, náležitostech podání v oblasti kybernetické bezpečnosti a likvidaci dat (vyhláška o kybernetické bezpečnosti) [17]
- Vyhláška č. 317/2014 Sb. o významných informačních systémech a jejich určujících kritériích [12]
- Vyhláška č. 437/2017 Sb., o kritériích pro určení provozovatele základní služby [14]

Zákon o kybernetické bezpečnosti

Zákon 181/2014 Sb. o kybernetické bezpečnosti řeší řadu problémů, které postupně budeme rozebírat. Zde je potřeba upozornit také návaznost na další prováděcí předpisy k zákonu, které probíranou problematiku dále rozvádějí. Některé z těchto předpisů jsou stručně popsány také v těchto skriptech.

Prvním problémem jsou tzv. *významné informační systémy*, ty jsou definovány zákonem (§2 písm. d): informační systém spravovaný orgánem veřejné moci, který není kritickou informační infrastrukturou a u kterého narušení bezpečnosti informací může omezit nebo výrazně ohrozit výkon působnosti orgánu veřejné moci.

Pro pochopení toho, co přesně výše uvedené znamená je potřeba definovat další pojem a to *Kritická informační infrastruktura (KII)*. Ta je definována následovně (§2, písm g): prvek nebo systém prvků kritické infrastruktury v odvětví komunikační a informační systémy v oblasti kybernetické bezpečnosti.

KII je tedy řešena jako kritická infrastruktura, včetně určování dle Nařízení vlády 432/2010 Sb., významný informační systém se ale určuje podle vyhlášky 317/2014 Sb.

Laicky je možno si to zdůvodnit tak, že kritická infrastruktura je realizována nějakým fyzickým prvkem, oproti tomu informační systém je software. Software sice běží na nějakém hardware, ale lze jej v případě potřeby relativně jednoduše přesouvat. Řídicí systémy, SCADA, síťovou infrastrukturu, které řadíme do KII, ale takto přesouvat nelze.

Zákon stanovuje okruh osob, kterým ukládá povinnosti v oblasti kybernetické bezpečnosti:

- a) poskytovatel služby elektronických komunikací a subjekt zajišťující síť elektronických komunikací, pokud není orgánem nebo osobou podle písmene b),
- b) orgán nebo osoba zajišťující významnou síť, pokud nejsou správcem nebo provozovatelem komunikačního systému podle písmene d),
- c) správce a provozovatel informačního systému kritické informační infrastruktury,
- d) správce a provozovatel komunikačního systému kritické informační infrastruktury,
- e) správce a provozovatel významného informačního systému,
- f) správce a provozovatel informačního systému základní služby, pokud nejsou správcem nebo provozovatelem podle písmene c) nebo d),
- g) provozovatel základní služby, pokud není správcem nebo provozovatelem podle písmene f), a
- h) poskytovatel digitální služby.

S některými pojmy jsme se již seznámili (významný informační systém a *KII*), zaměříme se proto na pojmy zbývající.

Službami elektronických komunikací definuje zákon o elektronických komunikacích [8] (§2 písm. n) jako služby obvykle poskytované 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.

Základní službou se rozumí služba závislá na sítích elektronických komunikací, popř. informačních systémech a jejíž narušení by mohlo mít významné dopady na zabezpečení společenských nebo ekonomických činností v některém z odvětví:

1. energetika,
2. doprava,
3. bankovníctví,
4. infrastruktura finančních trhů,
5. zdravotnictví,
6. vodní hospodářství,
7. digitální infrastruktura,
8. chemický průmysl,

Výše uvedený seznam by Vám neměl být neznámý - jedná se více méně o přehled odvětví kritické infrastruktury. Kritéria pro určení provozovatelů základních služeb jsou specifikována ve vyhlášce 437/2017 Sb.

Konečně digitální službu se rozumí vybraná skupina služba informační společnosti [7] zaměřená na provoz on-line tržišť, internetových vyhledávačů a tzv. cloud computingu (ať už ve smyslu výpočetních zdrojů v cloudu nebo úložného prostoru).

Právě oblast digitálních služeb do zákona byla přidána teprve na základě požadavků směrnice NIS. Vycházelo se přitom z toho, že „v cloudu“ je v současnosti provozováno velké množství systémů provozovaných osobami uvedenými v seznamu povinných osob (a) - (e), provozovatelé cloudové infrastruktury však dlouho regulováni nebyli.

Nová pravidla proto specifikují některé náležitosti, na které při využívání takových služeb musí povinné osoby musí brát zřetel. Jedná se např. o **Service Level Assurance (SLA)** - dohodě o úrovni poskytovaných služeb, podmínek bezpečnosti provozu apod.

Povinné osoby musí pro bezpečný provoz regulovaných systémů realizovat tzv. *bezpečnostní opatření*. Tato opatření jsou buďto organizační nebo technická.

Do *organizačních opatření* řadíme systém řízení bezpečnosti, rizik, bezpečnostní politiky, řízení lidských zdrojů apod. Jedná se tedy opatření, která je nutno implementovat ve způsobu fungování organizace. Ze své povahy je ale realizace plných účinků těchto opatření dlouhodobější záležitostí. Zavedení plnohodnotného nového systému řízení bezpečnosti může trvat měsíce a pokud organizace pracuje s obzvláště rozsáhlých nebo složitými systémy může trvat i let několik.

Technickými opatřeními jsou opatření vedoucí ke zvýšení bezpečnosti v organizaci. I charakter těchto opatření může být velmi různorodý od úpravy způsobu jakým funguje ostraha objektů organizace až po zavádění sofistikovaných nástrojů pro detekci bezpečnostních incidentů apod.

Podrobnosti o struktuře bezpečnostní dokumentace stanovuje vyhláška 82/2018 Sb. o kybernetické bezpečnosti [17]. Postup podle tohoto předpisu je povinný pro osoby c) - f) z okruhu osob.

Osoby b) - f) mají taktéž povinnost ze zákona detekovat kybernetické bezpečnostní události. Pojem *kybernetická bezpečnostní událost* je pro vás nový. Zákon totiž rozlišuje mezi těmito událostmi a kybernetickými bezpečnostními incidenty.

Kybernetickou bezpečnostní událostí je dle zákona událost, která může způsobit narušení bezpečnosti informací v informačních systémech nebo narušení bezpečnosti služeb anebo bezpečnosti a integrity sítí elektronických komunikací.

Oproti tomu *kybernetickým bezpečnostním incidentem* je dle zákona narušení bezpečnosti informací v informačních systémech nebo narušení bezpečnosti služeb anebo bezpečnosti a integrity sítí elektronických komunikací v důsledku kybernetické bezpečnostní události.

Jaký je tedy rozdíl mezi událostí a incidentem. Událostí může být např. pokud neautorizovaně přistoupit k citlivým datům významného informačního systému. Incidentem v tomto kontextu může být v případě úspěchu takového přístupu získání citlivých dat z takového systému.

Prakticky vzato většina útoků na systémy organizací není úspěšná. Zákon, ale pro specifikované okruhy osob ukládá povinnost detekce událostí. Není přípustné tedy u takových systémů pouze reagovat na incidenty.

Pokud už k bezpečnostnímu incidentu dojde mají osoby b) - f) povinnost hlášení takových incidentů, liší se ale komu má být incident hlášen.

V případě dopadů na základní služby musí být hlášen incident přímo **NÚKIB**. Podobně incidenty týkající se **KII** nebo významných systémů (osoby c) - f) incidenty hlásí **NÚKIB**. Provozovatelé významných sítí (b) a digitálních služeb (h) hlásí incidenty národnímu CERT týmu.

Úřad je také oprávněn ukládat opatření okruhu osob za určitých podmínek. Opatření jsou rozlišována na:

- varování
- reaktivní opatření
- ochranná opatření

Varování může úřad vydávat bez omezení. Reaktivní a ochranná opatření se ale v plném rozsahu vztahují pouze na provozovatele významných systémů a systémů KII (osoby c) - f). Osobám a), b) pak může Úřad ukládat opatření pouze v případě že je vyhlášen stav kybernetického nebezpečí nebo některých nouzových stavů.

Úřad naopak nemůže ukládat opatření provozovatelům digitálních služeb.

Vraťme se ke *stavu kybernetického nebezpečí*. Rozumí se jím stav, ve kterém je ve velkém rozsahu ohrožena bezpečnost informací v informačních systémech nebo bezpečnost služeb elektronických komunikací anebo bezpečnost a integrity sítí elektronických komunikací, a tím by mohlo dojít k porušení nebo došlo k ohrožení zájmu České republiky ve smyslu zákona upravujícího ochranu utajovaných

informací [9].

Základním zájmem dle zákona o ochraně utajovaných informací a o bezpečnostní způsobilosti je zachování ústavnosti, svrchovanosti a územní celistvosti, zajištění vnitřního pořádku a bezpečnosti, mezinárodních závazků a obrany, ochrana ekonomiky a ochrana života nebo zdraví fyzických osob.



Stav kybernetického nebezpečí

Z dalších předmětů by Vám měly být známy ostatní *nouzové stavy*, jako jsou?

Pro úplnost stav kybernetického nebezpečí vyhláší ředitel NÚKIB vyvěšením na úřední desce Úřadu. Informace o vyhlášení se také zveřejňuje v celoplošném rozhlasovém a televizním vysílání. Stav se vyhláší na dobu nezbytně nutnou, nejdéle však na sedm dní. Tuto dobu lze ale prodlužovat a to i opakovaně. Celková doba trvání stavu kybernetického nebezpečí však nesmí být delší než 30 dní.

V průběhu vyhlášeného stavu pak Úřad informuje vládu o přijatých krocích k řešení příčiny/aktuálního stavu hrozby, která vedla k vyhlášení stavu kybernetického nebezpečí.

Vyhláška o kybernetické bezpečnosti

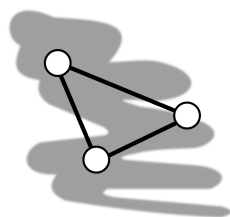
Vyhláška 82/2018 Sb. upravuje především:

- obsah a strukturu bezpečnostní dokumentace,
- obsah a rozsah bezpečnostních opatření,
- typy, kategorie a hodnocení významnosti kybernetických bezpečnostních incidentů,
- náležitosti a způsob hlášení kybernetického bezpečnostního incidentu,
- náležitosti oznámení o provedení reaktivního opatření a jeho výsledku,
- vzor oznámení kontaktních údajů a jeho formu a
- způsob likvidace dat, provozních údajů, informací a jejich kopií.

Vyhláška je prováděcím předpisem zákona o kybernetické bezpečnosti. Zákon (tak jak je obvyklé), ale specifikuje pouze že má být vedena bezpečnostní dokumentace, avšak nikoliv v jakém rozsahu nebo struktuře. Právě tyto otázky pak řeší vyhláška.

Předepsána jsou také organizační struktury, které by s ohledem na řízení kybernetické bezpečnosti povinné osoby měly být realizovány - např. role manažera, architekta, autora kybernetické bezpečnosti a existence výboru pro řízení kybernetické bezpečnosti.

Svou organizací filozoficky vyhláška vychází z doporučení kodexu norem ISO 27000 zaměřených na řešení tzv. **Information Security Management System (ISMS)**.



ISO 27000

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

- ISO 27001 [1] - systém **ISMS**
- ISO 27002 [2] - nejlepších praktik z oblasti bezpečnosti informací
- ISO 27005 [6] - rizika

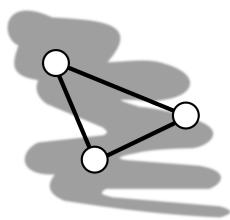
V tomto předmětu není prostor se touto problematikou zabývat ve větší míře, proto v případném studiu můžete pokračovat buďto samostatně nebo v rámci předmětu *Bezpečnosti informačních systémů*.

Vyhláška o významných informačních systémech a jejich určujících kritériích

Vyhláška 317/2014 Sb. řeší kritéria, podle kterých se určující 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 [11].

Dopadová kritéria hodnotí dopady úplné nebo částečné nefunkčnosti systému způsobené narušením bezpečnosti. Hodnotí se přitom dopady na provozovatele, případně poskytování služby nebo informací



Kritická infrastruktura

S problematikou kritické infrastruktury se můžete seznámit blíže v předmětu *Ochrana kritické infrastruktury*. Při studiu nezapomeňte, že kritická infrastruktura má svůj informatický rozměr.

orgány veřejné moci nebo další dopady mimo samotný systém. Spektrum takových dopadů je poměrně různorodé - může se jednat o narušení prvku kritické infrastruktury, počítat lze také se ztrátami na životech, zdraví, majetku nebo obecně veřejných zájmu.

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, vykonávat státní dozor, hospodařit, spravovat osobní údaje a další.

Úplný přehled oblastních kritérií je specifikován v příloze 2 vyhlášky.

Vyhláška obsahuje také seznam již určených významných informačních systémů a to konkrétně v příloze 1 vyhlášky. Seznam v současnosti (IX. 2018) obsahuje 153 významných informačních systémů. Ve vyhlášce je evidováno pořadové číslo systému, jeho provozovatel a název.

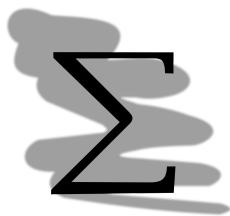


Významné informační systémy vs informační systémy veřejné správy

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 [3] a jeho prováděcími vyhláškami. V ČR je v současnosti provozováno dle Informačního systému o informačních systémech veřejné správy 6 633 [49] takových systémů, přičemž nejvíce je jich provozovány městy a obcemi v samostatné působnosti (1 818), přenesené působnosti (1 384) a ostatní (520).

Problematické **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 **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. Definujte významný informační systém.
2. Jaký je rozdíl mezi národním a vládním CERT týmem?
3. Co je poskytovatel digitální služby, uveďte příklad.
4. Jaký je rozdíl mezi událostí a incidentem (kybernetická bezpečnost).
5. Podle čeho se klasifikují významné informační systémy?



Správné odpovědi

1. IS provozovaný org. veřejné moci, u kterého by narušení bezpečnosti ohrozilo výkon působnosti tohoto org.
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 on-line tržišť, internetových vyhledávačů a cloudových služeb. Př.: Google, Microsoft, Amazon ...
4. Událost je obecnější, incident už představuje reálné narušení kybernetické bezpečnosti.
5. Podle dopadových (měří se dopad funkce veřejné správy, popř. systému KII) a oblastních (taxativně stanoveny vyhláškou) kritérií.

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.

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.

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

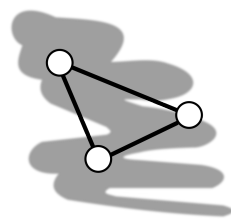
Představiteli takových nástrojů jsou např. SAS [62] od stejnojmenné společnosti a SPSS [36] od společnosti IBM. Obě tyto společnosti stály u zrodu data miningu a dokonce vyvinuly vlastní doporučené postupy realizace data miningu.

S postupem doby a rozvoji výpočetní techniky, ale objevovaly také alternativní cenově dostupné nástroje, které data mining zpřístupnili i malým a středním firmám nebo dokonce jednotlivcům. V

současnosti nejpoužívanější analytický nástroj celosvětově je R [16], 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 toolkitů poskytující nové funkce pokrývající prakticky všechny oblasti matematiky, statistiky, strojového učení apod. Dostupnost širokého spektra funkcí pak vedla k rychlé adopci systému 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.



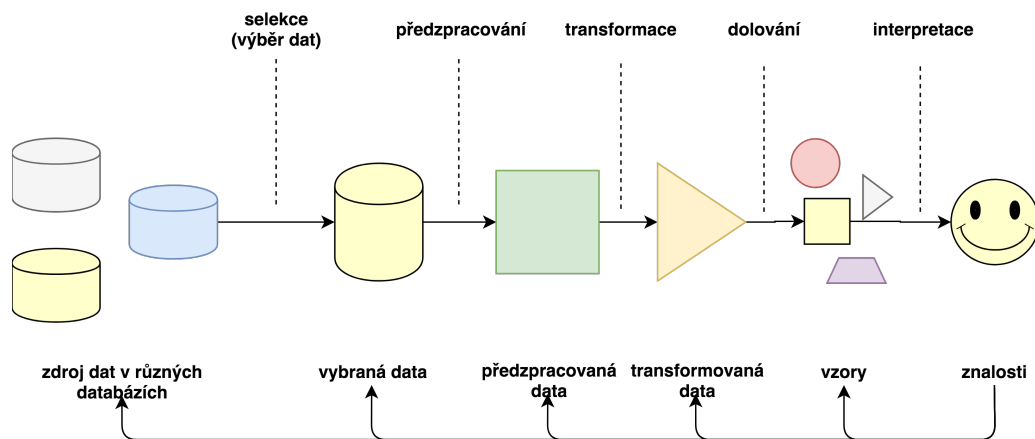
R zapojení do výuky

Dostupnost toolkitů 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. Následující kapitoly se pak budou zaměřovat na jednotlivé metody, 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.



Obrázek 6.1: Technický pohled na postup data miningu (adaptováno z [19])

Data použitelná v data miningu se obvykle nacházejí v různých informačních systémech, které organizace používá pro vedení svých agend. Data mohou být také získána z měřicí techniky, ale také externích zdrojů. Před jejich použitím pro účely data miningu je ale potřeba rozhodnout, která data jsou pro řešení zvoleného problému vůbec relevantní.

Vybíráme tak relevantní data (*selektce*) pro další zpracování. V tomto okamžiku 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 selektce 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 vazeb na obr. je možné se v rámci data miningu vrátit zpět a opakovat jeden nebo více kroků např. se změnou vstupních parametrů apod.

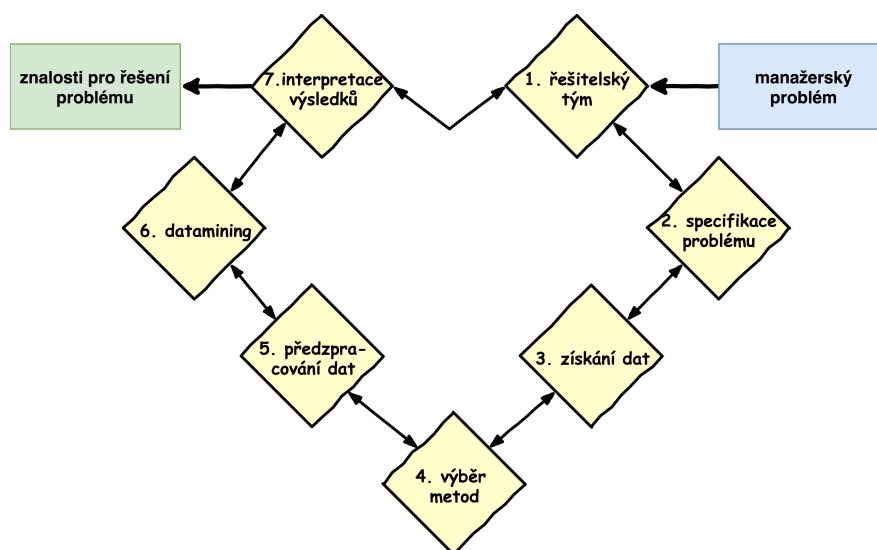
V kroku *transformace* připravujeme data na jejich zpracování pomocí zvolených data miningových metod. Tento krok je nutné provést, jelikož různé metody očekávají odlišné vstupy. V rámci transformace řešíme často následující problémy, ale nejen je:

- 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.

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



Obrázek 6.2: Manažerský pohled na postup data miningu (adaptováno z [19])

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 očekávány 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 záze-
mím.

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 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 [19]:

- klasifikace a predikce
- deskripce
- hledání nugetů

Klasifikace je pravděpodobně nejčastěji řešenou úlohou data miningu. Protřednictvím klasifikace se snažíme zatřídit hodnocený případ do některé z existujících kategorií. Např. vyhodnocením symptomů nemocného můžeme určit diagnózu (zaklasifikovat nemocného) a navrhnout vhodnou léčbu. Klasifikací je ale také například identifikace dopravních značek kamerovým systémem projíždějícího samostatného auta.

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.:

- statistika (např. regresní modely)
- pravidlově orientované metody jako rozhodovací stromy nebo pravidla
- metody strojového učení jako např. neuronové sítě
- apod.

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.

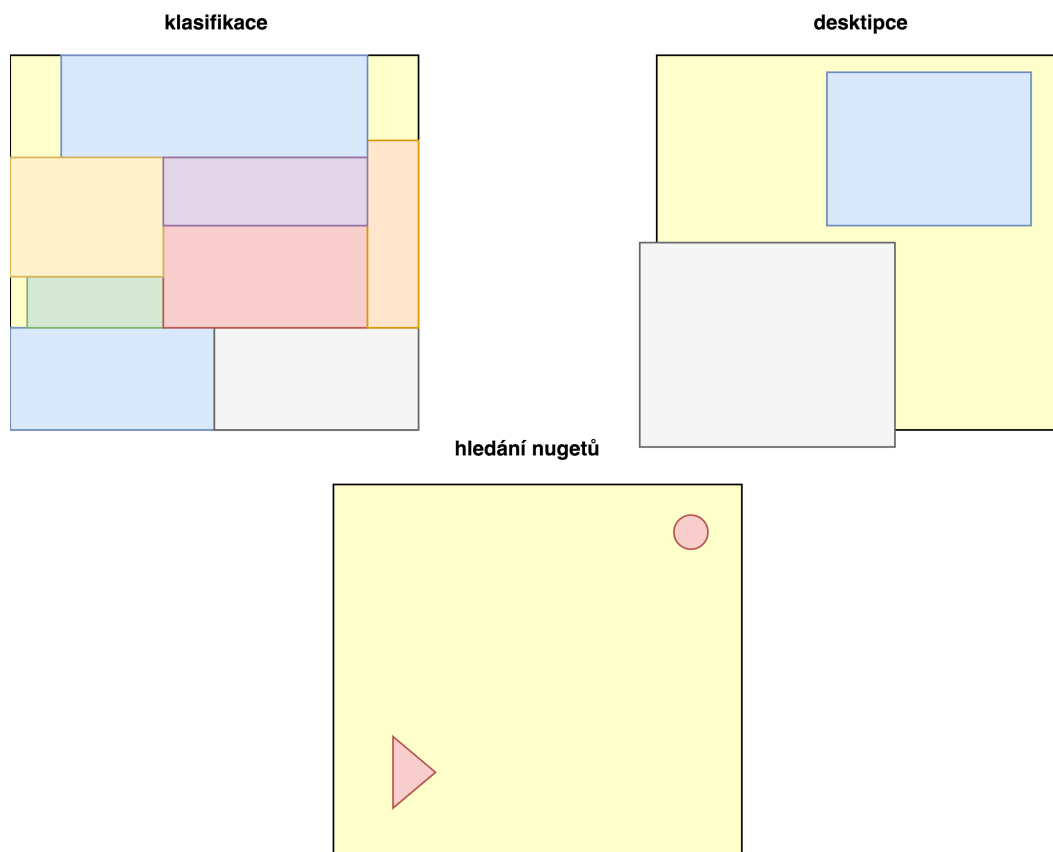
Konečně *dolování nugetů* se zaměřuje na identifikaci nových zajímavých, možná překvapivých informací v datech. Tam kde se tedy deskripce zaměřovala na získávání informací popisujících používaný datový soubor jako celek, dolování nugetů se zaměřuje spíše na minoritní část dat, která z nějakého důvodu 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.

Problematikou hledání nugetů se v tomto textu nebudeme příliš zabývat. Z metod, které by k tomuto účelu bylo možné použít lze zmínit např. asociativní pravidla.

Graficky si lze představit tyto tři typy úloh jako na obr. 6.3. Uvažujme na chvíli o těchto problémech jako o stěně, kterou máme postavit. Prostor, který má stěna pokrýt označíme jako koncept. Jednotlivé stavební prvky stěny nám budou sloužit pro pokrytí tohoto konceptu. Koncept je reprezentován na obr. žlutou barvou.



Obrázek 6.3: Úlohy data miningu - pokrytí konceptu (adaptováno z [19])

V případě klasifikace je požadováno, aby koncept byl pokrytý co možná nejlépe. Jednotlivé „stavební bloky“ by se neměly pokud možno překrývat. Nevyplněná místa, popř. překryvy totiž vedou ke snížení 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 také triviálním problémem.

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.

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 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 za 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).

Při posuzování nových žádostí o úvěr pak bankovní úřednice pouze vyplní shromážděné informace o klientovi do systému a kliknutím na tlačítko spustí proces vyhodnocování bonity.

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, Explore, Modify, Model a Assess.

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.

Modify (úprava) - připravuje data pro provedení analýz. Formálně odpovídá kroku transformace dat z předchozí podkapitoly. Výstupem jsou data připravená pro aplikaci zvolených analytických metod.

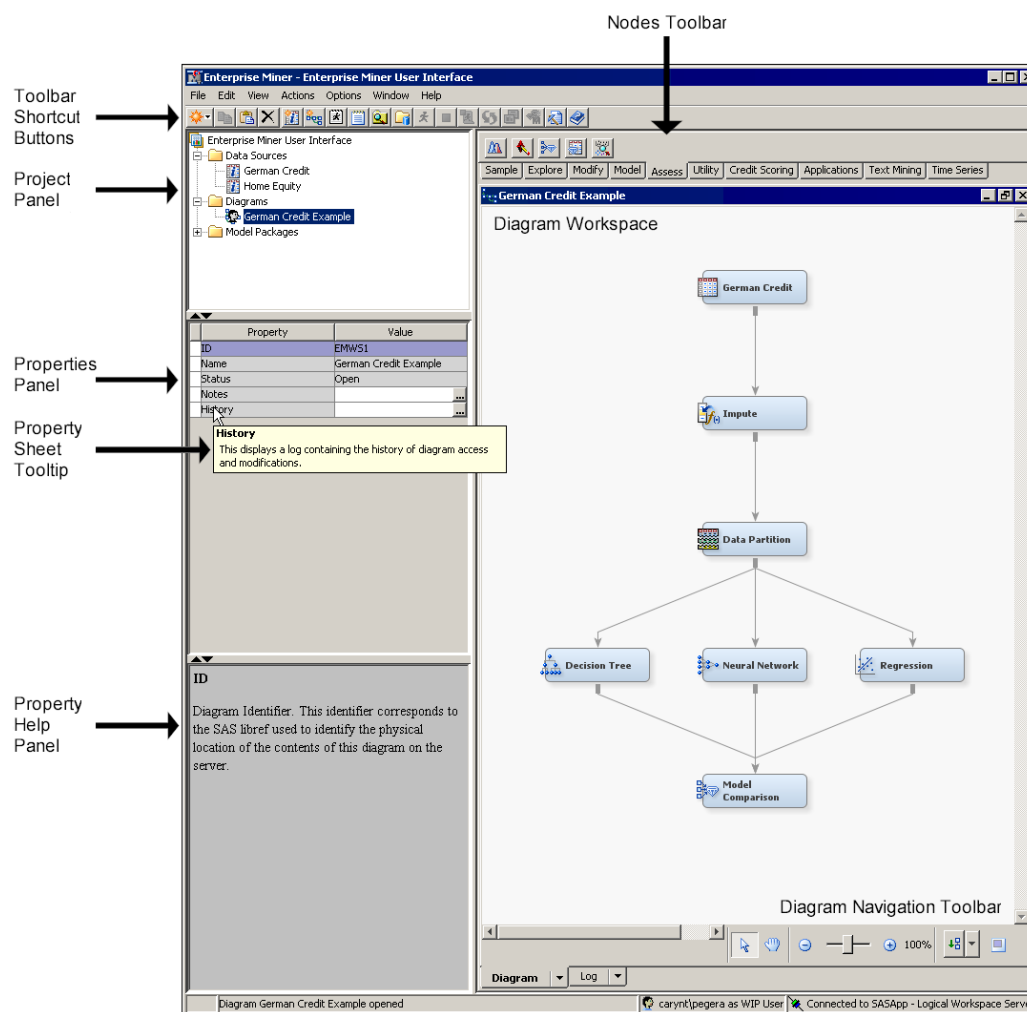
Zvolené metody jsou aplikovány v kroku *Model (modeluj)*. Tady je použití metodiky SEMMA poměrně specifické, úzce svázané s nástrojem SAS Enterprise Miner, pro který byla metodika vyvinuta. Tento nástroj má některé specifické vlastnosti v oblasti modelování a také hodnocení kvality modelů (*Assess*).

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. 6.4.

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.



Obrázek 6.4: Proces data miningu v SAS Enterprise Miner (převzato z [61])

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 ja 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. 6.5.

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 řešení izolované od fungování zbytku organizace.

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.

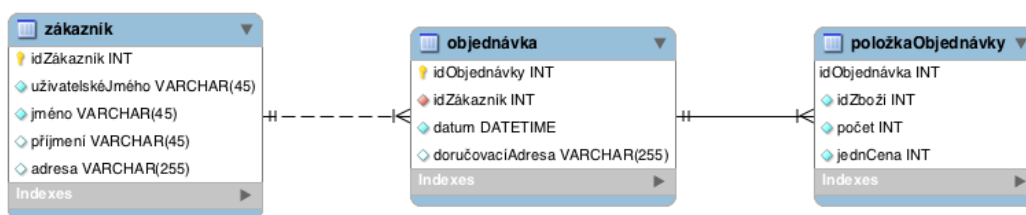
Byl splněn účel data miningu? Pokud ano, získali jsme dostatek informací k tomu, abychom problém vyřešili. Automatizované *nasazení* je pak logickým zakončením celého procesu.



Metodika vs metoda

Rozlišujte prosím mezi pojmy *metodika* a *metoda*. V této podkapitole (a v obecné rovině i v této 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!**

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. 6.6.

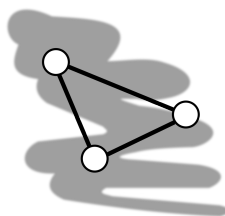


Obrázek 6.6: Příklad struktury zjednodušené objednávky

Příklad na obr. 6.6 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 podobnýchází dat (systémů) je potřeba jako zdroj použít často více. V takovém případě, ale vazba mezi daty z různých systémů nemusí být úplně jasná.



Databázová teorie

Základní informace o relačních databázích, způsobu návrhu báze dat je možno nalézt třeba ve skriptech předmětu *Bezpečnostní informatika* [75].

Ne všechny údaje jsou ale organizovány tímto způsobem. Na obr. 6.7 je znázorněna celková (zjednodušená) struktura informačních systémů středních/velkých společností.

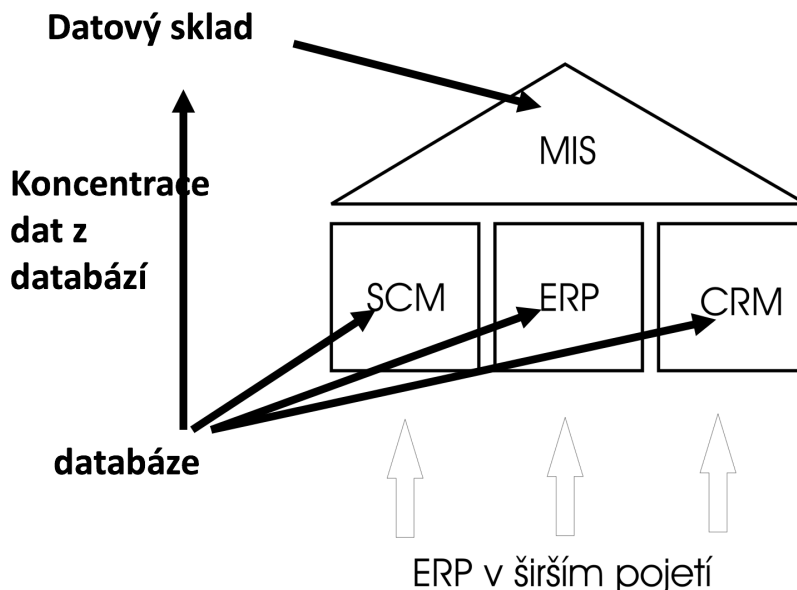
Jako základ fungování podniku jsou na obr. 6.7 ERP systémy sloužící po pokrytí běžných agend, které mají všechny organizace bez ohledu na předmět jejich činnosti. jedná se o:

- **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. 6.8.



Obrázek 6.7: Struktura informačních systémů podniku

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.

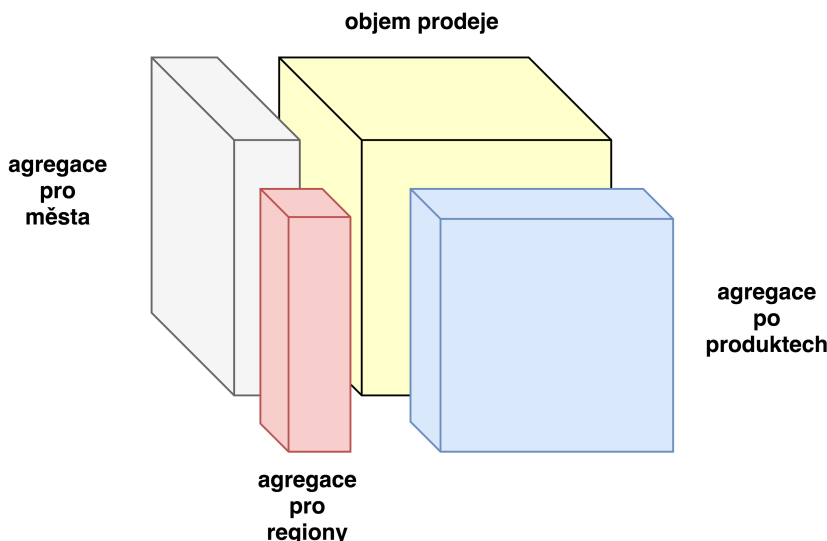
Řešení různých výrobců pak mohou přicházet s dalšími variantami práce s daty. Microsoft např. pracuje ještě s tzv. **Hybrid Online Analytical Processing (HOLAP)** [47]. Většinou se však jedná o speciální případy MOLAP nebo ROLAP řešení.

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.

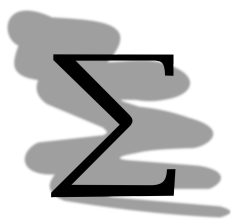
My v předmětu Bezpečnostní informatika 3 takové objemy dat zpracovávat nebudeme, proto berte výše uvedenou informaci jako doplňkovou.



Obrázek 6.8: Datová krychle v OLAP (adaptováno z [19])

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. Odvykle 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 jejích „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 strukturuující 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. Předtím, než vůbec začnete studovat R jako nástroj - nainstalujte jej 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 mohou obsahovat vždy pouze základy. Ilustrační příklady tak 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 stáh-

nout a spustit instalátor zvolené verze. Pro instalaci doporučujeme vždy poslední stabilní dostupnou verzi (v době psaní těchto skript to byla verze 3.5.1).

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 nainstalovanou 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://www.rstudio.com/> 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.

```
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 [55] 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.

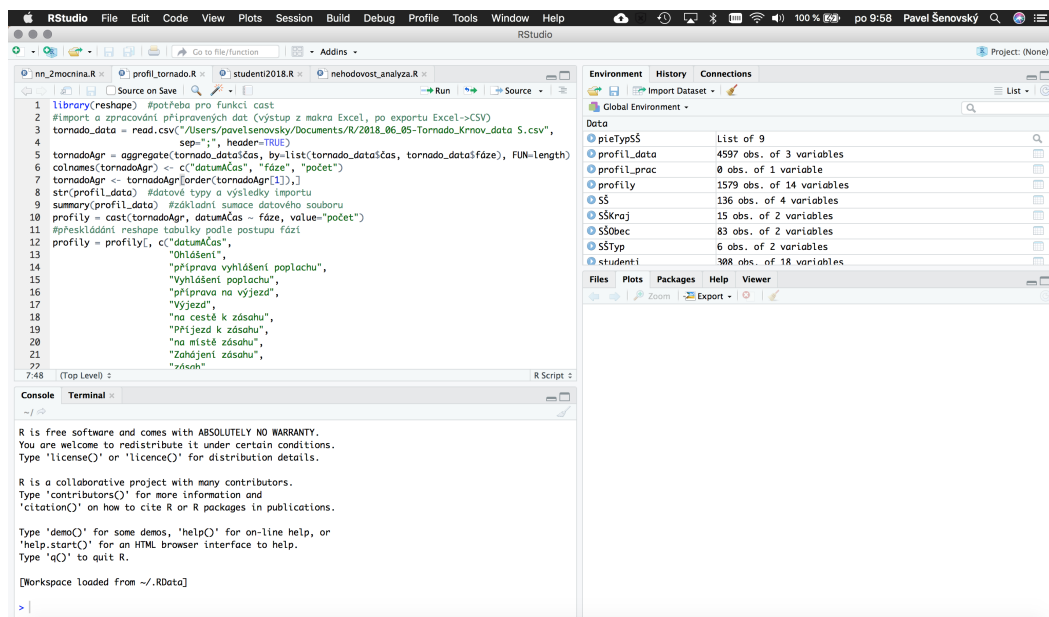
Oficiálně podporované jsou pak i distribuce RedHat (Fedora) a SuSe. Pokud používáte jinou distribuci Linux, bude nutné provést nejspíše kompilaci ze zdrojových kódů.

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.

Rozhraní RStudia má čtyři základní části:

- zdrojové kódy našich analytických projektů, popř. prohlížeč dat (vlevo nahoře)



Obrázek 7.1: Rozhraní RStudio v OS X

- konzole R (vlevo dole)
- prostředí (s jednotlivými načtenými datasy) - 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.

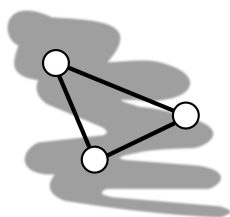
Např. na obr. 7.1 je znázorněn soubor řešící vytvoření časového profilu výjezdů jednotek HZS k zásahům vyvolaným tornádem v Krnově (2018). Editor podporuje zvýrazňování syntaxe, poskytuje bublinkovou nápovědu k jednotlivým parametrům používaných funkcí a také nápovědu podobnou intelisence - napovídající 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.



Proč R?

... a ne třeba Excel. MS Excel je skvělý produkt, ale v případě zpracování velkých objemů dat není právě nejpráhovější. Excel také neobsahuje plnohodnotnou podporu statistických nástrojů, nebo nástrojů data miningu. K těmto účelům jej tak lze použít, ale uživatelé 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“).

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 nového toolkitu, lze příkaz zadat do konzole přímo.

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 tabulka 4597 řádků a 3 sloupce.

Dvojitým kliknutím na název proměnné ji lze spustit v prohlížeči. Např. proměnná *tornado_data* vypadá v prohlížeči jako na obr. yrefObrTornadoData.

	Název.JPO	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 7.2: 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 porožena 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.

Konečně vpravo dole se zobrazuje případná grafika, jako jsou grafy, apod. Lze zde také prohlížet vestavěnou nápovědu apod.

7.3 Základy jazyka R

Od této sekce budou přikládány 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: +, -, *, /, ^ (umocnění), %% (zbytek po dělení). Pro přiřazení lze použít buďto = nebo <- (lomená závorka následovaná pomlčkou). My v zápisech budeme používat rovnítko, v literatuře se ale <- používá velmi často.

Experimentovat můžeme v tomto případě přímo v konzoli.

```

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.

```

sqrt(4) # $\sqrt{4} = 2$ 
abs(-4) # $|-4| = 4$ 
round(x, 2) #zaokrouhlí x na 2 desetinná místa
log(x) #přirozený logaritmus
log10(x) #logaritmus x o základu 10
log2(x) #logaritmus x o základu 2
trunc(x) #odebrání desetinných míst (zůstane celá část x)
ceiling(x) #zaokrouhlení x nahoru
floor(x) #zaokrouhlení x dolů
exp(x) # $e^x$ 
sin(x), cosin(x), tan(x) #trigonometrické funkce
asin(x), acos(x), atan(x) #nverzní trigonometrické funkce

```

Použití základních funkcí je relativně jednoduché, podívejme se jak na vektory.

Vektory

```

a = c(1, 2, 3, 4) #vytvoří vektor 1, 2, 3, 4
b = c(TRUE, FALSE, TRUE, FALSE) # vytvoří vektor booleanovských hodnot
c = c("one", "two", "three") # vytvoří vektor textových řetězců
class(a) #slouží pro zjištění datového typu ( výsledek je numeric, character, integer, complex,
logical)
d = as.integer(3) #vytvoří proměnnou typu integer o hodnotě 3
e = rep(v, times=3) #vytvoří vektor tak, že zopakuje první parameter (v) times krát
f = 1:5 #vytvoří vektor na základě sekvence 1, 2, 3, 4, 5, tedy od 1 do 5
g = seq(100, 2000, by=300) #vytvoří vektor na základě sekvence od 100 do 2000 s krokem 300

```

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.

```

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:

```
> ID = c(1,2,3,4)
> mesic = c("Leden", "Únor", "Březen", "Duben")
> prodeje = c(15000, 20000, 125000, 40000)
> region = c("východ", "západ", "jih", "sever")
> prodejData = data.frame(ID, mesic, prodeje, region)
> prodejData
  ID mesic prodeje region
1  1 Leden   15000 východ
2  2  Únor   20000  západ
3  3 Březen 125000    jih
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.

```
prodejData[2] # vypíše sloupec mesic
prodejData[c("mesic", "region")] #vypíše sloupce mesic a region
prodejData$prodeje # vypíše sloupec prodeje
prodejData[1:2] #vypíše první 2 sloupce
prodejData[-c(2,3)] #odebere sloupce 2 a 3
prodejData[-3] #odebere sloupec 3
prodejData$ID = NULL #odebere sloupec ID
#přidá další řádek
prodejData[nrow(prodejData) + 1,] = c(5, "Únor", 25000, "východ")
prodejData = prodejData[-nrow(prodejData)] #smaže poslední řádek
```

Všimněte si trochu odlišné práce buďto s označováním sloupců přes \$ nebo přes []. Zatímco notace \$ umožňuje přistoupit pouze k jednomu (pojmenovanému sloupci), pak [] umožňuje filtrovat tabulku, např.

```
> podminka = prodejData$prodeje > 20000 & prodejData$mesic == "Leden"
> prodejData[podminka,] #vybere řádky splňující podmínku
[1] ID      mesic   prodeje region
<0 rows> (or 0-length row.names)
> prodejData[!podminka,] #vybere řádky, které nesplňují podmínku
  ID mesic prodeje region
1  1 Leden   15000 východ
2  2  Únor   20000  západ
3  3 Březen 125000    jih
4  4  Duben   40000  sever
5  5  Únor   25000 východ
```

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:

```
prodejData[order(prodejData$prodeje),] #seřadit řádky datasetu prodejData podle prodejů
#seřadí podle prodejů a tam kde jsou stejné podle měsíce
prodejData[order(prodejData$prodeje, prodejData$mesic)]
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:

```
znamky = c("výborně", "velmi dobře", "dobře", "nedostatečně") #vektor typu character
znamky = factor(znamky)
znamky
```

Funkce *factor* přiřadí převáděným položkám integer hodnotu, tato hodnota, ale není viditelná. Jak to funguje můžeme zjistit porovnáním dvou prvků vektoru *znamky* (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*:

```
> znamky = c("výborně", "velmi dobře", "dobře", "nedostatečně")
> znamky = factor(znamky,
                  levels=c("výborně", "velmi dobře", "dobře", "nedostatečně"),
                  ordered=TRUE)
> znamky[1] > znamky[2]
[1] FALSE
> min(znamky)
[1] výborně
Levels: výborně < velmi dobře < dobře < nedostatečně
> max(znamky)
[1] nedostatečně
Levels: výborně < velmi dobře < dobře < nedostatečně
```

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:

```
import_data = read.csv(cesta, sep=";", header=TRUE)
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, RODB. 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.

```
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.

Základní statistické funkce

Základní přehled funkcí:

- `mean(x)` - průměr z `x`
- `length(x)` - počet prvků v `x`
- `median(x)` - medián z `x`
- `unique(dataset)` - z předloženého datasetu vytvoří vektor obsahující pouze unikátní položky datasetu (tedy bez opakování)
- `range(x)` - vrací rozsah hodnot v předloženém datasetu (nejmenší a největší), vrací jej jako dvouprvkový vektor
- `quantile(x, probs=seq(0, 1, 0.25))` - R nezná quartily, v tomto případě se to obchází jednoduchou sekvencí od 0 do 1 po 0.25. Místo sekvence lze zadat i jednu položku
- `sd(x)` - standardní odchylka (standard deviation)
- `min(x)`, `max(x)` - minimum a maximum
- `var(x)` - rozptyl
- `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. My budeme v tomto textu používat pouze tu nejjednodušší používající funkce:

- `plot` - běžné grafy
- `hist` - histogram

- pie - koláčový graf
- boxplot - graf typu box and whiskers

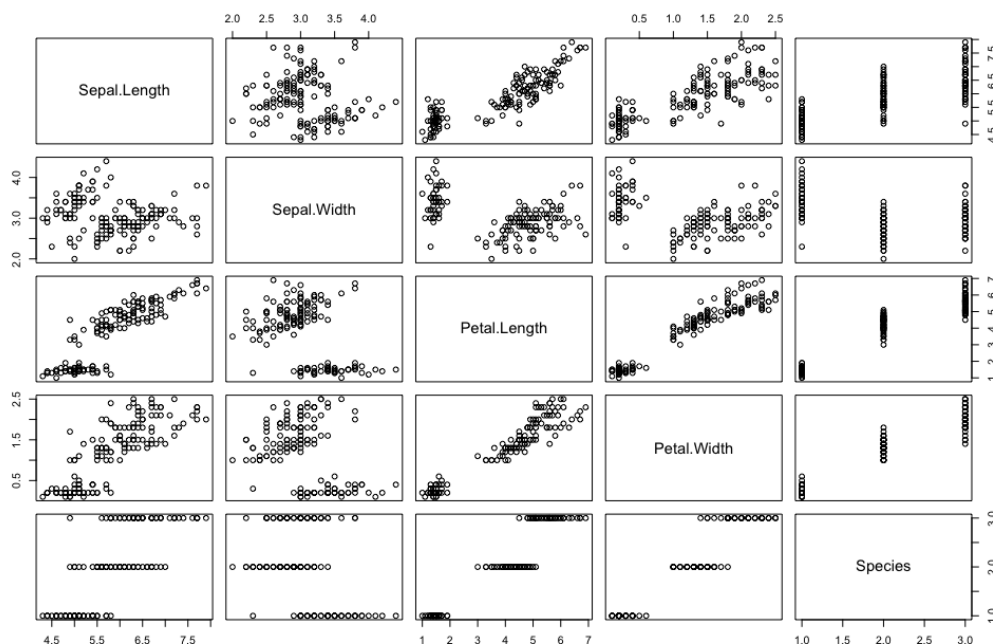
Bohužel tento typ grafů nevypadá vizuálně hezky. R poskytuje alternativu k tomuto prostředí pomocí knihovny GGPlot2, která umožňuje vytvářet graficky extrémně sofistikovanou grafiku. Cenou za větší možnosti knihovny je ale podstatně vyšší komplexita nastavení.

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 kosatců:

```
summary(iris)
plot(iris)
plot(iris$Sepal.Length~iris$Sepal.Width, col="red",
     main="Délka vs šířka kosatců", xlab="délka listu",
     ylab="šířka listu")
hist(iris$Sepal.Length, col="blue", main="Rozložení délky listu",
     xlab="délka listu", ylab="frekvence")
boxplot(iris, las=2,
        main="Boxplot graf naměřených vlastností kosatců", at=c(1,2,4,5,7),
        names=c("Sep.length", "Sep.width", "Pet.length", "Pet.wight", "Species"),
        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. 7.3.



Obrázek 7.3: Párové srovnání vlastností kosatců v datové sadě iris

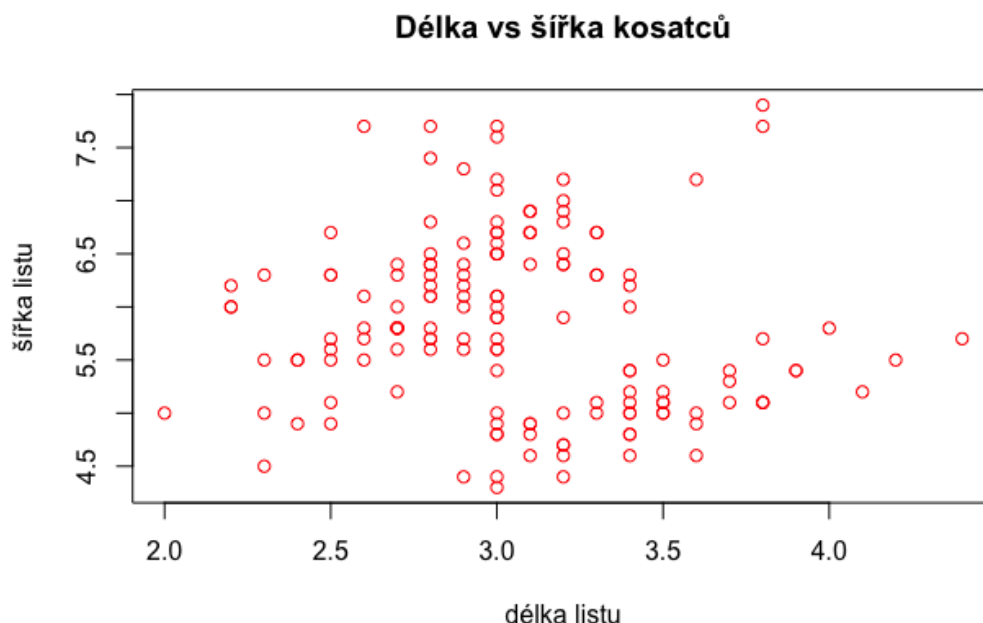
Názvy sloupců se vykreslují na diagonále grafu. Všimněte si, že grafy nad a pod diagonálou jsou podle diagonály zrcadlově obráceny.

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. 7.4.



Obrázek 7.4: Délka vs šířka listu kosatců v datasetu iris (bodový graf)

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. 7.5.

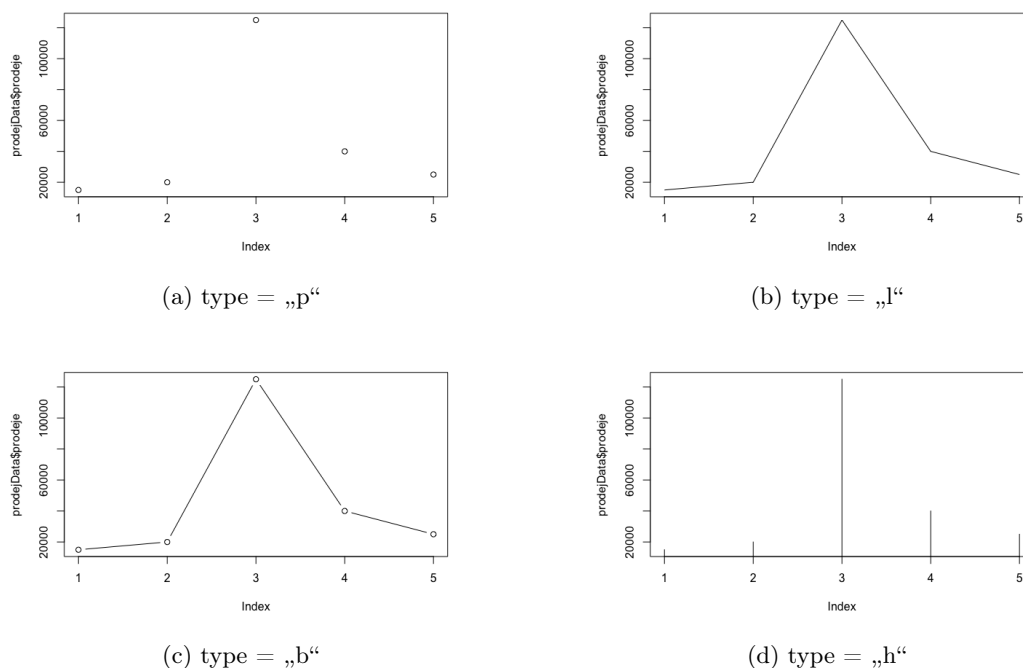
```
plot(prodejData$prodeje, type="p")
plot(prodejData$prodeje, type="l")
plot(prodejData$prodeje, type="b")
plot(prodejData$prodeje, type="h")
```

Vraťme se ale k datasetu iris - příklad histogramu je na obr. 7.6. I u histogramu lze použít řadu parametrů. Pro obr. 7.6 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. 7.7. 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.



Obrázek 7.5: Dostupné typy vykreslování ve funkci plot

Pomocí parametru `names` lze přepsat názvy polí. Pomocí parametru `col` jsou definovány barvy. V tomto případě jsou barvy zadávány vektorem.

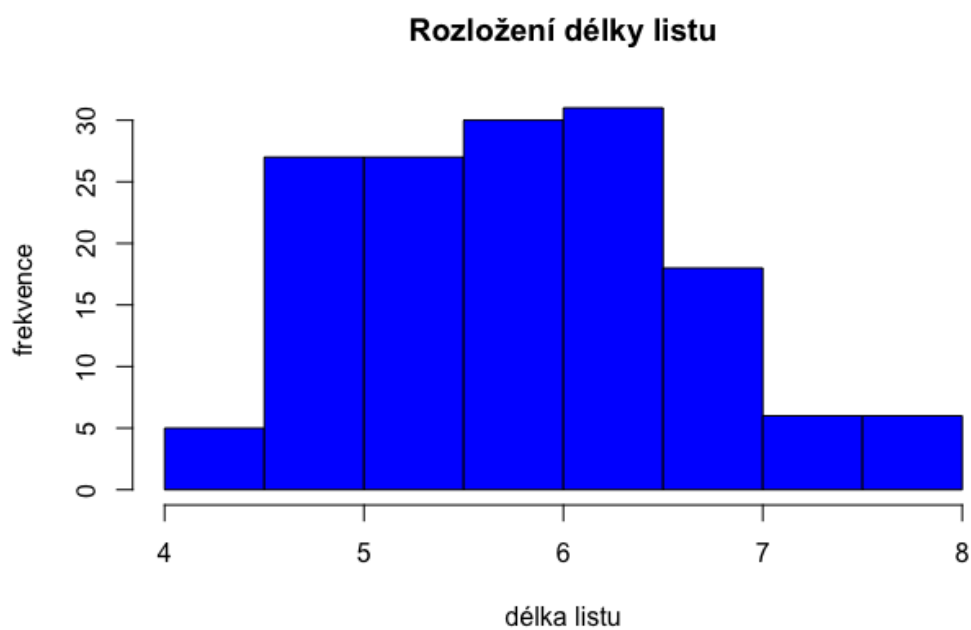
Poslední typ grafu, který jsem ještě neprobrali, je graf koláčový vytvářený funkcí `pie`. Tento graf je do určité míry specifický, protože vyžaduje určitou přípravu dat. Data, která jsou používána pro jeho vytvoření musí být agregovaná.

Funkce `pie` tak nemá agregaci zabudovanou přímo v sobě. Pro demonstraci použijeme náš upravený dataset prodejů. Jeho úpravu a vykreslení koláčového grafu budeme realizovat následujícím skriptem:

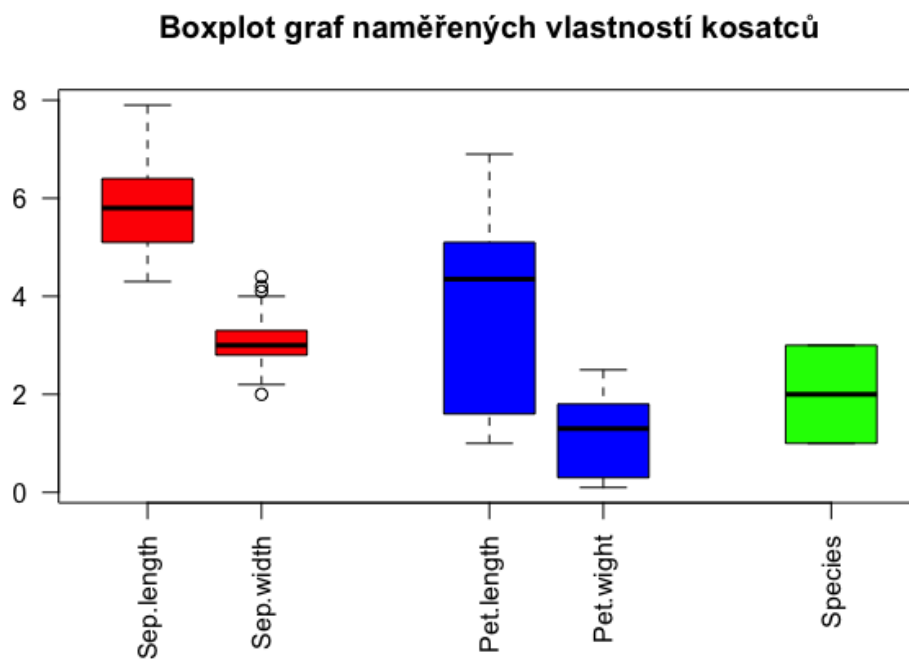
```
vykon <- c("mizerný", "mizerný", "skvělý", "průměrný", "skvělý")
vykon <- factor(vykon, levels=c("mizerný", "průměrný", "skvělý"))
prodejData$vykon <- vykon
prodeje = table(prodejData$vykon)
prodeje
pie(prodeje, main="Rozložení prodejů")
```

Všimněte si způsobu, jakým došlo ke zpracování údajů v proměnné `prodeje`. Takové údaje je již možno zobrazit v koláčovém grafu, viz obr. 7.8.

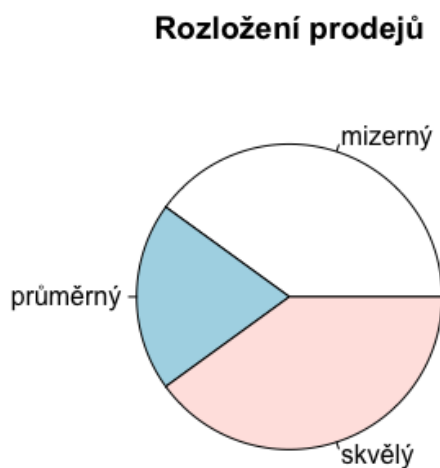
```
mizerný průměrný    skvělý
      2         1         2
```

Obrázek 7.6: Rozložení délky listu dataset iris (histogram)



Obrázek 7.7: Délka listu dataset iris (boxplot)



Obrázek 7.8: Rozložení počtu prodejů podle jejich úspěšnosti (pie)



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
- jaké jsou jejich výhody ale také omezení

umět

- prakticky použít metody *rozhodovacích stromů* a asociativních pravidel v R



Čas pro studium

Pro prostudování budete potřebovat 2 - 4 hodiny.

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 *trénovací množina* a *validační množina*. Trénovací množina se používá pro „natrénování“ pravidel, pomocí validační množiny se provádí ověření kvality natrénovaného modelu.

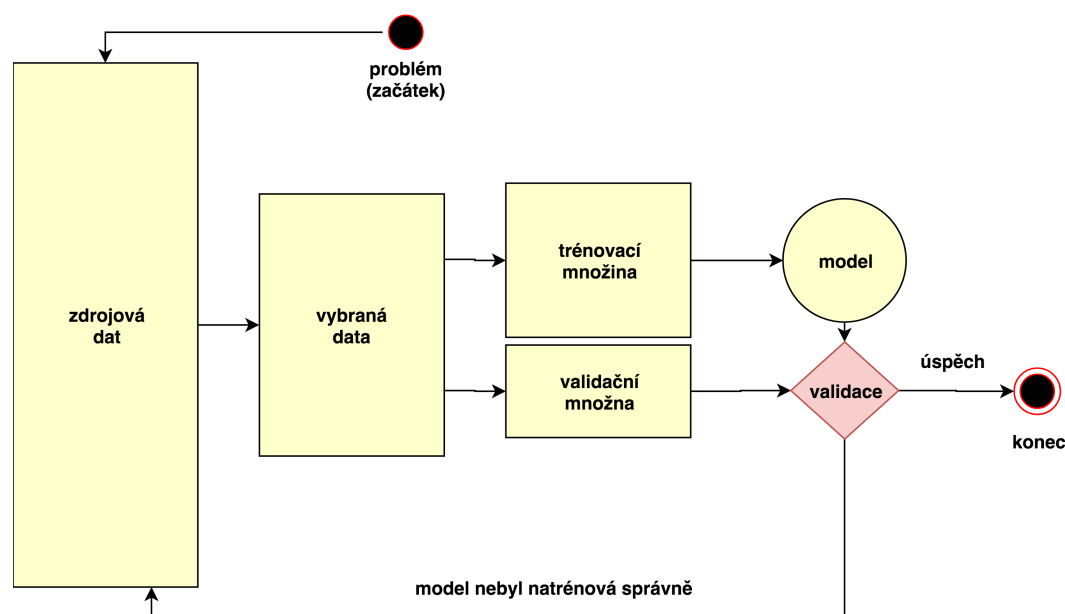
V žádném případě není možné kvalitu modelu hodnotit na základě trénovací množiny. Platí totiž, že jelikož byl model na základě trénovací množiny vytvořen, dobře reprezentuje data v ní obsažená. Otázka ale je, zda také dobře reprezentuje problém (resp. jeho řešení).

Trénovací 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 trénovací 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.

Graficky je postup použití množin znázorněn na obr. 8.1.



Obrázek 8.1: Filozofie použití trénovací a validační množiny

Proces zobrazený na obr. 8.1 je univerzálně použitelný. Není tedy specifický pro metody rozhodovacích stromů, popř. rozhodovacích pravidel.

U jednotlivých sloupců datového souboru 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 vypovídací 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 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. [38] (kapitola 2.2).

Takže máme připravená data, můžeme se pustit do metod samotných.

8.2 Rozhodovací stromy

Segmentují znalosti do podoby stromu. Pravidla pak popisují průchod tímto stromem od kořene stromu směrem k listu. List pak představuje samotné zatřídění případu.

Z požadavků na data (viz výše) už víme, že každý sloupec dat obsahuje konečný počet hodnot. Různé hodnoty nám pak představují větvení. Vhodné sestavení stromu je pak z velké části o identifikaci těch sloupců v datech, které mají největší potenciál vysvětlit zatřídění.

Existuje řada algoritmů, které pro segmentaci lze použít. Pravděpodobně nejpoužívanějším 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* [74] 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.

Dosazením 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: *Náším 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 [19])

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 zbytek 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).

Výpočet entropie sloupce *příjem* pak proběhne následovně podle vzorců 8.2 a (8.3):

$$H(\text{prijem}) = \frac{5}{12} H(\text{prijem}(\text{vysoky})) + \frac{7}{12} H(\text{prijem}(\text{nizky}))$$

Tabulka 8.2: Příjem vs úvěr

	úvěr ano	úvěr ne
příjem vysoký	5	0
příjem nízký	3	4

$$H(prijem(vysoky)) = -\frac{5}{5} \log_2 \frac{5}{5} - \frac{0}{5} \log_2 \frac{0}{5} = 0 + 0 = 0$$

$$H(prijem(nizky)) = -\frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} = 0,9852$$

$$H(prijem) = \frac{5}{12} 0 + \frac{7}{12} 0,985 = 0,5747$$

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.)

$$H(konto) = \frac{4}{12} 0 + \frac{4}{12} 1 + \frac{4}{12} 1 = 0,6667$$

$$H(pohlavi) = \frac{6}{12} 0,9183 + \frac{6}{12} 0,9183 = 0,9183$$

$$H(neram) = \frac{6}{12} 0,65 + \frac{6}{12} 1 = 0,825$$

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 ještě 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 analýzu. 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:

$$H(konto) = \frac{2}{7} 0 + \frac{3}{7} 0,9183 + \frac{2}{7} 0 = 0,3936$$

$$H(pohlavi) = \frac{4}{7} 1 + \frac{3}{7} 0,9183 = 0,965$$

$$H(nezam) = \frac{2}{7} 0 + \frac{5}{7} 0,971 = 0,9793$$

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.

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

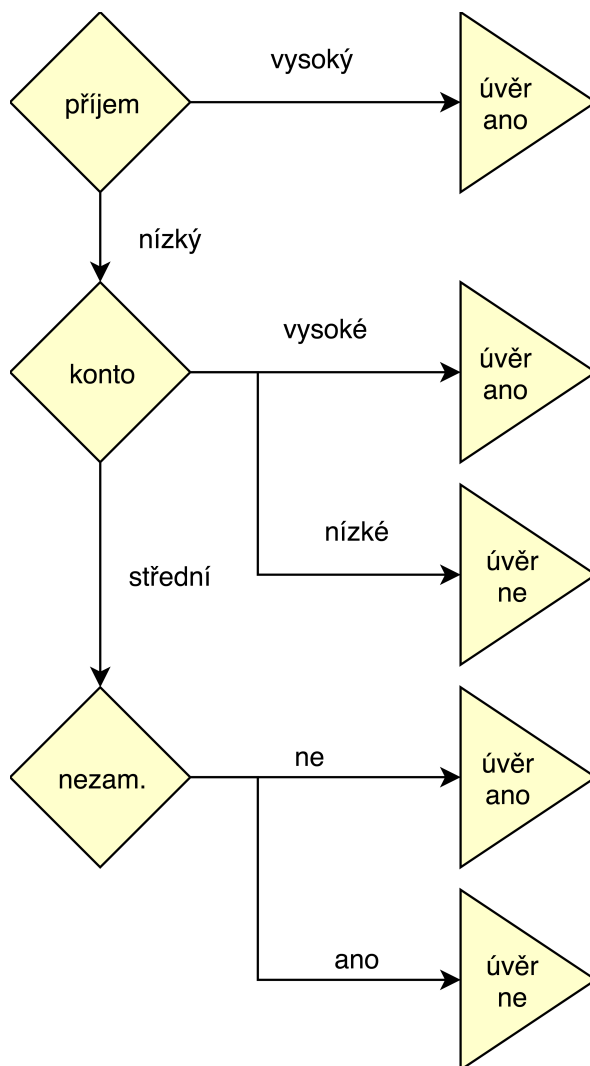
V těchto případech se tedy bude jednat o listové uzly. Konto střední ve dvou případech vede na přidělení a v jednom případě na zamítnutí úvěru.

Proto opět vyřadíme už zatříděné řádky (viz tabulka 8.4) a pokračujeme dále ve výpočtu.

$$H(\text{pohlaví}) = \frac{2}{3}1 + \frac{1}{3}0 = 0,6667$$

$$H(\text{nezamestnaný}) = \frac{2}{3}0 + \frac{1}{3}1 = 0,3333$$

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:

```

IF příjem(vysoký) THEN úvěr(ano)
IF příjem(nízký) && konto(vysoké) THEN úvěr(ano)
IF příjem(nízký) && konto(nízké) THEN úvěr(ne)
IF příjem(nízký) && konto(střední) && nezam(ne) THEN úvěr(ano)
IF příjem(nízký) && konto(střední) && nezam(ano) THEN úvěr(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:

```

IF příjem(vysoký) THEN
    úvěr(ano)
ELSE IF konto(vysoké) THEN
    úvěr(ano)
ELSE IF konto(nízké) THEN
    úvěr(ne)
ELSE IF nezam(ne) THEN
    úvěr(ano)
ELSE úvěr(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 12 vyhodnocení vždy.

Při vyhodnocování velkého množství případů tak může být dosaženo poměrně značné časové úspory.

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.

$$Ginni = 1 - \sum_{t=1}^T p_t^2 \quad (8.4)$$

Pravděpodobnosti opět převedeme do podoby relativních četností (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. 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“ [57]. Před prvním použitím je tento toolkit potřeba instalovat z CRAN. To provedeme zadáním příkazu:

```
> install.packages("tree")
```

do konzole R. R by se samo mělo připojit k vybranému repozitáři CRAN, stáhnout z něj toolkit a doinstalovat jej do prostředí R.

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.

```
library("tree")
klienti = read.csv("klienti_banky.csv", sep = ";", header=TRUE)
#následující nastavení je nutné pouze z důvodu velmi malého počtu měření
#v trénovací množině
tree_nastaveni = tree.control(12, mincut = 1, minsize = 2, mindev = 0.00000001)
model_tree = tree(
  klienti$úvěr ~ klienti$príjem + klienti$konto + klienti$pohlaví + klienti$nezam,
  data = klienti, control = tree_nastaveni)
summary(model_tree)
plot(model_tree) #vykreslí strom pomocí grafu
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 tom, 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 přiřazením do určité části modelu.

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:

```
Classification tree:
tree(formula = klienti$úvěr ~ klienti$príjem + klienti$konto +
      klienti$pohlaví + klienti$nezam, data = klienti, control = tree_nastaveni)
Variables actually used in tree construction:
```

```
[1] "klienti$príjem" "klienti$konto" "klienti$nezam"
```

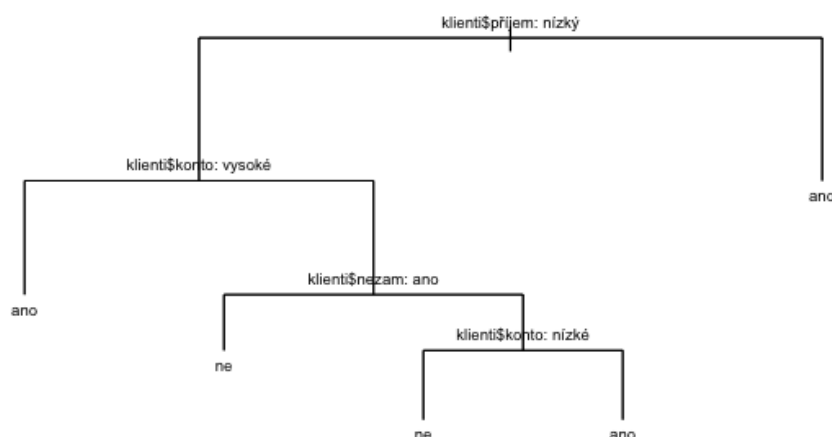
Number of terminal nodes: 5

Residual mean deviance: 0 = 0 / 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.

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.

```
scoreMnozina = klienti
predpoved = predict(model_tree, scoreMnozina, type="class")
predpoved #zobrazí, jak byly jednotlivé případy zaklasifikovány
table(pred=predpoved,true=scoreMnozina$úvěr) #konstrukce confusion matrix
mean(predpoved != scoreMnozina$úvěr) #výpočet průměrné chyby v klasifikaci
```

Předpověď výsledku je realizována funkcí *predict*. Všimněte si tzv. *confusion matice* (viz níže). Ta umožňuje porovnat jak byly klasifikovány případy a jak měly být klasifikovány případy.

	true	
pred	ano	ne
ano	8	0
ne	0	4

Podle tabulky výše jsou předpovědi v řádcích (pred) a skutečné hodnoty pak ve sloupcích (true).

8.3 Rozhodovací pravidla

K rozhodovacím pravidlům jsme dospěli již v předchozí kapitole, jako jedné z možností jak reprezentovat výsledek modelování. Co ale dělat v případě, kdy chceme odvodit pravidla aniž bychom odvozovali nejprve strom?

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 posledním srovnáním s existujícími pravidly můžeme toto pravidlo zjednodušit do podoby: IF konto(střední) && nezaměstnaný(ne) THEN úvěr(ano).

Vzniká nám sada pravidel

```
IF konto(vysoké) THEN úvěr(ano)
IF příjem(vysoký) THEN úvěr(ano)
IF konto(střední) && nezaměstnaný(ne) THEN úvěr(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 reálné dosažitelné řešení, které stále ještě splňuje podmínky zadání).

Z praktických důvodů ale doporučujeme používat spíše rozhodovací stromy.

8.4 Asociační pravidla

Oproti rozhodovacím stromům a pravidlům nesloučí pravidla asociační ke klasifikaci případů. Umožňují ná 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. [18] 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

Dobrym 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 vezte, 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 8.5. 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 8.5: Párek vs hořčice

	hořčice suc	!hořčice !suc	Σ
párek (ant)	a	b	r
!párek (!ant)	c	d	s
Σ	k	l	n

V tabulce 8.5 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).

$$\text{lift}(\text{ant} \Rightarrow \text{suc}) = \frac{P(\text{suc}|\text{ant})}{P(\text{suc})} = \frac{P(\text{ant} \wedge \text{suc})}{P(\text{ant})P(\text{suc})} \quad (8.11)$$

Lift je tedy definován jako poměr případů podporující pravidlo k součinu apriori pravděpodobností, že jednotlivé části pravidla nastanou bez ohledu na cokoliv jiného.

Hodnota lift může vyjít větší, menší nebo rovna 1. Hodnoty menší než 1 ukazují na negativní závislost popř. substituční efekt podle charakteru dat. Hodnoty větší než 1 ukazují na pozitivní závislost, popř. komplementární efekt.

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 Internetu (jedná se o open source dataset) nebo třeba z LMS, kde je k dispozici lokální kopie datasetu. Nejprve provedeme načtení a průzkum datasetu.

```
load("cesta/titanic.raw.rdata")
str(titanic.raw)
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. Vyhněte 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.

```
save(klienti, file = "cesta/klienti.rdata")
#uložení více datasetů do souboru
save(klienti, prodejData, file = "cesta/cviceni.rdata")
load("cesta/cviceni.rdata") #opětovné načtení souboru
```

Funkce *str* vypíše vlastnosti datového souboru. Dataset Titanic obsahuje celkově 2201 pozorování (pasažérů + posádky lody) 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 8.6.

Tabulka 8.6: 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:

```
library(arules)
pravidla = apriori(titanic.raw) #natrénování asociačních pravidel
pravidla #vypíše kolik pravidel bylo natrénováno
inspect(pravidla) #vypíše pravidla
```

Pro práci s asociačními pravidly použijeme toolkit *arules* [33]. 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.packages("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.

```
pravidla = apriori(titanic.raw, parameter = list(minlen=3, supp=0.002, conf=0.2),
  appearance = list(
    rhs=c("Survived=Yes"),
    lhs=c("Class=1st", "Class=2nd", "Class=3rd", "Age=Child", "Age=Adult"),
    default="none"), control = list(verbose=F))
pravidla.sorted = sort(pravidla, by="confidence")
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

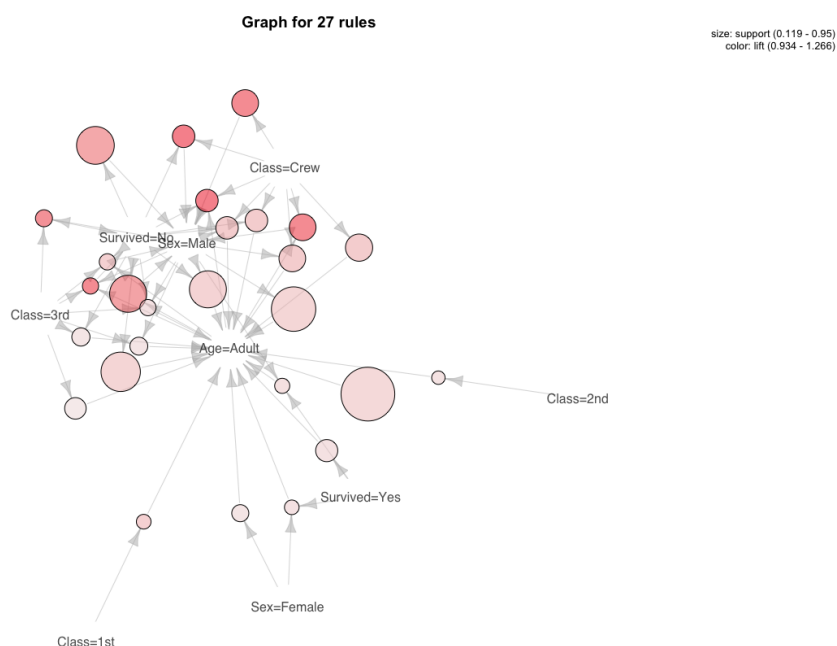
Tento pohled nám už umožňuje srovnat šanci na přežití podle věku - co z tohoto pohledu vyvozuje? Na pomoc si vezměte údaje o zachráněných - můžete použít buď přímo dataset Titanic nebo použijte pro srovnání předpřipravená data dostupná např. na Wikipedii (doporučuji anglickou verzi článku - má přehlednější 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. 8.4) tak více cílenou sadu pravidel (viz obr. 8.5).

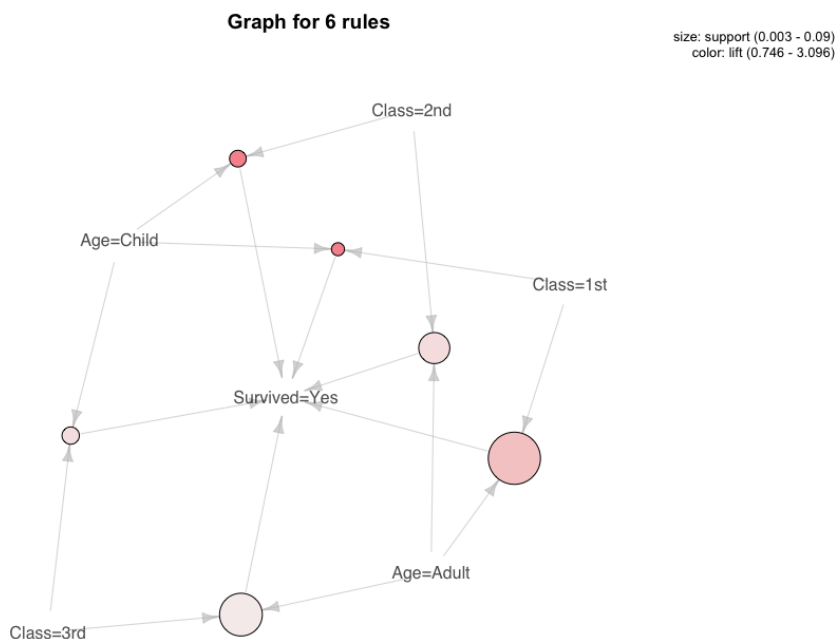
```
library(arulesViz)
plot(pravidla, method="graph")
plot(pravidla.sorted, method="graph")
```

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.

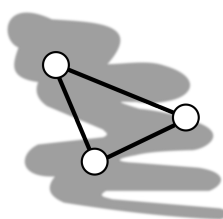
Jak je patrné z výše uvedeného - výpočet pravidel samotný je jednoduchý. To co je složité, je jejich interpretace. Pro úspěšnou analýzu proto musíme mít podrobné informace o zkoumaném problému. Odvozené znalosti bychom proto měli dále podrobit kritickému zkoumání. (Alespoň než na jejich základě začneme realizovat revoluci.)



Obrázek 8.4: Asociační pravidla - dataset Titanic (27 základních pravidel)



Obrázek 8.5: Asociační pravidla - dataset Titanic (zaměřeno na přeživší podle věku a třídy)



Titanic dataset a jeho analýza

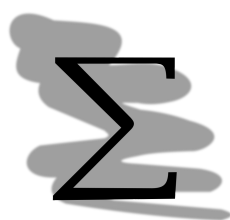
Tento dataset je jedním z nejpoužívanějších pro demonstraci tvorby asociačních pravidel. Na Internetu tak lze dohledat desítky příkladů, které dataset analyzují z různých pohledů. Tyto příklady můžete vyhledat např. pomocí vyhledávače Google pomocí řetězce: *association rules titanic example*.

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.

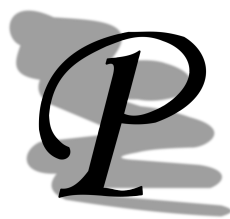
Vytvořený rozhodovací strom lze převést na pravidla.

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.

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):

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \quad (9.1)$$

Vw 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 mysteriózní 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 jediné, co je schopen rozeznat je, že nosí kalhoty. Jaká je pravděpodobnost toho, že se jedná o dívku?

Tento příklad je tak jednoduchý, že je možné jej spočítat v hlavě. My jej ale vypočteme formálně a následně problém převedeme do formy klasifikačního problému. a závěry zobecníme.

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 odpovědí 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 reformulovat. *Pokud vidím v dálce studujícího s kalhoty jedná se spíše o chlapce nebo o dívku?* 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 je pro všechny hypotézy stejný. Jeho velikost je tedy z hlediska pořadí hypotéz podle pravděpodobnosti irelevantní.

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).

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:

$P(\text{úvěr(ano)}) = 8/12 = 0,67$
 $P(\text{úvěr(ne)}) = 4/12 = 0,33$
 $P(\text{příjem(vysoký)}|\text{úvěr(ano)}) = 5/5 = 1$
 $P(\text{příjem(nízký)}|\text{úvěr(ano)}) = 4/7 = 0,43$
 $P(\text{příjem(vysoký)}|\text{úvěr(ne)}) = 0/5 = 0$
 $P(\text{příjem(nízký)}|\text{úvěr(ne)}) = 4/7 = 0,57$
 $P(\text{konto(vysoké)}|\text{úvěr(ano)}) = 4/4 = 1$
 $P(\text{konto(střední)}|\text{úvěr(ano)}) = 2/4 = 0,5$
 $P(\text{konto(nízké)}|\text{úvěr(ano)}) = 2/4 = 0,5$
 $P(\text{konto(vysoké)}|\text{úvěr(ne)}) = 0/4 = 0$
 $P(\text{konto(střední)}|\text{úvěr(ne)}) = 2/4 = 0,5$
 $P(\text{konto(nízké)}|\text{úvěr(ne)}) = 2/4 = 0,5$
 $P(\text{pohlaví(žena)}|\text{úvěr(ano)}) = 4/6 = 0,67$
 $P(\text{pohlaví(muž)}|\text{úvěr(ano)}) = 4/6 = 0,67$
 $P(\text{pohlaví(žena)}|\text{úvěr(ne)}) = 2/6 = 0,33$
 $P(\text{pohlaví(muž)}|\text{úvěr(ne)}) = 2/6 = 0,33$
 $P(\text{nezaměstnaný(ne)}|\text{úvěr(ano)}) = 5/6 = 0,83$
 $P(\text{nezaměstnaný(ano)}|\text{úvěr(ano)}) = 3/6 = 0,5$
 $P(\text{nezaměstnaný(ne)}|\text{úvěr(ne)}) = 1/6 = 0,17$
 $P(\text{nezaměstnaný(ano)}|\text{úvěr(ne)}) = 3/6 = 0,5$

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 k 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.

Pokud to srovnáme s konceptem „zapomínání“ používaným rozhodovacími pravidly nebo stromy (konzultujte prezentaci pokud si nevzpomínáte na přednášku) - jedná se o podstatně elegantnější řešení, které má koncept změny pevně zabudován přímo do jádra metody.

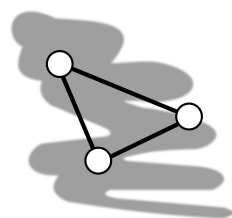
Byť výpočet není nikterak složitý, je očividné, že realizovat reálný data miningový projekt bez podpory výpočetní techniky by bylo minimálně ... nepohodlné, pokud ne extrémně nepříjemné. Jak nám tedy s naivním Bayesovým klasifikátorem může pomoci třeba R?

```
library("e1071")
klienti = read.csv("cesta/klienti_banky.csv", sep = ";", header = TRUE)
model = naiveBayes(klienti$úvěr ~ klienti$príjem + klienti$konto +
                    klienti$pohlaví + klienti$nezam, data = klienti)
model #vypíše natrénované podméněné pravděpodobnosti
scoreMnozina = read.csv("cesta/klienti_banky_score.csv", sep = ";", header = TRUE)
predpoved = predict(model, scoreMnozina)
predpoved #zobrazí, jak byly jednotlivé případy zaklasifikovány
table(pred=predpoved,true=scoreMnozina$úvěr) #konstrukce confusion matrix
```

Naivní Bayesův klasifikátor je implementován např. v ne úplně intuitivně pojmenovaném toolkitu e1071 [46] 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.



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?

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.



Příklady v R

Pokud jste ještě nevyzkoušeli realizovat příklad v R, teď je ta správná doba :-).



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 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 exportů 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, které jsou v současnosti využívány, jsou ale relativně malé - obsahují tisíce až milióny neuronů podle problému a architektury neuronové sítě zvolené pro jeho řešení. Pokud to srovnáme s počty neuronů v mozcích, pak průměrný hlodavec má v mozku 12 a člověk 86 miliard neuronů [34].

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 [34]). To, co považujeme za inteligenci je pak odvozováno také z toho, jak je mozek organizován, tedy k čemu jsou neurony používány.

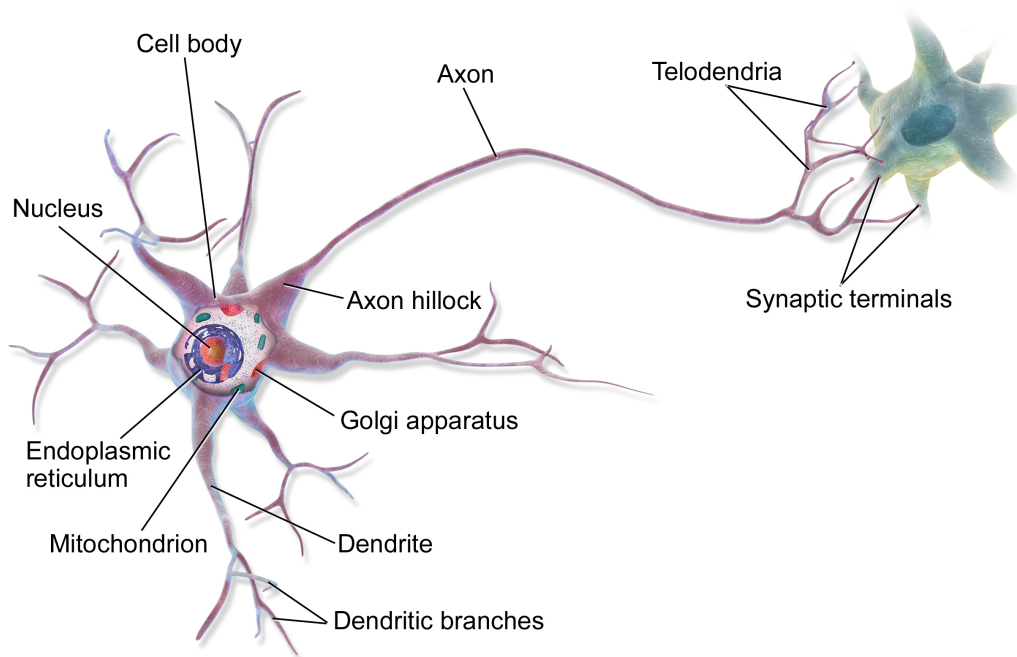
Umělé neuronové sítě, jak jsou používány dnes ke svému tréninku obvykle vyžadují nasazení poměrně značného množství strojového času superpočítače. Takové neuronové sítě jsou ale vždy specializované na řešení určitého konkrétního úkolu. Náš mozek oproti tomu je univerzální.

Lze také říci, že v současnosti výzkum ani nesměruje k vytvoření plnohodnotné umělé inteligence, byť už dnes jsou úvahy o otíckých otázkách spojených se spuštěním a provozováním takových systémů, viz např. iniciativa OpenAI [15].

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 [20])

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 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 je vzruch dojde k excitaci neuronu a vzruch se prostřednictvím jeho axonu šíří dále.

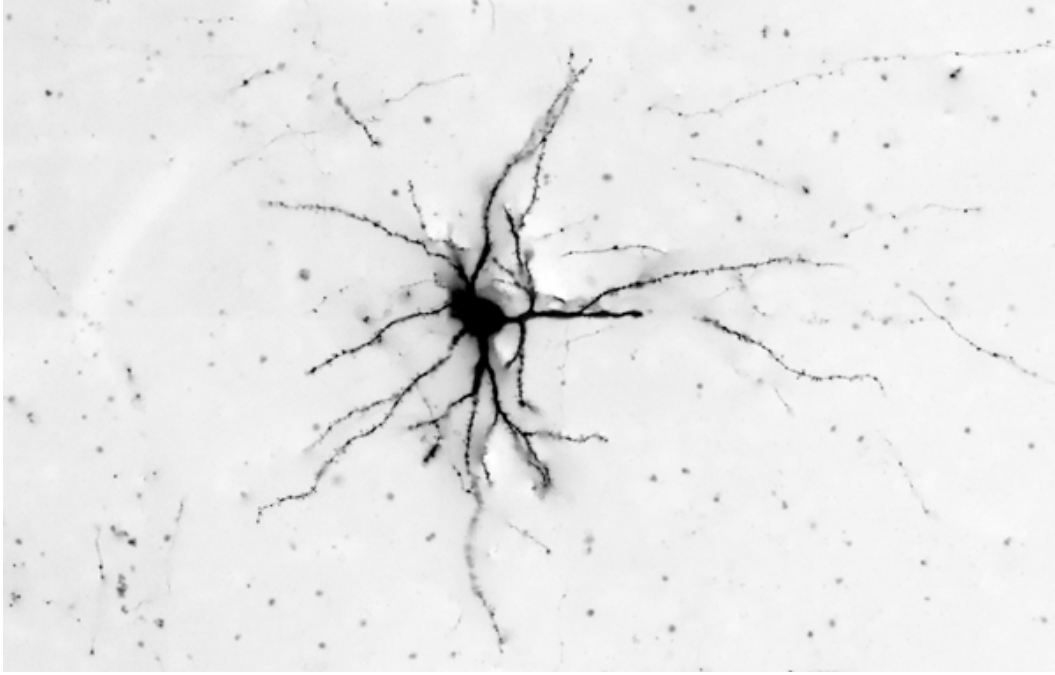
Naopak pokud intenzita nervového vzruchů nepřesáhne hodnotu prahu, dojde k utlumení vzruchu - neuron nepřejde do excitovaného stavu a nešíří vzruch dále.

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 [44] - 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.



Obrázek 10.2: Neuron vizuálního kortexu (převzato z [37])

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 $u(t)$ (??).

$$z(t) = x(t) \cdot w(t) \quad (10.1)$$

kde z je vážený vstupní signál, x normovaný vstupní signál (obvykle v intervalu 0-1), w jsou pak váhy.

$$u(t) = \sum_{i=1}^n x_i(t) \cdot w_i(t) \quad (10.2)$$

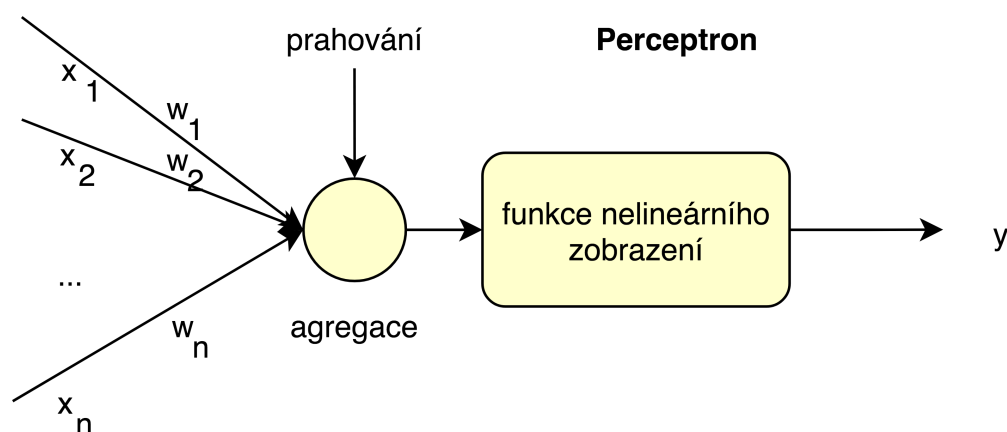
Prahování se provádí tak, že se od agregovaných vstupních signálů $u(t)$ odečte hodnota prahu w_0 , viz rovnice (10.3).

$$v(t) = u(t) - w_0 \quad (10.3)$$

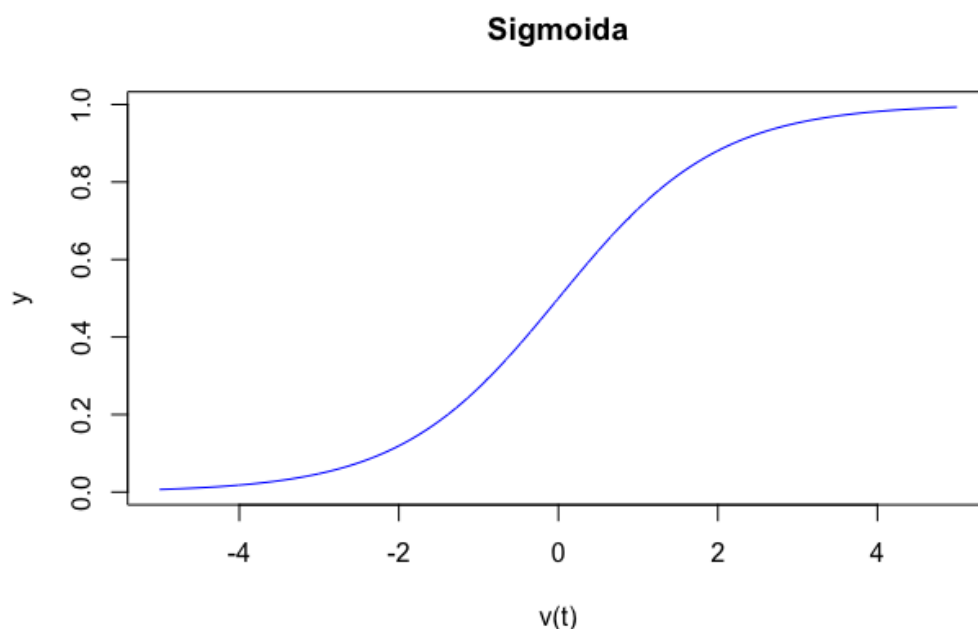
Funkci nelineárního zobrazení volíme podle požadavků, které klademe transformaci prahované hodnoty $v(t) \rightarrow y$ na hodnotu výstupní. Jednou z nejčastěji používaných funkcí tohoto typu je tzv. *sigmoida*, kterou je možno vypočítat dle vzorce (10.4). Graficky je pak sigmoida znázorněna na obr. 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.



Obrázek 10.3: Schématické znázornění modelu perceptronu



Obrázek 10.4: Graf sigmoidy

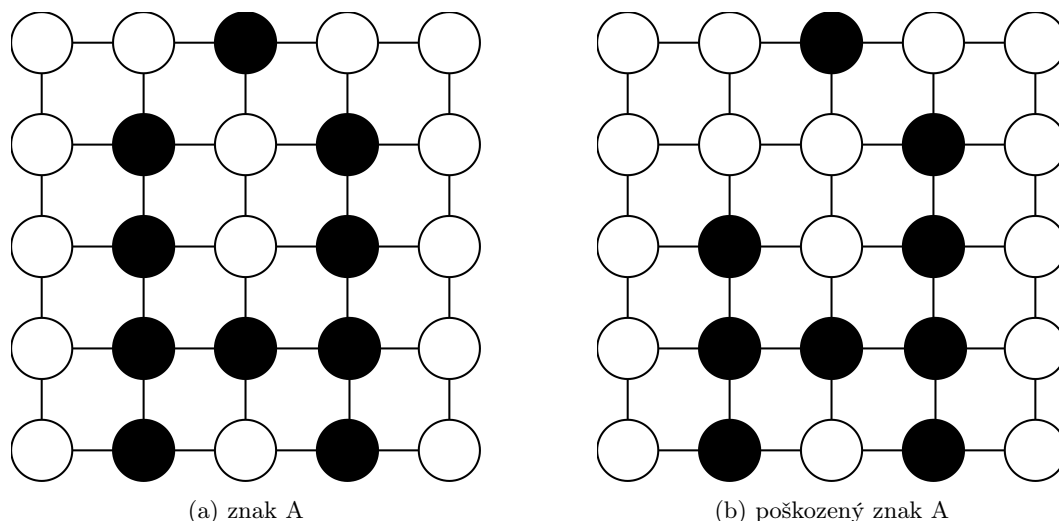
10.2 Neuronové sítě a jejich adaptace

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 většího počtu neuronů se skládá) tím dokáže být při řešení problémů přesnější.

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 reprezentované bílou a černou barvou znaku - černá znamená součást písmene, bílá znamená papír.*

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,

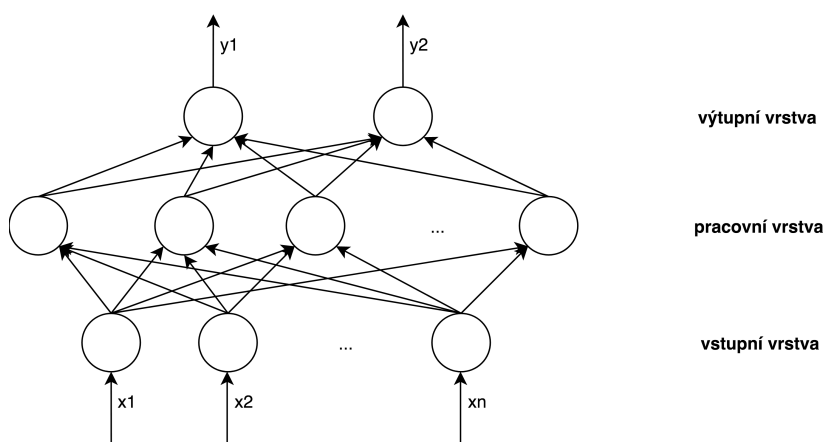


Obrázek 10.5: Jednoduchá neuronová síť pro OCR

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 [21].

Více vrstevné neuronové sítě jsou z hlediska použití mnohem univerzálnější. Na obr. 10.6 je znázorněn zjednodušený příklad takové sítě.



Obrázek 10.6: Schéma vícevrstevné 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/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 $x \in < 0; 10 >$ po celých číslech.

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

```
library("neuralnet")
mydata=read.csv("cesta/nn_fce_2mocnina.csv", sep=";", header=TRUE)
#trénování neuronové sítě
model=neuralnet(formula=output~input, data=mydata, hidden=10, threshold=0.01)
model_lm = lm(output~input, data=mydata) #pro srovnání lineární regrese
plot(model) #vykreslení vrstev sítě v (síťový graf)
#porovnání dat - skutečná a predikce
final_output=cbind(mydata$input, mydata$output,
                    as.data.frame(model$net.result),
                    as.data.frame(model_lm$fitted.values))
colnames(final_output) = c("Input", "Expeceted output", "NN Output", "LM Output")
plot(final_output$'Expeceted output', col="red", main="Predikce vs realita",
     xlab="čísla", ylab="druhá mocnina")
lines(final_output$'NN Output', col="blue")
lines(final_output$'LM Output', col="green")
legend(1,95, legend=c("skutečné hodnoty", "predikce NN", "predikce LM"),
      col=c("red", "blue", "green"), lty=1:3, cex=0.8)
```

Pro demonstraci použijeme toolkit *neuralnet* [31]. 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.

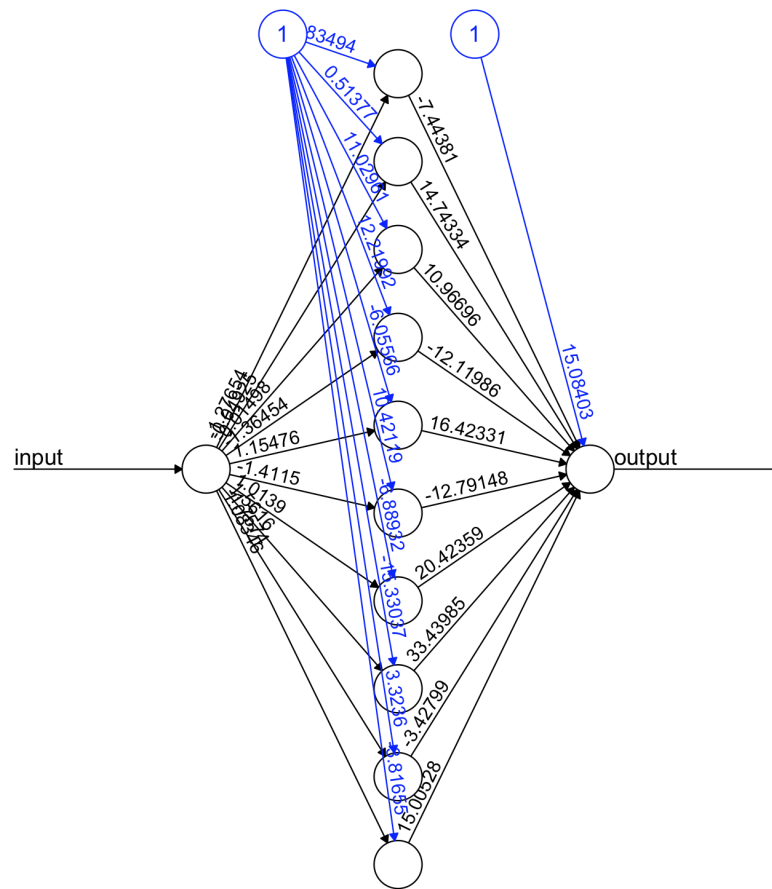
Prosím všimněte si, že chyba nemůže být nastavena na nula - důvodem je tzv. problém *přeučení* neuronové sítě. K problému přeučení se ale ještě vrátíme.

Struktura neuronové sítě je pak vykreslena pomocí funkce *plot*, viz obr. 10.7.

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.

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.

Obrázek 10.7: Neuronová síť pro predikci hodnot funkce $y = x^2$

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?

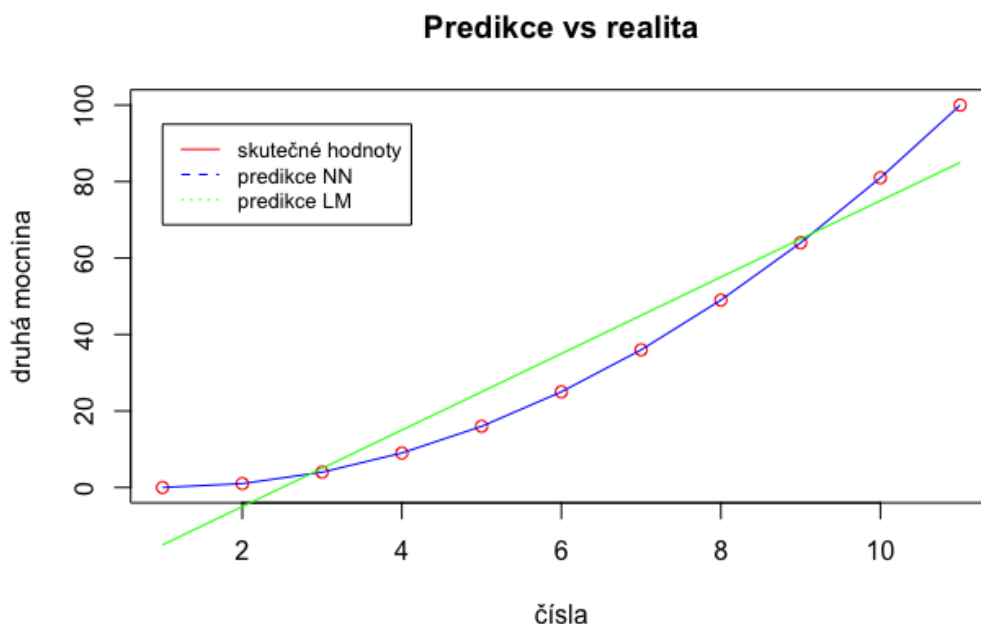
Pro adaptaci se většinou používá tzv. *učení s učitelem*. To znamená, že adaptačnímu algoritmu se předkládá trénovací množina ve formátu vstup $\rightarrow$ výstup, což umožňuje pro neuronovou síť porovnáním s jejím skutečným výstupem vypočítat chybu. Tato informace je pak použita pro další iteraci adaptace sítě.

Představit si to lze tak, že na začátku procesu adaptace je všem vazbám mezi neurony v síti přiřazena náhodná hodnota váhy w_n . Adaptací 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 (10.5).

$$GE = \sum_{i=1}^n (y_i - d_i)^2 \quad (10.5)$$

Kde GE je globální chyba (global error), y je očekávaný výsledek a d skutečný výsledek sítě.



Obrázek 10.8: Funkce $y = x^2$ - srovnání modelu neuronovou sítí a lineární regrese



Proč součet čtverců?

Podívejte se na vzorec 10.5 - proč myslíte, že je pro výpočet GE použit součet čtverců a ne např. $GE = \sum y_i - d_i$?

Z matematiky víme :-), že k nalezení minima funkce potřebujeme tuto funkci derivovat (10.6).

$$\Delta w = -\eta \frac{\partial GE}{\partial w} \quad (10.6)$$

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.

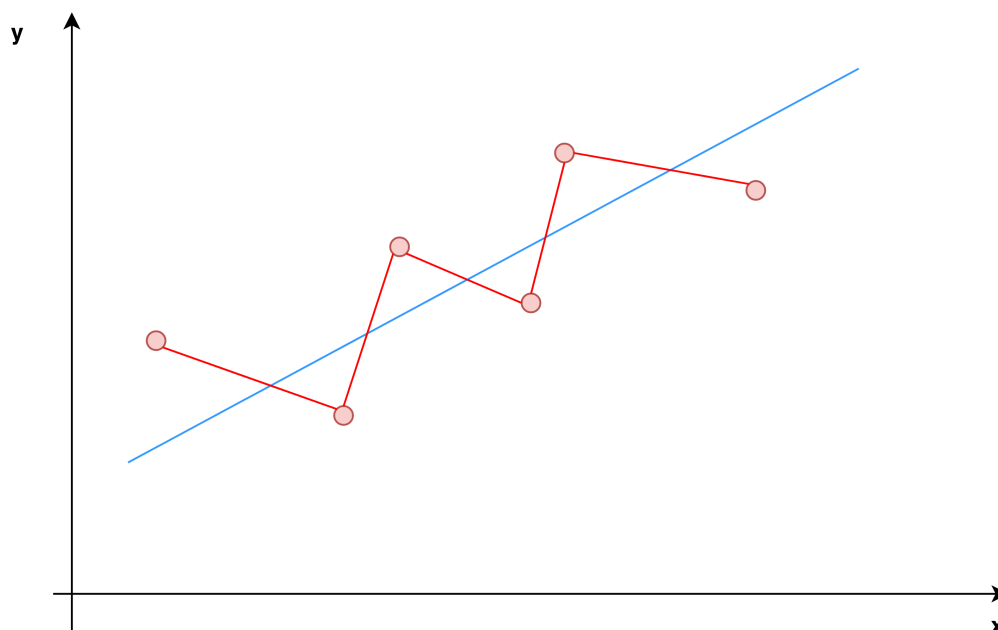
Graficky si to můžeme představit podobně jako na obr. 10.9.

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



Obrázek 10.9: Přeučený model

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.

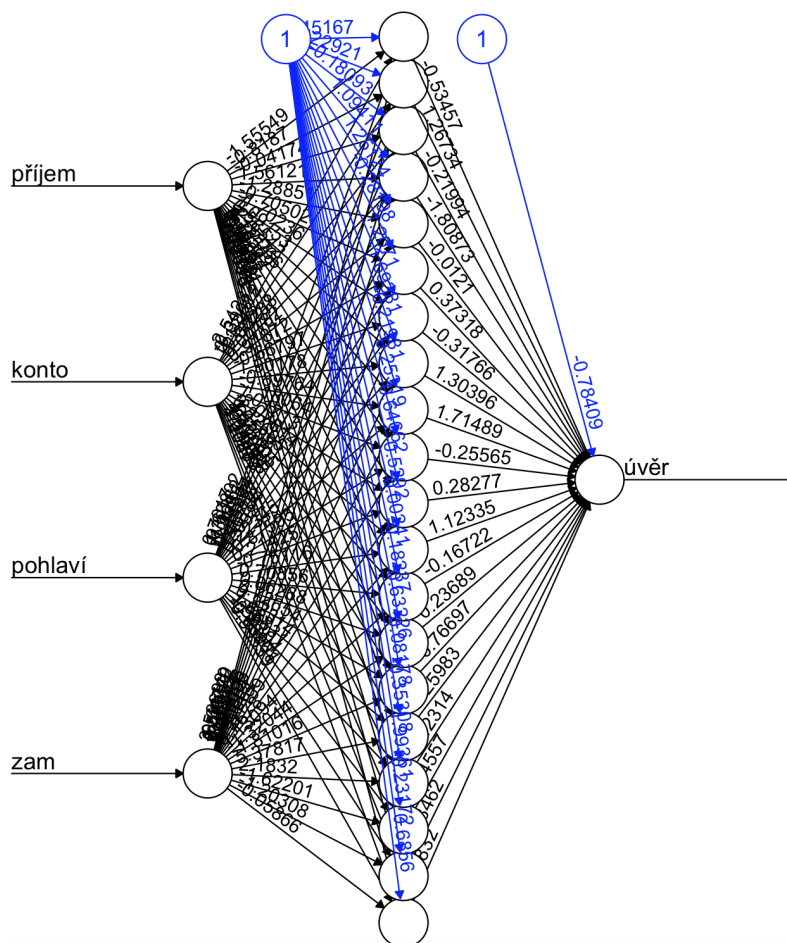
```
library("neuralnet")
mydata=read.csv("cesta/klienti.csv", sep=";", header=TRUE)
model=neuralnet(formula=úvěr~příjem+konto+pohlaví+zam,
                 data=mydata, hidden=20, threshold=0.01)
plot(model)
final_output=cbind(mydata$úvěr, as.data.frame(model$net.result))
colnames(final_output) = c("očekáváno", "spočteno")
final_output
```

Pouze v tomto případě jsem do pracovní vrstvy zvolil 20 neuronů. Vizuálně neuronová síť vypadá jako na obr. 10.10.

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:

	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



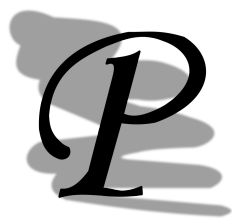
Obrázek 10.10: Klasifikace klientů banky neuronovou sítí

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.



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!



Shrnutí kapitoly

Neuronové sítě jsou v současnosti jednou z nejpopulárnějších a také nejuniverzálnějších 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).

Literatura

- [1] ISO/IEC 27001:2013 Information technology - Security techniques - Information security management systems - Requirements.
- [2] ISO/IEC 27002:2013 Information technology - Security techniques - Code of practice for information security management.
- [3] Zákon 365/2000 Sb. O informačních systémech veřejné správy.
- [4] zákon č. 181/2014 Sb., o kybernetické bezpečnosti a o změně souvisejících zákonů (zákon o kybernetické bezpečnosti).
- [5] zákon č. 239/2000 Sb., o integrovaném záchranném systému a o změně některých zákonů.
- [6] ČSN ISO/IEC 27005 Informační technologie - Bezpečnostní techniky - Řízení rizik bezpečnosti informací.
- [7] Zákon č. 480/2004 Sb. - Zákon o některých službách informační společnosti a o změně některých zákonů (zákon o některých službách informační společnosti), 2004.
- [8] Zákon č. 127/2005 Sb. - Zákon o elektronických komunikacích a o změně některých souvisejících zákonů (zákon o elektronických komunikacích), 2005.
- [9] Zákon č. 412/2005 Sb. - Zákon o ochraně utajovaných informací a o bezpečnostní způsobilosti, 2005.
- [10] Control panel with PLC, 2007.
- [11] Nařízení vlády č. 432/2010 Sb. - Nařízení vlády o kritériích pro určení prvku kritické infrastruktury, 2010.
- [12] Vyhláška č. 317/2014 Sb. o významných informačních systémech a jejich určujících kritériích, 2014.
- [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] Vyhláška č. 437/2017 Sb. - Vyhláška o kritériích pro určení provozovatele základní služby, 2017.
- [15] OpenAI, 2018.
- [16] R: The R Project for Statistical Computing, 2018.
- [17] Vyhláška č. 82/2018 Sb. o bezpečnostních opatřeních, kybernetických bezpečnostních incidentech, reaktivních opatřeních, náležitostech podání v oblasti kybernetické bezpečnosti a likvidaci dat (vyhláška o kybernetické bezpečnosti), 2018.
- [18] 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.
- [19] Petr Berka. *Dobývání znalostí z databází*. ACADEMIA, Praha, 2005.
- [20] Bruce Blaus. Neuron schéma, 2013. Page Version ID: 860153584.

- [21] 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.
- [22] Paul Boutin. Slammed! *Wired*, July 2003.
- [23] Comodo. Comodo Firewall | Get Best Free Personal Firewall Software, 2018.
- [24] CSIRT.CZ. Národní CSIRT České republiky, 2018.
- [25] DECCI. Servis FVE.
- [26] EC. Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on ENISA, the "EU Cybersecurity Agency", and repealing Regulation (EU) 526/2013, and on Information and Communication Technology cybersecurity certification ("Cybersecurity Act"), 2017.
- [27] ENISA. ENISA - homepage, 2018.
- [28] 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.
- [29] Nicolas Falliere, Liam O. Murchu, and Eric Chien. *W32.Stuxnet Dossier*. Symantec Corporation, Cupertino, 2011.
- [30] FIRST. Vision and Mission Statement, 2018.
- [31] Frauke Guenther. *Package 'neuralnet'*. 2016.
- [32] GÉANT Assoc. Trusted Introducer : Home, 2018.
- [33] Michael Hahsler, Christian Buchta, Bettina Gruen, and Kurt Hornik. *Package 'arules'*. 2018.
- [34] Suzana Herculano-Houzel. The Human Brain in Numbers: A Linearly Scaled-up Primate Brain. *Frontiers in Human Neuroscience*, 3, November 2009.
- [35] IBM. *IBM SPSS Modeler CRISP-DM Guide*. IBM, Armonk, NY, USA, 2016.
- [36] IBM. IBM SPSS Software | IBM Analytics, 2018.
- [37] Brian William Jones. Visual Cortex Neuron – Webvision, 2013.
- [38] Petr Kladivo. *Základy statistiky*. Univerzita Palackého v Olomouci, Olomouc, 2013.
- [39] 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.
- [40] Koordinační rada ministra dopravy pro kosmické aktivity. ITS v silniční dopravě - Český Kosmický Portál - Odbor ITS, kosmických aktivit a VaVaI.
- [41] Mohit Kumar. TSMC Chip Maker Blames WannaCry Malware for Production Halt, 2018.
- [42] Ted G. Lewis. *Critical Infrastructure Protection in Homeland Security*. Wiley, New Jersey, 2 edition, 2015.
- [43] 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.
- [44] 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.
- [45] Merriam-Webster. Definition of WAR, 2018.

- [46] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *Package ‘e1071’*. 2018.
- [47] Microsoft. Differences Between OLAP, ROLAP, MOLAP, and HOLAP - TechNet Articles - United States (English) - TechNet Wiki, 2018.
- [48] Microsys. SCADA/HMI systém PROMOTIC.
- [49] MV ČR. Počty ISVS | Informační systém o informačních systémech veřejné správy, 2018.
- [50] NAO. *Investigation: WannaCry cyber attack and the NHS*. NAO, Londýn, 2018.
- [51] NÚKIB. NÚKIB - homepage, 2018.
- [52] NÚKIB. Poskytované služby, 2018.
- [53] Národní úřad pro kybernetickou a informační bezpečnost. GovCERT.CZ, 2018.
- [54] Dale Peterson. Offensive Cyber Weapons: Construction, Development, and Employment. *Journal of Strategic Studies*, 36(1):120–124, February 2013.
- [55] R Project. Ubuntu Packages for R, 2018.
- [56] Thomas Rid. *Cyber War Will Not Take Place*. C Hurst & Co Publishers Ltd, London, 2013.
- [57] Brian Ripley. *Package Tree*. 2018.
- [58] RISI. Slammer Impact on Ohio Nuclear Plant, 2003.
- [59] RISI. Browns Ferry Nuclear Plant Scrammed (Shut Down), 2006.
- [60] Paul F. Roberts. Zotob, PnP Worms Slam 13 DaimlerChrysler Plants, 2005.
- [61] SAS. *Data Mining Using SAS® Enterprise Miner™: A Case Study Approach*. SAS Institute Inc., Cary, NC, USA, 3 edition, 2013.
- [62] SAS. Analytics, Business Intelligence and Data Management, 2018.
- [63] Secunia. Secunia Research Community - Advisories, 2018.
- [64] Tony Smith. Hacker jailed for revenge sewage attacks, 2001.
- [65] Darlene Storm. Gauss malware: Nation-state cyber-espionage banking Trojan related to Flame, Stuxnet, August 2012.
- [66] Symantec. W32.SQLEXP.Worm, 2003.
- [67] Symantec. *W32.Duqu: The precursor to the next Stuxnet*. Symantec Corporation, Cupertino, 2011.
- [68] Symantec. W32.Flamer, 2012.
- [69] Symantec. Ransom.Wannacry, 2017.
- [70] Twitter. Twitter API Docs, 2018.
- [71] VARS. Národní dopravní a informační centrum, 2018.
- [72] W3Schools. SQL Tutorial.
- [73] Kim Zetter. *Countdown to Zero Day: Stuxnet and the Launch of the World’s First Digital Weapon*. Broadway Books, New York, 2015.
- [74] Pavel Šenovský. *Bezpečnostní informatika 1*. VŠB-TU Ostrava, Fakulta bezpečnostního inženýrství, Ostrava, 8 edition, 2017.
- [75] Pavel Šenovský. *Bezpečnostní informatika 1 - Návod do cvičení*. VŠB-TU Ostrava, Fakulta bezpečnostního inženýrství, Ostrava, 2017.

Slovník

AJAX Asynchronous JavaScript and XML.

ASIC Application Specific Integrated Circuit.

ASUM-DM Analytics Solutions Unified Method for Data Mining/Predictive Analytics.

CERT Computer Emergency Response Team.

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.

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.

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.

IPS Intruder Prevention System.

ISMS Information Security Management System.

ISVS Informační Systém Veřejné Správy.

IT informační technologie.

JDBC Java Database Connect.

KDD Knowledge Discovery in Databases.

KII Kritická informační infrastruktura.

MIS Management Information System.

MOLAP Multidimensional Online Analytical Processing.

NAO National Audit Office.

NBÚ Národní bezpečnostní úřad.

NCKB Národní centrum kybernetické bezpečnosti.

NDIC Národní dopravní a informační centrum.

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.

ROLAP Relational Online Analytical Processing.

RTU Remote Terminal Unit.

SCADA Supervisory Control and Data Acquisition.

SCM Supply Chain Management.

SLA Service Level Assurance.

SQL Structured Query Language.

TDIDT Top Down Induction of Decision Trees.

TI Trusted Introducer.

TSMC Taiwan Semiconductor Manufacturing Company.

VIS Veřejný informační systém.

ČR Česká republika.

[title=Seznam zkratk]

Rejstřík

řešení

optimální, 108
suboptimální, 108

air gap, 54

aplikace bezpečnostních záplat, 51

asociační pravidla, 108

automatizace, 13

Bayesův teorém, 115

bezpečnost přípojných bodů, 50

bezpečnostní opatření, 69

organizační, 69

technická, 69

CERT, 63

certifikační rámec kyberbezpečnosti EU, 66

CSIRT, 64

CSIRT.CZ, 64

data historian, 23, 56

data mining, 73

deskripce, 76

hledání nugetů, 76

klasifikace, 76

predikce, 76

databáze

realtimová, 23

relační, 23

transakce, 24

datová krychle

hyperkrychle, 82

multikrychle, 82

datové typy

BLOB, 27

DATETIME, 27

DOUBLE, 27

INT, 27

TEXT, 27

VARCHAR, 27

datový sklad, 81

dispečerské pracoviště, 16

dohledové pracoviště, 16

dolování dat, 73

dual home computer, 56

dual home server, 56

ENISA, 65

ERD, 80

FIRST, 65

ginni index, 105

HMI, 18

honeypot, 67

informační entropie, 101

Informační systém veřejné správy, 71

ISMS, 70

ISVS, 71

klastr, 24

kritéria

dopadová, 70

oblastní, 70

kybernetická bezpečnostní událost, 69

kybernetická válka, 53

kybernetický bezpečnostní incident, 69

load balancer, 24

management rizik, 51

metodologie

5A, 78

CRISP-DM, 79

SEMMA, 78

Národní úřad pro kybernetickou a informační bezpečnost, 66

národní CERT, 64

naivní bayesovský klasifikátor, 116

NDIC, 16

nervový vzruch, 122

neuronová síť, 121

NoSQL, 82

NUKIB, 66

OLAP, 81

MOLAP, 82

ROLAP, 82

přeučení, 128

perceptron, 122

perimetr organizace, 54

PLC, 14

proměnná

nominální, 91

ordinální, 91

Promotic

- instalace, 33
 - PmData, 38
 - PmFolder, 38
 - PmNumber, 38
 - PmRoot, 37
 - PmString, 38
 - PmTimer, 40
 - PmWorkspace, 37
- R
- Data.Frame, 90
 - export dat, 91
 - faktor, 91
 - import dat, 91
 - instalace balíků, 106
 - operátory, 88
 - vektory, 89
 - vykreslování grafů, 92
 - základní funkce, 89
 - základní statistické funkce, 92
- real-timová databáze, 16
- regulační soustava, 13
- relační databáze, 80
- rozděl a panuj, 101
- rozhodovací pravidla, 99, 108
- rozhodovací seznam, 105
- rozhodovací strom, 128
- rozhodovací stromy, 99
- RTU, 17
- SCADA, 16
- senzor, 13
- sigmoida, 123
- služby elektronických komunikací, 68
- SQL, 26
- ALTER, 28
 - CREATE, 27
 - DELETE, 30
 - INSERT, 29
 - SELECT, 28
 - UPDATE, 29
- stav kybernetického nebezpečí, 69
- Stuxnet, 52
- tansakce
- ACID, 30
- TDIDT, 101
- TI, 65
- trénovací množina, 99
- transakce, 30
- Trusted Introducer, 65
- učení s učitelem, 127
- významná síť, 68
- významné informační systémy, 67
- významný informační systém, 70
- validační množina, 99
- Visual Basic
- deklarace proměnných, 46
 - dim, 46
 - if, 47
 - komentář, 47
 - objektová notace, 47
 - Pm, 47
 - podmínka, 47
 - rnd, 47
 - set, 47
- vrcholový CERT, 64
- vrcholový CSIRT, 64
- Vyhláška o kybernetické bezpečnosti, 70
- základní služba, 68