VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODUL PRO DOLOVÁNÍ Z DAT

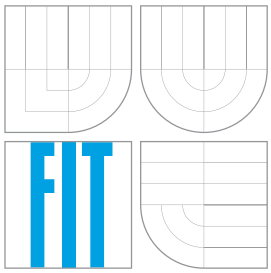
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN HLOSTA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODUL PRO DOLOVÁNÍ Z DAT

DATA MINING MODULE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN HLOSTA
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2008

Zadání a licenční smlouva jsou uvedeny v archivním výtisku uloženém v knihovně FIT VUT v Brně.

Abstrakt

Tato práce se zabývá problematikou získávání znalostí z databází (ZZD), a to zejména klasifikací pomocí Support Vector Machines (SVM). Na FIT VUT v Brně je vyvjíjen systém pro ZZD s modulární strukturou. Pro popis procesu dolování se používá jazyk DMSL. Cílem práce bylo rozšířit DMSL o potřeby SVM klasifikátoru, navrhnout, implementovat a otestovat modul pro tento systém.

Klíčová slova

získávání znalostí z databází, dolování z dat, DMSL, SVM, SMO, klasifikace, Java, MySQL

Abstract

This thesis concerns knowledge discovery in databases (KDD), especially classification by Support Vector Machines (SVM). System for KDD has been developed at FIT BUT. For KDD process description is used language DMSL. The goal of the thesis was to extend DMSL with respect to SVM classifier, propose, implement and test a module for this system.

Keywords

knowledge discovery in databases, data mining, DMSL, SVM, SMO, classification, Java, MySQL

Citace

Martin Hlosta: Modul pro dolování z dat, bakalářská práce, Brno, FIT VUT v Brně, 2008

Modul pro dolování z dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. Ing. Jaroslava Zendulky, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Hlosta
13. května 2008

Poděkování

Děkuji svému vedoucímu Doc. Ing. Jaroslavu Zendulkovi, CSc. za odborné vedení, cenné rady a podněty, které mi při řešení tohoto projektu poskytl.

Obsah

Kapitola 1

Úvod

Bakalářská práce se zabývá tématem získávání znalostí z databází(ZZD). Počítačové databáze, zejména ty založené na relačním modelu dat, se staly velice oblíbenými prostředky pro uchovávání persistentních dat. S rostoucím výkonem počítačů a zvětšováním diskových kapacit rostlo i množství dat v databázích ukládané. Po jisté době jsme byli zahlceni velkým množstvím dat, která neposkytovala žádné znalosti. Začaly se proto hledat způsoby, jak z dat získávat nové, netriviální, potenciálně zajímavé znalosti. O spojení databázových technologií s metodami statistiky a umělé inteligence se pak hovoří jako o ZZD.

V oblasti získávání znalostí z databází vzniká velké množství jak komerčních, tak zdarma dostupných systémů. Na FIT VUT v Brně se vyvíjí dva. Oba používají k popisu procesu získávání znalostí jazyk DMSL, taktéž vyvinutý na FIT. Oba mají rovněž modulární architekturu. Jeden spolupracuje s databází Oracle a druhý s MySQL. Úkolem práce bylo rozšířit systém přistupující k databázi MySQL o jeden modul. Po dohodě s vedoucím práce bylo zvoleno rozšíření klasifikačního modulu o SVM (Support Vector Machines) klasifikátor. Ten se v poslední době stává velmi populární zejména díky své přesnosti a srozumitelnému modelu výsledku.

Práce je systematicky členěna do osmi kapitol. Na tento úvod navazuje druhá kapitola, ve které jsou popsány základy získávání znalostí z databází. Protože je práce zaměřena na SVM klasifikátor, je třetí kapitola věnována uvedení do problematiky klasifikce a kapitola čtvrtá pak pojednává o samotném SVM. Je zde popsán princip tohoto modelu, jeho rozšíření a algoritmy, které řeší SVM problém. Větší význam je kladen na SMO (Sequential minimization optimization) algoritmus, který byl v rámci řešení práce implementován.

V páté kapitole je popsána koncepce systému získávání znalostí, který je vyvíjen na FIT VUT v Brně. Jedná se o strukturu systému a jazyk DMSL, sloužící pro popis dolovací úlohy.

Šestá kapitola popisuje návrh koncepce přídavného modulu a jeho začlenění do stávajícího systému a již existujícího klasifikačního modulu. Rovněž je popsán postup implementace.

Sedmá kapitola se soustřeďuje na testování na vhodném vzorku dat a zhodnocení dosažených výsledků.

Celá práce je pak shrnuta v závěru a jsou nastíněny možnosti dalšího vývoje.

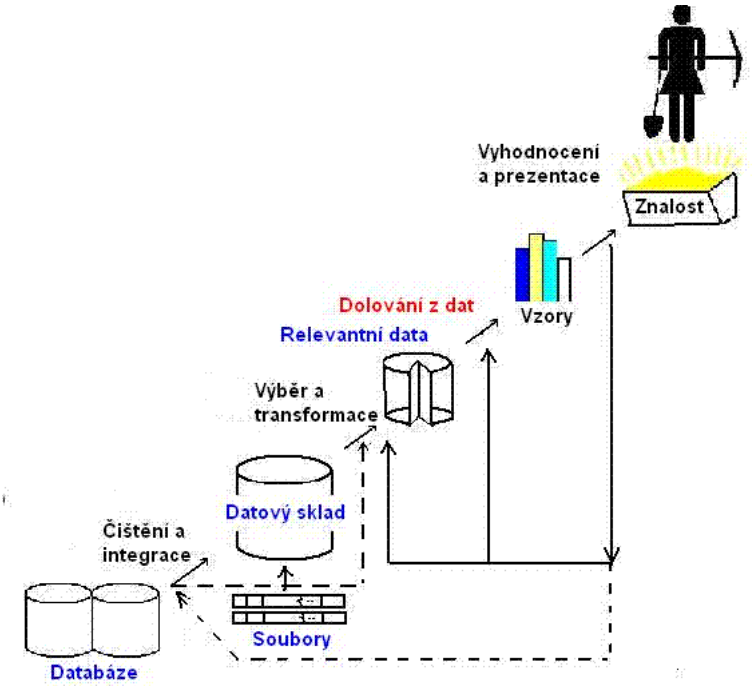
Kapitola 2

Získávání znalostí z databází

Získávání znalostí z databází je proces hledání zajímavých, tj. nových, netriviálních a potenciálně užitečných znalostí z dat, nejčastěji velmi rozsáhlých. Nalezené znalosti mohou být reprezentovány různými modely [14]. Znalostmi rozumíme informace dané do souvislostí.

2.1 Proces získávání znalostí

Proces získávání znalostí lze popsat jako sekvenci několika fází. Existuje několik mírně se lišících popisů těchto fází. Já použiji model z [14]. Je popsán sedmi fázemi, z nichž první čtyři se souhrnně označují jako předzpracování dat. Několik dalších modelů lze najít například v [1].



Obrázek 2.1: Proces získávání znalostí z databází, převzato z [14]

1. **Čištění dat** - odstranění šumu dat, řešení nekonzistence dat a chybějících dat.
2. **Integrace dat** - data se integrují z několika zdrojů do jednoho, například do jedné tabulky.
3. **Výběr dat** - pokud byla výsledkem integrace tabulka, zajímají nás v ní pouze některé sloupce. Úkolem této fáze je tyto sloupce vybrat.
4. **Transformace dat** - data jsou transformována do podoby vhodné pro dolování. Pokud například metoda vyžaduje na svém vstupu pouze numerická data, je potřeba na ně převést všechna data kategorická.
5. **Modelování - dolování z dat** - cílem je nad předzpracovanými daty aplikovat vybranou metodu s příslušným algoritmem a získat z nich zajímavé znalosti. Výsledné znalosti mohou sloužit opět jako vstup pro tuto fázi, například pro jinou metodu. Téma bude popsáno dále v sekci 2.2.
6. **Hodnocení modelu** - v závislosti na typu modelu existuje několik metrik, jak ohodnotit získané výsledky, znalosti.
7. **Prezentace znalostí** - výsledná znalost nemá pro uživatele smysl, pokud jí nerozumí. Proto je důležité zabývat se reprezentací znalostí, formátem výstupních dat a využít různé metody vizualizace.

2.2 Modelování - data mining

Jedná se o pátou fázi procesu získávání znalostí modelu z 2.1. Protože jde o jeho hlavní část, často se název data mining (dolování z dat) používá ve smyslu celého procesu získávání znalostí. Aplikací vybrané metody a algoritmu na předzpracovaná data se snažíme najít model (nebo více modelů), který reprezentuje výsledné znalosti. Podle modelu, který chceme získat, můžeme rozlišovat tyto typy dolovacích úloh[4].

- **Popis pojmů** - jedná se o dva typy: charakterizace a diskriminace. Charakterizace sumarizuje hlavní charakteristiky a znaky cílové datové třídy. Diskriminaci jde naopak o porovnávání rozdílů cílové třídy a jedné nebo více tříd ostatních.
- **Klasifikace** - Hledá se model (nebo funkce), který bude schopen správně zařazovat nová data do předem známých tříd. Model je založen na analýze trénovacích dat, u nichž je příslušnost ke třídě známa. Podrobněji bude probráno v kapitole 3.
- **Predikce** - je podobná jako klasifikace. Rozdíl je pouze v tom, že naučený model neklasifikuje nová vstupní data do tříd, ale snaží se pro ně předpovědět spojitou hodnotu (například předpověď teploty vzduchu).
- **Shluková analýza** - snaží se najít v prostoru dat místa, kde je hustota dat relativně vysoká. Tato místa se nazývají shluky. Nalezené shluky se často používají jako třídy pro klasifikaci.
- **Asociační analýza** - Hledají se vztahy mezi vstupními daty, reprezentovány asociačními pravidly. Zajímavá jsou pro nás data, která se spolu vyskytují často pohromadě.

2.3 Použití

Dolování z dat má využití jak ve vědě a výzkumu, tak i v obchodní sféře. Z vědního využití uvedu získávání znalostí z genetických dat (DNA, proteiny), nebo určování možnosti propuknutí nemoci, například rakoviny, u pacienta na základě symptomů. V obchodní sféře je známo použití analýzy nákupního košíku (jaké zboží si zákazník často kupuje dohromady), odhalování podvodů nebo spamu v elektronické poště na základě získávání znalostí z textu. Rovněž je znám pojem BI (Business inteligence). Jedná se o nástroje a technologie pro podporu rozhodování ve firmě. Sem patří i data mining.

2.4 Vývoj data miningu

Protože data mining se stále vyvíjí, zmíním na závěr této kapitoly některé trendy, jimiž se v poslední době ubírá [3].

- Integrace do databázových systémů (Oracle, MS SQLServer)
- Standardizace jazyka pro popis dolovací úlohy (např. jazyk PMML)
- Získávání znalostí z multimediálních dat
- Web mining - získávání znalostí z internetových stránek

Kapitola 3

Klasifikace

Klasifikace je jedním z typů modelování v procesu získávání znalostí z dat. Jejím cílem je najít na základě informací z trénovacích dat nejlepší model, který by správně zařazoval data nová do předem známých tříd.

3.1 Kritéria kladená na klasifikátory

Po výsledných klasifikátorech požadujeme tyto vlastnosti [4].

- **Přesnost předpovědi** - procento správně klasifikovaných nových neznámých dat. Bývá součástí testování popsaného v sekci 3.2.2.
- **Rychlost** - výpočetní cena tvorby modelu.
- **Robustnost** - schopnost vytvořit kvalitní model i přes zašuměná data a vypořádat se s chybějícími hodnotami.
- **Stabilita** - schopnost tvorby kvalitního modelu s velkými daty.
- **Interpretovatelnost** - srozumitelnost reprezentace výsledného modelu.

3.2 Fáze klasifikace

V procesu klasifikace můžeme hovořit o třech fázích - trénování, testování a klasifikace neznámých dat.

3.2.1 Trénování

Pokud chápeme data jako tabulku, pak jedním ze sloupců může být třída, ke které řádek přísluší. U trénovacích dat tuto příslušnost známe. Toho využíváme k nalezení nejlepšího možného modelu pro klasifikaci dat nových.

Cílem klasifikace není dosáhnout složitého modelu, jenž bude přesně popisovat trénovací data. Pokud velice dobře klasifikuje trénovací data, ale není schopen generalizace na datech neznámých, hovoří se o *přeučení*.

3.2.2 Testování

Než je natrénovaný model uveden do praxe na neznámá data, je jeho funkčnost ověřena pomocí testovacích dat. Model necháváme určit třídu a tu pak porovnáváme se správnou třídou, kterou známe. Informace o tom, zda se klasifikátor spletl nebo naopak určil třídu správně, se ukládá do *matice záměn (confusion matrix)*, uvedenou v tabulce 3.1. Tabulka ukazuje příklad zařazení do dvou tříd "+","-". TP (true positive) udává počet správně klasifikovaných příkladů do třídy "+", FN (false negative) počet špatně zařazených do třídy "-", FP počet špatně zařazených do "+" a TN počet správně zařazených do "-".

	Klasifikace systémem	
Správné zařazení	+	-
+	TP	FN
-	FP	TN

Tabulka 3.1: matice záměn (confusion matrix), převzato z [1]

Používá se několik metrik vypovídajících o správnosti naučeného modelu. Všechny se dají vyjádřit pomocí TP, TN, FP a FN z tabulky 3.1. U každé z nich uvedu pro ilustraci příklad, kdy na základě různých atributů (plat, věk) se snažíme zjistit, zda klient banky dostane nebo nedostane úvěr.

- **Celková správnost** = (TP + TN) / (TP + FP + TN + FN) - jaká část klientů byla správně klasifikována.
- **Celková chyba** = 1 − Celková správnost = (FP + FN) / (TP + FP + TN + FN) - jaká část klientů byla klasifikována chybně.
- **Přesnost** = TP / (TP + FP) - jaká část klientů z těch, kteří dostali úvěr, jej skutečně měli dostat.
- **Senzitivita (úplnost)** = TP / (TP + FN) - jaká část klientů z těch, kteří měli dostat úvěr, jej skutečně dostali.
- **Specificita** = TN / (TN + FP) - jaká část klientů z těch, kteří neměli dostat úvěr, jej skutečně nedostali.

Tyto metriky se dají spočítat několika různými druhy testů. Všechny fungují na podobném principu. Ze vstupních dat se použije určitá část pro natrénování a zbylá pro otestovaní.

- **Holdout** - Data se rozdělí v určitém poměru. Např. 2/3. Pro trénování se použijí 2/3 a zbylá 1/3 na testování. Opakováním tohoto testu a průměrováním výsledků se říká *Random subsampling*.
- **Křížová validace (Cross-validation)**. Data se rozdělí na k částí: $k-1$ se použije na trénování a 1 na testování. Toto se provede k-krát, přičemž pokaždé se vybere jiná část pro testování. Výsledky se sčítají. Speciálním případem je *Leave-one-out (Vypuštění jednoho)*. V tomto případě $k = 1$.
- **Bootstrap** - obdobný mechanismus jako u cross-validation. Při n krocích se vybere pokaždé vždy jeden příklad pro testování a ostatní pro trénování. Rozdíl je v tom, že výběr je náhodný, a příklad se tedy může vybrat i několikrát.

3.2.3 Aplikace

Ve chvíli, kdy jsme spokojeni s výsledky testování, můžeme natrénovaný model uvést do praxe a použít jej pro klasifikaci neznámých dat.

3.3 Klasifikační modely

Existuje několik různých modelů pro klasifikaci. Protože žádný z nich není jednoznačně nejlepší, nedá se univerzálně zvolit pro řešení všech problémů a stojí za to si několik hlavních modelů představit.

3.3.1 Rozhodovací stromy

Cílem je vytvoření rozhodovacího stromu. V něm představuje každý nelistový uzel test na atribut, větve z těchto uzlů vedoucí znamenají výsledky těchto testů a listové uzly značí klasifikační třídu. Rozhodování začíná vždy od kořene a pokračuje podle výsledků v uzlech směrem k listům. Při trénování se hledají atributy, které mají na klasifikaci největší vliv, a ve stromu se umisťují co nejvýše (směrem ke kořenu).

Metoda je používána pro dobrou interpretovatelnost, dobrou přesnost i pro větší dimenzi dat a také nepotřebuje žádné parametry. Vhodná je pro kategoriální data. Numerická je potřeba transformovat.

3.3.2 Bayesovská klasifikace

Jedná se o metodu založenou na statistice. Při klasifikaci se využívá Bayesův vzorec. Pro každý příklad se pomocí pravděpodobnosti, že bude mít stejné atributy jako výsledná třída, spočítá pravděpodobnost, že do třídy patří. Příklad je zařazen do třídy s největší vypočtenou pravděpodobností.

3.3.3 Neuronové sítě

Jde o metodu strojového učení. Princip je podobný jako u neuronů z nervové soustavy. Z neuronů, které mají vstupy a výstup, je sestavena síť. Při učení metodou *backpropagation* hledáme vhodné nastavení parametrů vstupů jednotlivých neuronů tak, že trénovací příklad necháme "projít" sítí a určit výslednou třídu. Porovnáním výsledku této klasifikace a skutečné třídy šíříme vzniklou chybu zpět a podle ní přenastavujeme parametry.

Neuronové sítě dosahují přesných výsledků, jejich nevýhoda je špatná interpretovatelnost a jejich natrénování trvá delší dobu.

3.3.4 SVM - Support Vector Machines

Podobně jako se snažíme najít v neuronových sítích parametry funkcí jednotlivých neuronů, zde hledáme funkci nadroviny, která by data co nejvíce rozdělovala. Tento model jsem si vybral pro implementaci do systému a je mu proto věnována celá kapitola 4.

3.3.5 Další modely

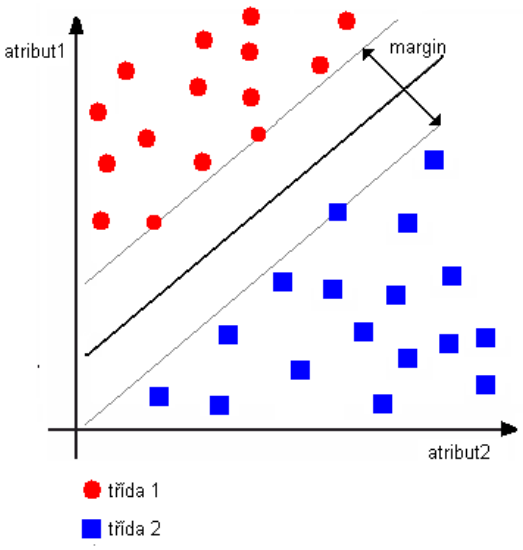
Představené modely nejsou všechny existující. Z dalších uvádím klasifikaci pomocí fuzzy množin, pomocí rozhodovacích pravidel, klasifikaci založenou na k-nejbližším sousedství nebo klasifikaci pomocí genetických algoritmů.

Kapitola 4

SVM

Support vector machines (SVM) je jedním z klasifikačních modelů, který se v poslední době stal velice oblíbeným díky přesnosti výsledků, dobré generalizaci (metoda není příliš náchylná k přeuční) a také interpretovatelnosti. Svědčí o tom i 3.místo v anketě oblíbenosti dolovacích algoritmů na konferenci ICDM v roce 2006 [3].

Jedná se o matematický model a slouží pro klasifikaci numerických atributů. SVM vynalezl Vladimir Vapnik v roce 1979. Ve své základní verzi zařazuje pouze do dvou tříd. Pokud se díváme na data jako na body reprezentovány vektory v n-rozměrném prostoru, hlavní myšlenkou SVM je najít mezi nimi nadrovinu, která bude rozdělovat prostor dat na dva podprostory. Požadavek na rozdělující nadrovinu je, aby vzdálenost od dat různých tříd byla co největší. Tento odstup se označuje jako *margin*. Hovoří se o *maximal marginal hyperplane (MMH)* - největší rozdělující nadrovina. Z pojmu nadrovina vyplývá, že jsme schopni klasifikovat pouze lineárně oddělitelná data. SVM je schopno díky rozšíření pracovat i s lineárně neoddělitelnými daty, což je tématem sekce 4.2. Lineární SVM problém je nejlépe ilustrovatelný na dvoudimenzionálních datech - obrázek 4.1.



Obrázek 4.1: Lineární SVM problém pro 2-D data

4.1 Lineární problém

Při řešení lineárního problému (obr. 4.1) hledáme rovnici

$$u = \vec{w} \cdot \vec{x} - b \tag{4.1}$$

kde $\vec{w}$ je normálový vektor vah nadroviny, $\vec{x}$ vstupní vektor a b absolutní člen (práh). Pro 2-D data by to bylo.

$$-b + w_1 x_1 + w_2 x_2 = 0 \tag{4.2}$$

Data z třídy, ležící nad, respektive pod rovinou, budou vyhovovat rovnici:

$$-b + w_1 x_1 + w_2 x_2 > 1, \quad resp. -b + w_1 x_1 + w_2 x_2 < 1 \tag{4.3}$$

Odvozením z těchto rovnic dostaneme tvar

$$y_i(-b + w_1 x_1 + w_2 x_2) >= 1, \forall i \tag{4.4}$$

kde y_i je výsledná třídá +1 nebo -1. Nejbližší bod leží ve vzdálenosti $u = \pm 1$. Těmto bodům se říká *support vectors* a daly název celé metodě. V rovnici (4.4) by se rovnaly jedné. Vzdálenost mezi rovinou a daty je rovna $\frac{1}{||W||}$ a velikost mezi daty z různých tříd tedy $\frac{2}{||W||}$, kde $||W|| = \sqrt{W \cdot W}$. Jak bylo zmíněno, úkolem je najít nadrovinu s největším odstupem od dat. To lze zapsat jako optimalizační problém [12]

$$\min \frac{1}{2}||w||^2 \quad \text{vzhledem k} \quad c_i(w \cdot x_i - b) \geq 1, \forall i. \tag{4.5}$$

4.1.1 Duální forma a Lagrangián

Použitím Lagrangiánu lze přepsat rovnici (4.4) do tzv. duální formy, kdy jde vidět, že rovnice je funkcí pouze support vektorů. To hraje významnou roli při klasifikaci neznámých dat, protože není potřeba počítat vektorový součin všech vektorů, ale jen těch nejblíže MMH.

$$\sum_{i=1}^{N} \sum_{j=1}^{N} y_i \alpha_i \vec{x_i} \vec{x}^T + b_0 \tag{4.6}$$

N je počet trénovacích dat a α_i Lagrangovy multiplikátory. Zadání optimalizačního problému pak má tvar

$$\min \Psi(\vec{\alpha}) = \min \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \vec{x_i} \vec{x_j} \alpha_i \alpha_j - \sum_{i=1}^{N} \alpha_i \tag{4.7}$$

$$\text{vzhledem k} \quad 0 \leq \alpha_i \forall i, \sum_{i=1}^{N} y_i \alpha_i = 0. \tag{4.8}$$

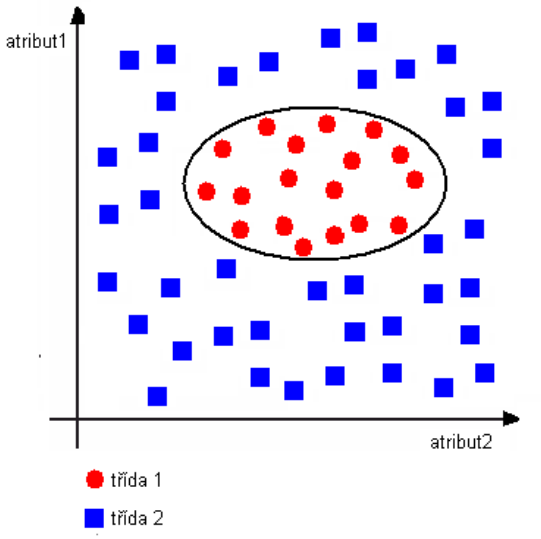
4.2 Rozšíření

Protože data nejsou vždy lineárně oddělitelná, bylo potřeba se s tímto problémem vyrovnat. Výsledkem byla tato dvě rozšíření.

4.2.1 Soft margin

Toto rozšíření zavádí pojem chyby. Pokud SVM není schopno najít správnou nadrovinu, která bude vyhovovat všem datům, je připuštěno několik, které vyhovovat nemusí. Zavádí se přídavná proměnná ξ, která povoluje chybu a parametr C, který řeší kompromis mezi velkým odstupem od dat a malý počet chyb. Menší C povoluje větší nepřesnost. V duální formě se pouze upraví omezující podmínka v (4.8) na $0 \leq \alpha_i \leq C, \forall i$.

4.2.2 Jádrové funkce



Obrázek 4.2: Lineárně nerozdělitelná data

S problémem uvedeným na obrázku 4.2 by se předchozí rozšíření nevyrovnalo. Druhým způsobem, jak si poradit s nelinearitou, je transformace vstupních dat do jiného prostoru, kde už lineárně oddělitelná jsou.

Tyto transformace probíhají často do prostoru s vyšší dimenzí. To by zapříčinilo nárůst výpočetního času při vyhodnocování vektorových součinů v rovnici 4.6 jak při učení, tak i klasifikaci. Využívá se proto faktu, že vektory se zde vyskytují právě pouze ve formě vektorového součinu. Existují transformační funkce, pro které platí:

$$K(\vec{x_i}, \vec{x_j}) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x_j}) \tag{4.9}$$

kde $\Phi(\vec{x})$ je mapování do jiného prostoru a $K(\vec{x_i}, \vec{x_j})$ je funkce, nazývaná jádrová *(kernel function)*. Touto funkcí se nahradí mapování a výpočty vektorového součinu probíhají v původním prostoru dat s nižší dimenzí a jsou tedy rychlejší. Nejčastěji se používají tyto jádrové funkce.

- Polynomiální $K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^h$
- Radial-basis $K(\vec{x_i}, \vec{x_j}) = e^{||\vec{x_i} - \vec{x_j}||^2 / 2\sigma^2}$
- Sigmoidální $K(\vec{x_i}, \vec{x_j}) = tanh(\kappa \vec{x_i} \cdot \vec{x_j} - \delta)$

4.2.3 Další rozšíření

SVM se dá rovněž použít pro klasifikaci do více tříd. Pro m tříd se natrénuje celkem m modelů, pro každou třídu jeden. Každý model potom bude vracet pro třídu, kterou reprezentuje, kladné výsledky a pro ostatní hodnoty záporné. SVM může být navrženo i pro predikci. Na rozdíl od klasifikace se hledá vztah mezi vstupními n-ticemi a výstupní spojitou hodnotou.

4.3 Škálování

Škálování původních hodnot do menšího intervalu je důležité ze dvou důvodů. Brání atributům s velkým rozpětím dominovat nad těmi s menším rozpětím. Druhým důvodem je snížení rizika výskytu problémů při výpočtu jádrových funkcí, které mohou způsobit velké vektorové součiny těchto dat. Nejčastěji se doporučuje transformace do intervalů $[0, 1]$ nebo $[-1, +1]$ [5].

4.4 Řešení problému

Použitím výše uvedených rozšíření Soft margin a jádrových funkcí má výsledný kvadratický optimalizační problém tuto podobu:

$$\min\Psi(\vec{\alpha}) = \min\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j K(\vec{x_i}\vec{x_j})\alpha_i\alpha_j - \sum_{i=1}^{N}\alpha_i \tag{4.10}$$

$$0 \leq \alpha_i \leq C, \forall i \tag{4.11}$$

$$\sum_{i=1}^{N} y_i\alpha_i = 0 \tag{4.12}$$

Hlavním úkolem algoritmů je tento problém vyřešit. O optimální bod se jedná tehdy, když platí Karush-Kahn-Tuckerovy (KKT) podmínky (4.13).

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y_i f(x_i) \geq 1 \quad a \quad \xi = 0 \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y_i f(x_i) = 1 \quad a \quad \xi = 0 \\ \alpha_i = C \quad &\Rightarrow \quad y_i f(x_i) \leq 1 \quad a \quad \xi \geq 0 \end{aligned} \tag{4.13}$$

Původní metody, které řešily kvadratický optimalizační problém spotřebovaly příliš mnoho paměti, protože při výpočtu potřebují matici o velikosti druhé mocniny počtu trénovacích příkladů. To při velkých datech činilo problém.

Proto se začaly hledat další metody. Metoda, zvaná "chunking", využívá faktu, že z výše uvedené matice lze vyjmout příklady s nulovou hodnotou Lagrangových multiplikátorů α. Potom je proces rozdělěn na sérii podprocesů. V průběhu každého je přidáno k příkladům s nenulovými multiplikátory určitý počet příkladů, které nevyhovují KKT. Po posledním kroku vyhovují všechny příklady KKT a problém je vyřešen.

Osunův teorém dokazuje, že pokud se rozdělí celý problém na několik podproblémů, a během každého z nich je přidán minimálně jeden příklad odporující KKT podmínkám, bude algoritmus konvergovat. Tomuto vyhovuje jak předchozí metoda, tak i ta navržená Osunou. Ta navrhuje optimalizaci kvadratického problému se stále stejným počtem příkladů, přičemž se vždy určitý počet přidá a stejný počet i odebere.

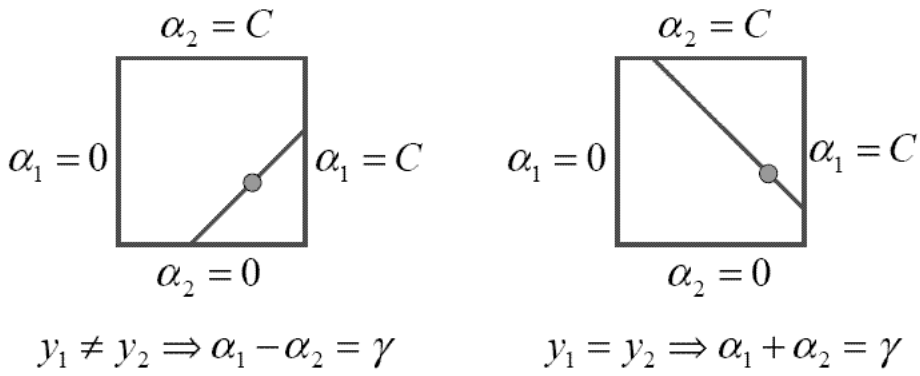
4.5 SMO algoritmus

SMO (Sequential Minimal Optimization) vymyslel a publikoval John C.Platt [9],[8]. Poměrně rychle řeší SVM problém. Všechny předchozí algoritmy potřebují k celkovému řešení výpočet kvadratického optimalizačního problému pomocí numerických metod, které přináší vždy úskalí. SMO toto nepotřebuje.

Stejně jako předchozí metody rozděluje řešení na sérii podproblémů. Vždy ale řeší optimalizaci pouze dvou příkladů, a to analyticky. V každém kroku se vybírá nejlepší vhodná dvojice pro optimalizaci. Z toho plynou dvě hlavní části algoritmu.

4.5.1 Analytické řešení dvojice Lagrangových multiplikátorů $\alpha_1\alpha_2$

Omezení kladená na optimalizaci zapřičiňují, že optimální bod bude ležet ve čtverci na diagonále 4.3. Nejprve se řeší α_2 a jsou vyjádřeny hranice čtverce, kde protíná diagonálu, L(low) a H(high). Výpočty se liší v závislosti na tom, jestli se výsledné třídy y rovnají nebo liší.



Obrázek 4.3: Omezení kladená na optimalizaci. První (4.11) znamená že bod leží uvnitř čtverce a druhé (4.12), že bude na diagonále. Převzato z [8]

.

$$L = max(0, \alpha_2 - \alpha_1) \quad H = min(C, C + \alpha_2 - \alpha_1) \quad y_1 = y_2 \tag{4.14}$$

$$L = max(0, \alpha_2 + \alpha_1 - C) \quad H = min(C, \alpha_2 + \alpha_1) \quad y_1 \neq y_2 \tag{4.15}$$

Druhou derivaci účelové funkce podél diagonály lze vyjádřit jako

$$\eta = 2K(\vec{x_1} - \vec{x_2}) - K(\vec{x_1}, \vec{x_1}) - K(\vec{x_2}, \vec{x_2}) \tag{4.16}$$

η je ve většině případů záporná a pak je α_2 spočítáno jako

$$\alpha_2^{nove} = \frac{\alpha_2 + y_2 + y_2(E_1 - E_2)}{\eta} \tag{4.17}$$

kde $E1$ a $E2$ jsou chyby, kterých se momentálně naučený model dopouští na optimalizovaných příkladech. Pro zvýšení rychlosti se uchovávají hodnoty v pomocném vektoru. Pokud je nové α_2 mimo čtverec, ve kterém se má nacházet, je ořezáno na H pokud leží nad horní hranicí, případně na L. Nové α_1 je dopočítáno:

$$\alpha_1^{nove} = \alpha_1 + y_1 y_2(\alpha_2^{nove} - \alpha_{2,orezane}^{nove}) \tag{4.18}$$

Ve zvláštních případech se stane, že η není negativní. To nastává, pokud více příkladů má stejný vektor $\vec{x}$. SMO potom spočítá hodnoty účelové funkce na koncích úsečky a vybere z nich nejvyšší hodnotu. Pokud se tyto hodnoty rovnají, pak optimalizace nemůže pokročit.

Po optimalizaci dvojice příkladů se provádí aktualizace absolutního členu b a vektoru chyb. Pro lineární jádro se dá urychlit pracné počítání výstupu klasifikátoru, které se počítá při zjišťování chyb. Využije se možnosti spočítat ze support vektorů váhový vektor používaný v původním řešení a pomocí něho zjišťovat výstup klasifikátoru (4.4). Tento váhový vektor se na konci každé optimalizace aktualizuje.

4.5.2 Výběr příkladů

Algoritmus běží ve dvou cyklech - vnější a vnitřní. Ve vnějším se nejdříve vybírají příklady, které nesplňují KKT podmínky. Cyklus potom běží znova a vybírá už pouze ty, jejichž Lagrangovy multiplikátory jsou různé od 0 a C. Tyto cykly se střídají. Algoritmus je ukončen ve chvíli, kdy během cyklu prověřujícího všechny příklady ani jeden z nich neodporuje KKT podmínkám.

Ve vnitřním cyklu se pomocí heuristiky vybírá k prvnímu příkladu ten nejlepší možný k optimalizaci. SMO se zde snaží maximalizovat velikost optimalizačního kroku. Bude hledat k prvnímu příkladu 1 takový příklad 2, pro který platí, aby $|E_1 - E_2|$ bylo co největší. Při tomto hledání využívá uložený vektor chyb. Pokud optimalizace pro tuto heuristiku nepřinese výsledek, hledá se druhý příklad mezi příklady s multiplikátory různých od 0 a C. Prochází se jeden po druhém a zkouší se, jestli optimalizace přinese výsledek. Pokud ani pro jeden z těchto příkladů se algoritmus nepohne vpřed, probíhá cyklus znova, ale tentokráte přes úplně všechny příklady. V případě, že ani nyní by se optimalizace nepovedla, pak je potřeba v předchozím vnějším cyklu vybrat jiný první příklad.

Kapitola 5

Koncepce systému pro získávání znalostí na FIT

Systém, jehož rozšíření jsem v rámci bakalářské práce řešil, je vyvíjen studenty na Fakultě informačních technologií Vysokého učení technického v Brně. Má modulární architekturu, umožňující jeho snadnou rozšiřitelnost.

Pro implementaci byl zvolen programovací jazyk Java pro svou jednoduchost, platformovou nezávislost a snadnost programování. Pro přístup k databázi využívá knihovnu JDBC (Java Database Connectivity). Ta umožňuje přístup do jakékoliv databáze, pokud tato databáze poskytuje JDBC ovladač. V současnosti je v systému pro ukládání dat použita relační, volně dostupná databáze MySQL.

Z toho plynou požadavky pro uživatele. Pokud tento dolovací systém chce použít, potřebuje Java runtime environment pro běh javovské aplikace. MySQL server, ke kterému program přistupuje, musí mít nainstalovaný JDBC ovladač.

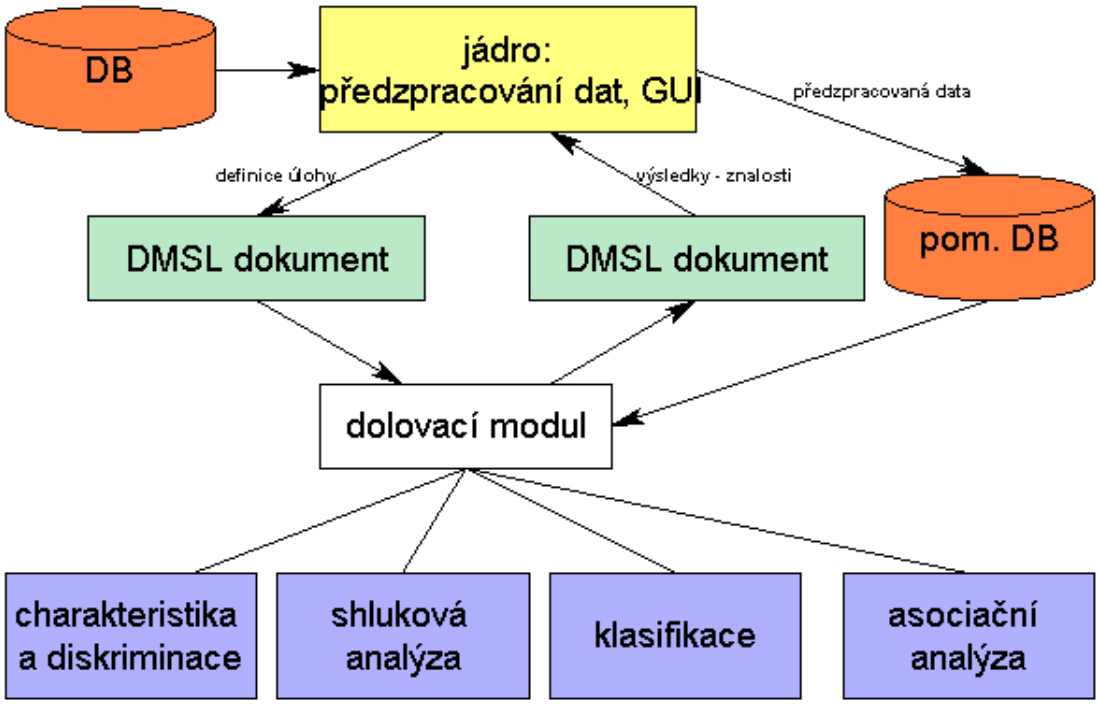
5.1 Struktura

Stěžejní část systému představuje jádro, které zajišťuje přezpracování dat a grafické uživatelské rozhraní pro zadání dolovací úlohy a vizualizací výsledků. Podle typu dolovací úlohy potom předává řízení jednomu z modulů, který provede dolování a vrací jádru výsledek, který zobrazí uživateli pomocí vhodné vizualizace. Jádro komunikuje s ostatními moduly pomocí jazyka DMSL, který slouží pro popis celé dolovací úlohy, více v 5.2. Protože MySQL databáze nepodporuje přímo dolování v databázi, jsou předzpracovaná data uložena do pomocné tabulky, nad kterou probíhá dolování. Struktura je znázorněna na obrázku 5.1.

5.1.1 Existující moduly

V současné době obsahuje systém tyto dolovací moduly.

- **Asociační analýza** - řešená pomocí algoritmu Apriori
- **Popis pojmů** - realizace úloh charakterizace a diskriminace
- **Klasifikace** - rozhodovací stromy pomocí základního algoritmu ID3.
- **Shluková analýza** - implementace algoritmů K-Means a BIRCH.



Obrázek 5.1: Současná struktura systému pro získávání znalostí na FIT

Více informací o modulech popisu pojmů a klasifikace lze najít v diplomové práci slečny Ševčíkové [11], o shlukové analýze pak v diplomové práci pana Forgáče [2].

5.2 DMSL

DMSL je jazyk odvozený z XML. Buduje se na FIT a jeho autorem je Petr Kotásek, který ho vytvořil v rámci disertační práce [6]. Jazyk byl navržen k popisu celého procesu získávání znalostí z databází, tedy od předzpracování dat přes tvorbu modelu až po prezentaci výsledků. Velký důraz je přitom kladen na předzpracování dat a jejich transformaci.

5.2.1 Struktura

DMSL se skládá z pěti základních částí, které kopírují průběh získávání znalostí.

1. **Data model (datový model)** - popis zdrojových dat
2. **Data mining model (dolovací model)** - popis dat použitých k dolování. Jsou zde popsány i transformace vstupních dat.
3. **Domain knowledge (doménová znalost)** - popis znalosti, kterou mohou využívat dolovací moduly.
4. **Data mining task (dolovací úloha)** - popis dolovací úlohy.
5. **Knowledge (znalost)** - popis výsledné znalosti.

Každá z částí je reprezentována vlastním XML elementem. Společně s elementem `FunctionPool` pro uložení funkcí použitých na data a volitelným elementem `Header` jsou uloženy v hlavním elementu dokumentu `DMSL`. Jazyk DMSL je popsán pomocí DTD. Hlavní element má v DTD tuto podobu:

```
<!ELEMENT DMSL (Header?, (FunctionPool | DataModel | DataMiningModel |
  DomainKnowledge | DataMiningTask | Knowledge)+) >
```

Z této definice plyne, že hlavní element smí obsahovat maximálně jeden element `Header` a alespoň jeden z elementů `FunctionPool`, `DataModel`, `DataMiningModel`, `DomainKnowledge`, `DataMiningTask` a `Knowledge`.

Pokud vytvářím nový přídavný modul, zajímají mě elementy `DataMiningTask` pro popis úlohy a element `Knowledge` pro prezentaci výsledných znalostí. V nich se nacházejí podle typu dolovacího modelu příslušné elementy definující tento typ. Tyto elementy v DMSL nejsou popsány, a proto je bude nutné pro potřeby SVM klasifikátoru dodefinovat. Struktura zmíněných dvou elementů vypadá následovně.

5.2.2 Element DataMiningTask

```
<!ELEMENT DataMiningTask ANY >
<!ATTLIST DataMiningTask
    name                CDATA                   #REQUIRED
    type                CDATA                   #IMPLIED
    dataMiningModelRef  CDATA                   #IMPLIED >
```

Význam atributů:

- **name** - název dolovací úlohy
- **type** - typ dolovací úlohy, u klasifikace se uvádí *clasification*
- **dataMiningModelRef** - identifikace dolovacího modelu, který používá dolovací úloha

Klasifikační modul už obsahuje model rozhodovacích stromů. Protože to byl jediný modul, nebylo potřeba rozlišit, o který se jedná. Kromě toho měl původní modul jednu zásadní vadu. Nedalo se v něm explicitně určit, který atribut z dolovacích dat v datové matici bude klasifikační.

5.2.3 Element Knowledge

```
<!ELEMENT Knowledge
    name                CDATA                   #REQUIRED
    type                CDATA                   #IMPLIED
    dataMiningTaskRef   CDATA                   #IMPLIED >
```

Význam atributů:

- **name** - název znalosti
- **type** - typ znalosti, u klasifikace se uvádí *clasification*
- **dataMiningTaskRef** - identifikace dolovací úlohy, která zadala dolovací úlohu.

Kapitola 6

Rozšíření systému o SVM modul

V této kapitole uvedu, jak jsem postupoval při návrhu rozšíření o SVM modul a jak probíhala jeho implementace.

6.1 Návrh koncepce modulu

Při návrhu koncepce bylo cílem vytvořit nový modul a začlenit jej do stávajícího systému. Protože padla volba na klasifikátor, a klasifikační modul už byl vytvořen slečnou Ševčíkovou, bylo zapotřebí jej integrovat právě do něj. Uživatel by dostal v grafickém rozhraní po zvolení dolovací úlohy klasifikace možnost výběru buď stávajícího ID3 algoritmu budující rozhodovací strom nebo SVM klasifikátoru se zadáním vstupních parametrů.

Jednou z prvních věcí bylo dodefinování jazyka DMSL o potřeby nového klasifikátoru. Toto je popsáno v následující sekci 6.2.

Jedním z cílů, které jsme si s vedoucím vytyčili, bylo zahrnout v novém modulu celý proces klasifikace, tedy trénování, testování a klasifikaci neznámých dat. Bohužel klasifikace neznámých dat by znamenala zásah do jádra systému. Pro tuto fázi je totiž vhodné mít druhou datovou matici, která reprezentuje informace o datech, nad kterými probíhá dolovanání, a možnost vybrat více než jednu v uživatelském rozhraní systému není. Úkolem tedy bylo pouze naučit a otestovat konkrétní naučený model. Právě z důvodu konkrétního modelu byla pro otestování zvolena metoda Holdout 3.2.2, která ověřuje pouze jeden model.

Při návrhu jsem rovněž zvažoval možnost použít některé existující kódy slečny Ševčíkové, aby nebyly dvě stejné věci řešeny zbytečně dvakrát. Takovou věcí se ukázal být přístup k databázi. Naopak struktura pro tabulku dat se neukázala být vhodná, protože SVM pracuje primárně s daty s plovoucí řádovou čárkou a bylo by potřeba při každém přístupu pracně přetypovávat z obecných objektů, se kterými pracuje.

6.2 Změny v DMSL

Na základě původního řešení klasifikačního modulu a návrhu koncepce přídavného SVM jsem provedl dodefinování jazyka DMSL o potřeby SVM klasifikátoru v elementech `DataMiningTask` a `Knowledge`. Ukázku dokumentu popisujícího upravenou verzi pro tento model lze najít v příloze A. Popis dat použitých v uvedeném DMSL dokumentu je vysvětlen v kapitole věnující se testování 7. Následuje popis provedených úprav ve zmíněných elementech.

6.2.1 Element DataMiningTask

V této části bylo potřeba najít způsob, jak se vypořádat s problémy, které byly popsány v sekci 5.2.2. Moje řešení spočívá v tom, že se v elementu `DataMiningTask` vytvoří nový element `MineClassification`. Název jsem zvolil podobný, jaký je použitý v modulech asociační a shlukové analýzy. V něm se nachází informace o jméně vybraného modelu, identifikace použité dolovací matice a identifikace klasifikačního atributu z této datové matice. Čili všechny společné parametry pro klasifikaci se nacházejí na jednom místě.

```
<!ELEMENT MineClassification >
<!ATTLIST MineClassification
  dataMiningMatrixRef    CDATA                                    #REQUIRED
  modelName              CDATA                                    #REQUIRED
  classifAttribute       CDATA                                    #REQUIRED >
```

Význam atributů:

- **dataMiningMatrixRef** - identifikace dolovací matice
- **modelName** - jméno modelu (SVM nebo decisionTrees)
- **classifAttribute** - jméno klasifikačního atributu z dataMinngMatrixRef

Pro zadání dolovací úlohy samotného SVM slouží element SVMTask. Jediným jeho elementem je jedna z jeho jádrových funkcí s příslušnými atributy, obdobně jako v [10]. Součástí modulu je i jeho otestování pomocí metody Holdout, proto je přidán atribut o poměru testovacích a trénovacích dat.

```
<!ENTITY % KernelFunctions '(LinearKernel|PolynomialKernel|
                             RadialBasisKernel|SigmoidKernel)' >
<!ELEMENT SVMTask (KernelFunctions)  >
<!ATTLIST SVMTask
    trainTestRatio       %REAL-NUMBER                             #REQUIRED >
```

Následuje seznam jednotlivých elementů jádrových funkcí.

```
<!ELEMENT LinearKernel EMPTY>
<!ELEMENT PolynomialKernel EMPTY>
<!ATTLIST PolynomialKernel
    gamma                %REAL-NUMBER;                            #REQUIRED
    coef0                %REAL-NUMBER;                            #REQUIRED
    degree               %REAL-NUMBER;                            #REQUIRED >
<!ELEMENT RadialBasisKernel EMPTY>
<!ATTLIST RadialBasisKernel
    gamma                %REAL-NUMBER;                            #REQUIRED >
<!ELEMENT SigmoidKernel EMPTY>
<!ATTLIST SigmoidKernel
    gamma                %REAL-NUMBER;                            #REQUIRED
    coef0                %REAL-NUMBER;                            #REQUIRED >
```

Význam atributů:

- **gamma** - v každém jádře má jiný význam
- **coef0** - nulový koeficient
- **degree** - stupeň polynomu v polynomiálním jádře

6.2.2 Element Knowledge

Výsledek dolování SVM modulu bude uložen v elementu `SVMResult`. Ten v sobě obsahuje elementy `SupportVectors` a `SVMCoefficients`.

```
<!ELEMENT SVMResult (SupportVectors), (SVMCoefficients)>
```

Element `SupportVectors` obsahuje výsledné vektory reprezentovány samostatným elementem `SupportVector`.

```
<!ELEMENT SupportVectors (SupportVector*)>
<!ATTLIST SupportVectors
   dimensionCount        %INT-NUMBER                    #REQUIRED
   vectCount             %INT-NUMBER                    #IMPLIED >
```

Význam atributů:

- **dimensionCount** - počet dimenzí vektoru
- **vectCount** - počet support vektorů

```
<!ELEMENT SupportVector EMPTY>
<!ATTLIST SupportVector
   values                CDATA                          #REQUIRED
   class                 %INT-NUMBER                    #REQUIRED >
```

Význam atributů:

- **values** - hodnoty vektoru oddělené čárkou
- **class** - zařazující třída

```
<!ELEMENT SVMCoefficients EMPTY>
<!ATTLIST SVMCoefficients
    coefs                CDATA                          #REQUIRED
    absValue             %REAL-NUMBER                   #REQUIRED
    coefCount            %INT-NUMBER                    #IMPLIED  >
```

Význam atributů:

- **coefs** - hodnoty koeficientů odděleny čárkou
- **absValue** - hodnota absolutního členu
- **coefCount** - počet koeficientů (stejný jako počet support vektorů)

Pro testování jsem přidal do `Knowledge` informace z matice záměn 3.1.

```
<!ELEMENT ClassTesting >
<!ATTLIST
  TP                  %INT-NUMBER                          #REQUIRED
  TN                  %INT-NUMBER                          #REQUIRED
  FN                  %INT-NUMBER                          #REQUIRED
  FP                  %INT-NUMBER                          #REQUIRED >
```

Význam atributů:

- **TP** (true positive), **FN** (false negative), **TN**, **FP** - informace z matice záměn

6.3 Implementace

Při implementaci jsem se snažil naplnit všechny úkoly stanovené při návrhu včetně využití některých kódů existujícího modulu. Řešení přibližně kopírovalo následující postup.

1. **grafické uživatelské rozhraní (GUI)** - upravil jsem třídu `ClassificationGui` z balíku `preprocessing/tasks` . Pomocí grafické knihovny `swing` jsem vytvořil obrazovku pro zadání vstupních parametrů dolovací úlohy a obrazovku vizualizace výsledků SVM. Také bylo potřeba upravit metody pro načítání dat z DMSL dokumentů a ukládání do nich: `LoadDefinitionFromDMSL`, `SaveDefinitionToDMSL`, `LoadResultFromDMSL` a `SaveResultToDMSL`. Ukázky obou uživatelských obrazovek jsou pro ilustraci uvedeny a vysvětleny v sekci 6.4.

2. **xmlclasses** - jedná se o knihovnu ulehčující práci s DMSL dokumenty. Rozšířil jsem ji o třídy pro práci s elementy SVM popsané v 5.2: `XQMineClassification`, `XQSVMTask`, `XQSVMKernel`, `XQSupportVectors`, `XQSupportVector`, `XQClassifTesting`. Také jsem musel upravit třídy `XQKnowledge` a `XQDataMiningTask`. Proto jsem nově upravenou verzi označil číslem 4.

3. **načítání dat** - využil jsem existující kódy pro tvorbu SQL dotazů a připojení k databázi. Ve třídě `DBManager`, která ukládá data do datového objektu v programu, jsem vytvořil novou metodu pro načítání do mnou definované třídy `SVMDataSet`, kde používám pro uložení dat primitivní datový typ `double`. Třída také poskytuje do budoucna možnost uchování dat pomocí řídkých vektorů (sparse vectors).

4. **třída Svm** - je stěžejní částí přídavného modulu. Obsahuje metody pro natrénování binárního klasifikátoru pomocí algoritmu SMO, popsaného v 4.5. Vycházel jsem především z pseudokódu uvedeného v [8] a také z [13]. Rovněž je zde řešeno i testování metodou Holdout a ukládání modelu a výsledků testování do elementu `Knowledge` v DMSL dokumentu.

   Pro výpočty hodnot jádrových funkcí byla vytvořena samostaná třída `SVMKernel`. Při vytváření instance třídy je určen typ jádra, na jehož základě se pak rozhodne, kterou konkrétní funkci má spočítat. Protože se ve funkcích počítají vektorové součiny příkladů ze vstupních dat, musí mít objekt této třídy přístup k dolovaným datům. Proto obsahuje jako atribut referenci na příslušný `SVMDataSet`.

5. **třída Classification** - volá se při provádění klasifikační dolovací úlohy. Zde se rozhoduje, zda se budou volat metody pro tvorbu rozhodovacího stromu nebo se vytvoří nová instance třídy `Svm`. V druhém případě jsou volány metody pro natrénování, otestování a uložení výsledků do DMSL dokumentu, který se potom předá jako výsledek jádru.

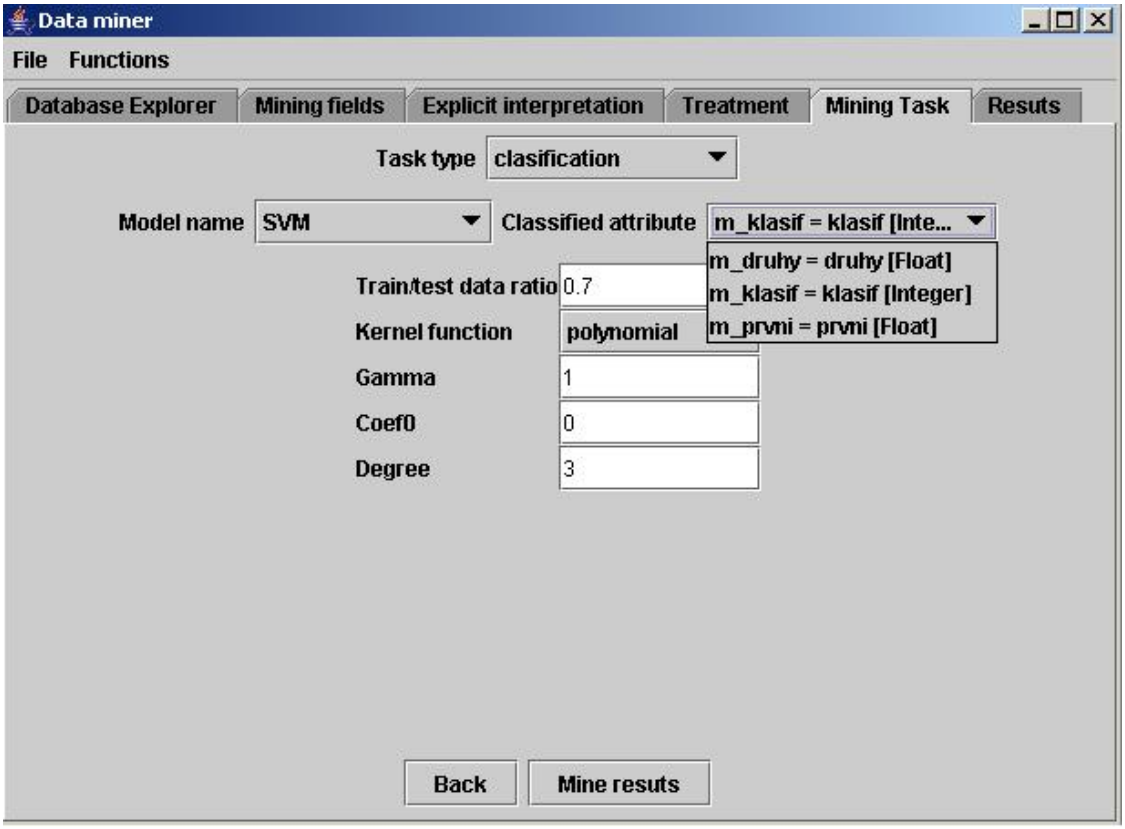
6. **třída ClassifTestMeasures** - jedná se o jednoduchou pomocnou třídu, která slouží k výpočtům testovacích metrik pomocí informací o počtech správně a špatně zařazených příkladů z matice záměn.

7. **třída Utils** - je to další pomocná třída, tentokráte pro matematické výpočty.

6.4 Grafické rozhraní

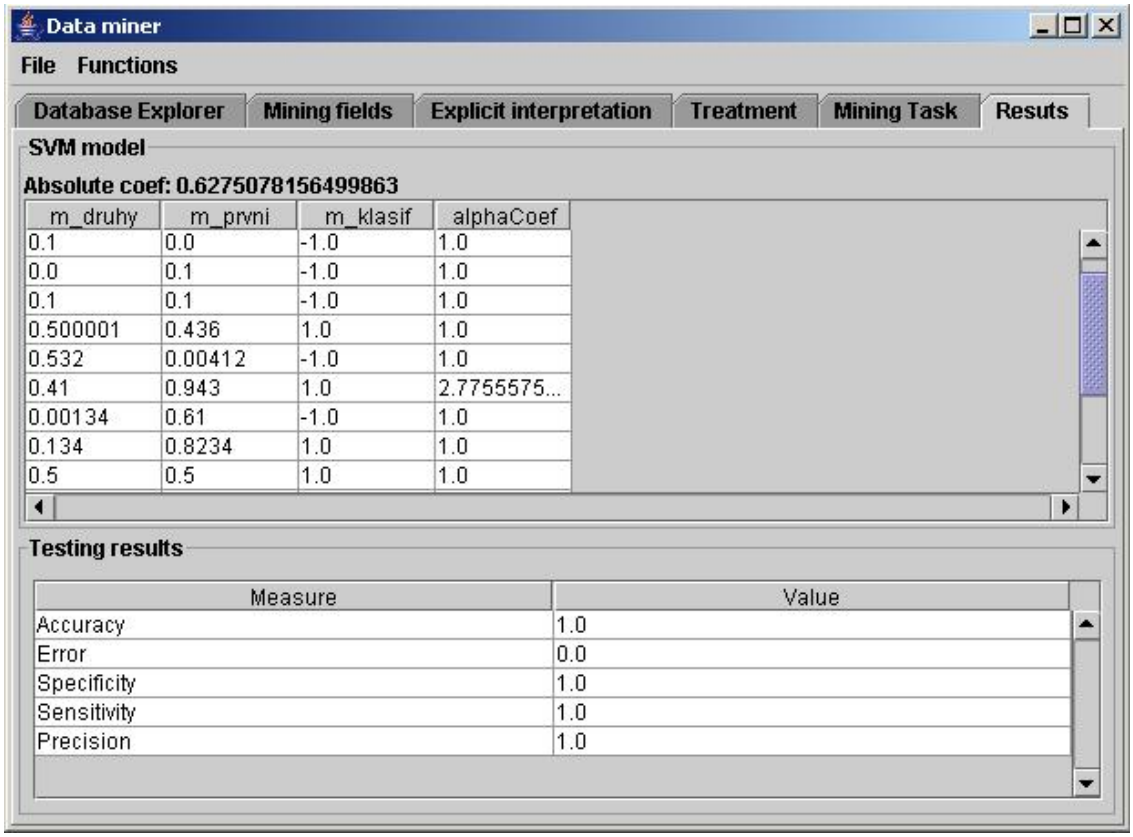
V této části popíšu vytvořené obrazovky grafického uživatelského rozhraní.

Na obrázku 6.1 je zachycena obrazovka zadávání dolovací úlohy (Mining Task). Pro zvolení SVM klasifikátoru je potřeba nejdříve vybrat v horní části ve výběru typu dolovací úlohy (Task type) *classification*. Následně se zvolí jméno modelu (Model name) *SVM*. Vedle něj je nabídka klasifikačních atributů. Zbývá vybrat poměr trénovacích a testovacích dat v rozmezí [0,1] a zvolit jádrovou funkci. Podle typu jádra je uživateli nabídnut seznam vstupních parametrů. Zmáčknutím tlačítka Mine Results se spustí dolovací úloha. Po jejím skončení je aktivována karta výsledků (Results).



Obrázek 6.1: Obrazovka se zadáním dolovací úlohy SVM

Obrazovka výsledků SVM klasifikátoru (obrázek 6.2) je rozdělena do dvou nad sebou umístěných částí. Horní část popisuje natrénovaný model. Nejprve je uvedena hodnota absolutního členu (Absolute coef.) a pod ní se nachází tabulka se support vektory. V jejím záhlaví jsou názvy příslušných atributů těchto vektorů. Na konci je uvedena příslušnost řádku ke třídě a za ní pak hodnota Lagrangova multiplikátoru α (alphaCoef). Ve spodní části nalezneme informace o výsledku druhé fáze klasifikace, tedy testování. Výsledky jsou vyjádřeny v tabulce pomocí metrik představených v sekci 3.2.2: celková správnost (Accuracy), chyba (Error), specificita (Specificity), sensitivita (Sensitivity) a přesnost (Precision).



Obrázek 6.2: Obrazovka s výsledky dolovací úlohy SVM a testování.

Kapitola 7

Testování a zhodnocení výsledků

Pro ověření implementovaného modulu bylo důležité jej otestovat na vhodném vzorku dat. Pro základní otestování, zda modul funguje jsem zvolil jednoduchá 2D data v intervalu $[0; 1]$, která byla lineárně oddělitelná. Při zvoleném poměru testovacích a trénovacích dat 0,7 všechna jádra správně zařadila testovací data s celkovou správností 1. Příloha A 8, jenž obsahuje ukázkový DMSL dokument, ukazuje právě řešení tohoto problému pomocí sigmoidální jádrové funkce. Vstupními dolovacími atributy jsou sloupce `m_prvni` a `m_druhy`, klasifikační atribut je označen jako `m_klasif`. Pokud bych chtěl uvést příklad na některém z následujících řešení, vydalo by to z důvodu velkého množství support vektorů na mnoho stran.

Pro otestování na reálných a složitějších datech jsem zvolil data pojišťovací společnosti - Databáze TIC. Tato data jsou používána při řešení projektů do předmětu ZZN na FIT VUT. Tam studenti využívají k řešení program SAS Enterprise Miner. Poprosil jsem tedy vedoucího o možnost porovnat své výsledky s výsledky některého z projektu, který používal stejná data. Dostal se mi do rukou projekt Tomáše Krejčíře a Lukáše Madrona [7].

Data znázorňují nashromážděné údaje pojišťovací firmy. Celkem 86 atributů popisuje informace o zákaznících. Polovina z nich obsahuje sociodemografické údaje (věk zákazníka, výše jeho platového příjmu, příslušnost k náboženství, ...) a druhá polovina informace o využívání pojišťovacích služeb (počet a výše plateb za určitý typ pojištění).

V projektu bylo řešeno více druhů dolovacích úloh (klasifikace, predikce, asociační analýza). Můj program provádí klasifikaci, a je schopen rozlišovat pouze dvě třídy. Proto jsem hledal v projektu ZZN pouze takovéto testy. Jednalo se o tyto dva:

1. zkoumání, zda zákazník využije možnosti pojistit karavan (atribut 86). V datech se uvádí název atributu jako počet pojistek karavanů. Ten je ale ve všech případech buď 0 nebo 1 a mohl být tedy bez dalších úprav použit.

2. zjištění, jestli má zákazník koupenou pojistku proti požáru. Protože v datech jsou uvedeni i lidé mající více než jednu pojistku, byl vytvořen nový binární atribut, který pouze udává, jestli člověk pojistku má nebo ne.

U obou testů budu srovnávat výsledky celkové správnosti testů z projektu, kde byly řešeny pomocí rozhodovacích stromů, regrese a neuronové sítě. Dále budu srovnávat výsledky všech čtyř jader. Z důvodu dlouhého trvání fáze předzpracování dat na všech zhruba 5000 původních datových záznamech jsem použil pouze prvních 1000. K natrénování SVM modelu bylo vybráno 70 procent dat, k testování se použilo zbylých 30 procent. U jádrových funkcí jsem nastavil nulté keoficienty na 0, parametr gamma na 1 a stupeň polynomiální funkce na 3.

Výsledky prvního z testů dopadly u tří jader stejně jako pro rozhodovací stromy. Poměrně dobré výsledky jsou dány tím, že tři jádra (lineární, radiální a sigmoidální) klasifikovala všechny testované příklady do záporné třídy -1, protože zastoupení kladných záznamů značících zakoupenou pojistku karavanu bylo velmi řídké.

Polynomiální jádro nebylo původně schopno úlohu v rozumném čase vůbec dokončit. Proto jsem zaexperimentoval s nastavením parametrů a při hodnotě gamma = 0.4 dosáhlo podobných výsledků jako ostatní jádra. Jeden negativní příklad ovšem špatně zařadil do pozitivní a to mu vyneslo horší správnost, srovnatelnou s neuronovou sítí.

Metoda	**Správnost**
Regrese	0.930
Rozhodovací stromy	0.940
Neuronová síť	0.937
SVM - linérní	0.940
SVM - polynomiální	0.936
SVM - radialBasis	0.940
SVM - sigmoidální	0.940

Tabulka 7.1: Výsledky prvního testu - pojistka karavanu

Druhý test dopadl z informačního hlediska o poznání lépe. Správnost všech jader se pohybuje okolo 80 procent, čili podobná jako u metod použitých v projektu. Sigmoidální jádro neprve propadlo a dosáhlo přesnosti necelých 60 procent. Proto jsem opět zkusil měnit parametry a při snižující se hodnotě gamma dosahovalo stále lepších výsledků. Při hodnotě 0.1 dosáhlo podobných hodnot jako jádra ostatní.

U polynomiálního jádra jsem opět nastavil parametr gamma na 0.4 ze stejného důvodu jako v předchozí úloze.

Z výsledků lze vypozorovat, že podobně jako u neuronových sítí záleží u SVM klasifikátoru na nastavení vstupních parametrů. Někdy má špatná volba za následek i to, že algoritmus není schopen v rozumném čase vůbec skončit.

Metoda	**Správnost**
Regrese	0.790
Rozhodovací stromy	0.800
Neuronová síť	0.802
SVM - linérní	0.800
SVM - polynomiální	0.770
SVM - radialBasis	0.780
SVM - sigmoidální	0.796

Tabulka 7.2: Výsledky druhého testu - pojistka proti požáru

Kapitola 8

Závěr

Cílem bakalářské práce bylo navrhnout, implementovat a otestovat rozšiřující modul do systému získávání znalostí na FIT. Po dohodě s vedoucím bylo zvoleno rozšíření klasifikačního modulu o SVM klasifikátor. Nejprve jsem se musel seznámit se základy získávání znalostí z databází s bližším zaměřením na klasifikaci a SVM klasifikátor. Poté jsem z algoritmů řešících SVM problém zvolil algoritmus SMO, protože nepotřebuje řešit kvadratický optimalizační problém numericky, ale analyticky.

Pak bylo potřeba navrhnout změnu DMSL jazyka a grafického uživatelského rozhraní o potřeby SVM klasifikátoru. Navržené změny byly společně s algoritmem SMO implementovány a celý modul jsem začlenil do systému. Po implementaci modulu jsem ověřil funkčnost nejdříve na jednoduchých 2D datech a poté na složitějších a rozsáhlejších datech pojišťovací společnosti. Na těchto datech se moje výsledky podobaly těm, jaké získali T. Krejčíř s L. Madronem při řešení projektu do předmětu ZZN pomocí jiných klasifikačních modelů (neuronové sítě, rozhodovací stromy a regrese) [7].

Současné řešení SVM klasifikátoru nemusí být konečné. Nabízí se několik možností jeho rozšíření, například klasifikace do více tříd nebo rozšíření na predikci. U obou možností se dá použít jako základ algoritmus SMO. Po přebudování jádra by bylo možno jednoduše implementovat klasifikaci neznámých dat. Celý klasifikační modul pak může být rozšířen o několik dalších modelů (Bayesova klasikiace, neuronové sítě, jiný algoritmus pro řešení rozhodovacích stromů). Pokračovat lze i řešením jiných částí celého systému. Jednalo by se o úpravy jádra, například pro přístup k jiné databázi než MySQL, či přidání nebo úpravy dalších modulů.

Literatura

[1] BERKA, P.: *Dobývání znalostí z databází*. Praha: Academia, 2003, ISBN 80-200-1062-9, 366 s.

[2] FORGÁČ, M.: *Modul shlukové analýzy systému pro získávání znalostí z databází*. diplomová práce, FIT VUT v Brně, 2006.

[3] HAN, J.: *Classification and prediction*. [online], [cit. 2008-04-20].
URL `<http://www.cs.uiuc.edu/~hanj/bk2/06.ppt>`

[4] HAN, J.; KAMBER, M.: *Data Mining: Concepts and Techniques*. second edition, Morgan Kaufmann Publishers, 2006, ISBN 1-55860-901-6, 770 s.

[5] HSU, C.-W.; CHANG, C.-C.; LIN, C.-J.: *A Practical Guide to Support Vector Classification*. [online], [cit. 2008-04-10].
URL `<http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>`

[6] KOTÁSEK, P.: *DMSL: The Data Mining Specification Language*. Dizertační práce, 2003.
URL `<http://www.fit.vutbr.cz/research/view_pub.php?id=7348>`

[7] KREJČÍŘ, T.; MARDON, L.: *Databáze TIC (pojišťovací společnost) - řešení*. 2007, projekt do předmětu ZZN na FIT VUT v Brně.

[8] PLATT, J. C.: *Fast Training of Support Vector Machines using Sequential Minimal Optimization*. In *Advances in Kernel Methods - Support Vector Learning*, editace B. Schoelkopf; C. Burges; A. Smola, MIT Press, 1998, ISBN 0-262-19416-3, s. 185–208.
URL `<http://research.microsoft.com/~jplatt/smo.html>`

[9] PLATT, J. C.: *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Microsoft Research Technical Report MSR-TR-98-14, 1998.
URL `<http://research.microsoft.com/~jplatt/smoTR.pdf>`

[10] The Data Mining Group: *PMML 3.0 - Support Vector Machines*. [online], [cit. 2008-03-18].
URL `<http://www.dmg.org/v3-1/SupportVectorMachine.html>`

[11] ŠEVČÍKOVÁ, L.: *Modul deskriptivního dolování a klasifikace*. diplomová práce, FIT VUT v Brně, 2003.

[12] Wikipedia: *Support vector machine — Wikipedia, The Free Encyclopedia*. [online], 2008, [cit. 2008-04-20].
URL <http://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=206610626>

[13] WIKSTRÖM, G.: *Data classification using Support Vector Machines*. 2005.
URL <http://www.physto.se/~wikstrom/stat_report.pdf>

[14] ZENDULKA, J.; kol.: *Získávání znalostí z databází - studiní opora*. FIT VUT v Brně, 2006.

Seznam použitých zkratek

- **DB** - Database (databáze)
- **DMSL** - Data mining specification language
- **DTD** - Definition type document
- **FN** - False negative
- **FP** - False positive
- **GUI** - Graphical user interface (grafické uživatelské rozhraní)
- **ICDM** - International Conference on Data Mining
- **JDBC** - Java Database Connectivity
- **KKT** - Karush-Kahn-Tucker
- **MMH** - Maximal marginal hyperplane (největší rozdělující nadrovina)
- **QP** - Quadratic problem (kvadratický problém)
- **TN** - True negative
- **TP** - True positive
- **SMO** - Sequential minimal optimization
- **SVM** - Support vector machines
- **XML** - Extensible Markup Language
- **ZZD** - získávání znalostí z databází

Seznam příloh

Příloha A: ukázka DMSL dokumentu pro SVM
Příloha B: přiložené CD se zdrojovými kódy programu

Příloha A: Ukázka DMSL dokumentu pro SVM

```
<DMSL>
    <FunctionPool name="default"/>
    <DataModel name="default" functionPoolRef="default">
        <DataMatrix name="default">
            <DataField name="druhy">
                <FieldProperties atomicity="scalar" dataType="real"/>
            </DataField>
            <DataField name="klasif">
                <FieldProperties atomicity="scalar" dataType="integer"/>
            </DataField>
            <DataField name="prvni">
                <FieldProperties atomicity="scalar" dataType="real"/>
            </DataField>
        </DataMatrix>
    </DataModel>
    <DataMiningModel dataModelRef="default" name="default">
        <DataMiningMatrix name="default">
            <Query language="SimpleSelect">
                <Database name="local" urlAddress="localhost"
                 userName="root" password=""
                 driver="org.gjt.mm.mysql.Driver">
                    <Column name="druhy" table="test_jednoduche2d"
                     catalogue="bakalarka">
                        <FieldRef name="druhy"/>
                    </Column>
                    <Column name="klasif" table="test_jednoduche2d"
                    catalogue="bakalarka">
                        <FieldRef name="klasif"/>
                    </Column>
                    <Column name="prvni" table="test_jednoduche2d"
                    catalogue="bakalarka">
                        <FieldRef name="prvni"/>
                    </Column>
                </Database>
                <TempDatabase urlAddress="localhost" userName="root"
                 password="" driver="org.gjt.mm.mysql.Driver"
```

```
            catalogue="bakalarka" table="auxTable"/>
        </Query>
        <UseMatrix matrixRef="default"/>
        <DataMiningField name="m_druhy">
            <FieldProperties dataType="real" atomicity="scalar"/>
            <FieldExpression id="0">
                <FieldRef name="druhy"/>
            </FieldExpression>
        </DataMiningField>
        <DataMiningField name="m_klasif">
            <FieldProperties dataType="integer" atomicity="scalar"/>
            <FieldExpression id="0">
                <FieldRef name="klasif"/>
            </FieldExpression>
        </DataMiningField>
        <DataMiningField name="m_prvni">
            <FieldProperties dataType="real" atomicity="scalar"/>
            <FieldExpression id="0">
                <FieldRef name="prvni"/>
            </FieldExpression>
        </DataMiningField>
        <MatrixTreatment name="default"/>
    </DataMiningMatrix>
</DataMiningModel>
<DataMiningTask type="clasification" name="default">
    <MineClassification modelName="SVM"
     dataMiningMatrixRef="default" classAttribute="m_klasif">
        <SVMTask trainTestRatio="0.7">
            <SigmoidKernel gamma="1" coef0="0"/>
        </SVMTask>
    </MineClassification>
</DataMiningTask>
<Knowledge type="clasification">
    <SVMResult>
        <SupportVectors dimensionCount="2" vectCount="12">
            <SupportVector values="0.1,0.0" class="-1"/>
            <SupportVector values="0.0,0.1" class="-1"/>
            <SupportVector values="0.1,0.1" class="-1"/>
            <SupportVector values="0.500001,0.436" class="1"/>
            <SupportVector values="0.532,0.00412" class="-1"/>
            <SupportVector values="0.00134,0.61" class="-1"/>
            <SupportVector values="0.134,0.8234" class="1"/>
            <SupportVector values="0.5,0.5" class="1"/>
            <SupportVector values="0.4,0.5" class="1"/>
            <SupportVector values="0.5657,0.54334" class="1"/>
            <SupportVector values="1.0E-4,0.231" class="-1"/>
            <SupportVector values="0.4,0.4" class="1"/>
        </SupportVectors>
```

```
            <SVMCoefficients coefCount="12" coefs="1.0,1.0,1.0,1.0,
             1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0"
             absValue="1.0463678610736893"/>
        </SVMResult>
        <ClassifTesting TP="6" TN="3" FP="0" FN="0"/>
    </Knowledge>
</DMSL>
```