VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

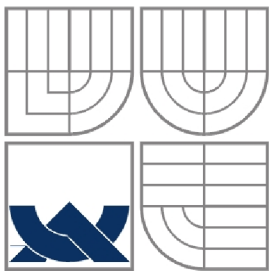
DOLOVACÍ MODULY SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

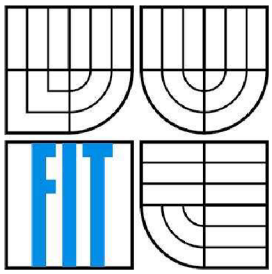
AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ HENKL

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DOLOVACÍ MODULY SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

MINING MODULES OF DATA MINING SYSTEM ON NETBEANS PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. TOMÁŠ HENKL
AUTHOR

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

BRNO 2009

Abstrakt

Diplomová práce se zabývá problematikou získávání znalostí z databází a rozšířením systému pro dolování z dat v prostředí Oracle vyvíjený na VUT FIT. V koncepci jádra tohoto systému je zabudováno rozhraní, umožňující přidávání dolovacích modulů. Hlavním cílem této práce je nastudovat toto rozhraní, naimplementovat a začlenit do aplikace dolovací modul pro klasifikaci metodou rozhodovacího stromu. Práce se také zabývá porovnáním aplikace s podobným komerčním produktem SAS Enterprise Miner.

Abstract

The master's thesis deals with the knowledge discover in databases and with the extending of the data mining systems in the Oracle environment developed at the VUT FIT. The system kernel conception incorporates an interface that enables the adding of data mining modules. The objective of the thesis is to learn this interface and implement and embed the data mining module for decision-tree classification into the application. In addition, the thesis compares the application with similar commercial product SAS Enterprise Miner.

Klíčová slova

Získávání znalostí z databází, data mining, Data Miner, Oracle Data Mining, NetBeans, DMSL, klasifikace, rozhodovací strom.

Keywords

Knowledge discovery in data, data mining, Data Miner, Oracle Data Mining, NetBeans, DMSL, classification, decision tree.

Citace

Tomáš Henkl: Dolovací moduly systému pro dolování z dat na platformě NetBeans, diplomová práce, Brno, FIT VUT v Brně, 2009

Dolovací moduly systému pro dolování z dat na platformě NetBeans

Prohlášení

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

.....
Tomáš Henkl
25.5. 2009

Poděkování

Rád bych poděkoval panu Doc. Ing. Jaroslavu Zendulkovi, CSc. za jeho pomoc a podporu při řešení této diplomové práce.

Obsah

1 Úvod

Na konci 20. století vzniká s rozvojem informačních systémů, nejen počtem, ale i velikostí, velké množství dat, které tyto systémy ukládají v databázích nebo jiných datových úložištích. Počátkem let 90. začíná být problém velkého množství dat výrazný a objevuje se zde snaha získávat zajímavé, potenciálně užitečné a nové informace jinými postupy než pouhým dotazem nad databázovou tabulkou.

Pro tyto účely byly vytvořeny datové sklady(data warehaus), které umožňují data nejenom skladovat, ale také poskytují podporu pro analýzu dat. Model dat je u datového skladu v podobě multidimenzionální datové kostky, která má nad sebou definovány operace. Datové kostky využívá pro svoji činnost OLAP(Online Analytical Procesing) systémy, kde prováděná analýza probíhá většinou interaktivně a spoustu rozhodnutí provádí člověk. Získání komplexních údajů pomocí OLAP nástrojů bývá pracné a časově náročné.

Jako druhý směr pro analýzu velkého množství dat se v 90. letech začal vyvíjet systém pro získávání znalostí z databází. Rozdílnost tohoto přístupu, oproti OLAP operacím, je v tom, že je zde snaha o co nejvíce automatizovaný postup při hledání potenciálně užitečných vztahů, modelů a vzorů. V dnešní době se rozvíjí především metody a nástroje pro získávání znalostí z databází.

Mezi první, kteří využívali získávání znalostí z databází patřili především finanční analytikové. Snažili se zjistit, co lidé nakupuji nejčastěji ve spojení s jinými produkty (tzv. analýza nákupního košíku) nebo se snažili odhadnou, kterému klientovi dát bankovní úvěr a kterému ne na základě zkušeností s jinými klienty (tzv. klasifikace). Dnes je využití tohoto odvětví mnohem dále. Pomocí metod získávání znalostí z databází se odhadují ceny na burzovním trhu, zkoumají se zdravotní rizika podle životního stylu lidí nebo se experimentuje s možností, že by bylo možné pomocí některých metod předpovídat počasí. Odvětví pro použití získávání znalostí z databází je velké množství a jejich počet a úspěšnost se bude v budoucnosti zvyšovat s rostoucím výkonem počítačů.

Tato práce se nesnaží obsáhnout celé téma kolem získávání znalostí z databází, ale má za úkol uvést čtenáře do problematiky. Součástí práce je nastudování metody klasifikace a algoritmů pro tuto metodu, především však metodu rozhodovacího stromu. Dále popisuje postup implementace této metody jako modul pro klasifikaci systému získávání znalostí z databází na platformě NetBeans vyvíjený na fakultě FIT VUT. Tento systém se jmenuje Data Miner. Práce navazuje na diplomové projekty Ing. Krásného [2] a Ing. Madera [3].

Práce je organizována do 7 kapitol. Druhá kapitola popisuje proces získávání znalostí z databází. Mělo by být jasné, že tento proces je relativně složitý a výpočetně náročný, že jeho součástí není jen získání určitých znalostí, ale také úprava, výběr dat a další techniky, které jsou stejně důležité jako samotné dolování dat.

Třetí kapitola je věnována popisu klasifikace. Popisuje, z jakých kroků se skládá a jaká data jsou pro tuto metodu vhodná. Obsahuje také podrobnější popis algoritmu rozhodovacího stromu, kterým se tato práce zabývá a okrajově si všímá některých dalších metod.

Čtvrtá kapitola představuje současný stav systému pro získávání znalostí z databáze vyvíjený na fakultě FIT VUT. Snahou je popsat jednotlivé částí potřebné pro pochopení činnosti systému a pro připojení modulů. Zaměřuje se také na použité technologie při tvorbě modulu a technologie spojené s celým systémem. Částečně popisuje jazyk DMSL, Oracle Data Mining a vývojové prostředí NetBeans.

Kapitola 5 podrobněji popisuje vytvořený modul pro klasifikaci metodou rozhodovacího stromu, podporu Oracle Data Mining pro dolování zvolenou metodou, úpravu jazyka DMSL pro ukládání průběhu dolování, parametrů modulu a vydolované znalosti. Krátce popisuje rozhraní Mining Piece. Věnuje se podrobnějšímu popsání jednotlivých implementovaných tříd a podrobně

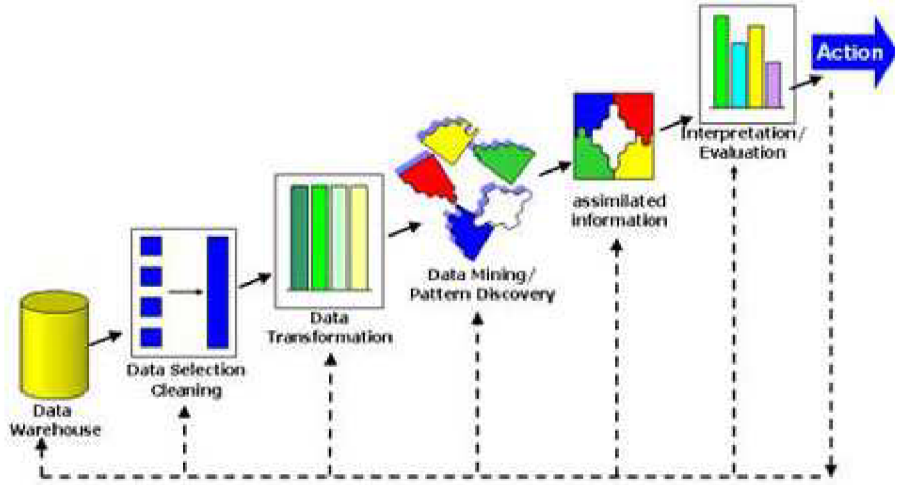
popisuje nově vytvořený balík pro vizualizaci klasifikačních metrik, který by měl sloužit i dalším řešitelům. Šestá kapitola ukazuje grafické rozhraní celé aplikace, vytvoření nové dolovací úlohy, která obsahuje dolovací modul klasifikace rozhodovacím stromem. Popisuje a ukazuje některé komponenty, které jsou potřebné pro vytvoření podmínek pro správnou funkčnost modulu.

Kapitola 7 srovnává námi vytvořený modul s podobným modulem z komerčního systému SAS Enterprise Miner, který se používá pro výuku na FIT VUT. Jedná se o zhodnocení použitelnosti jednotlivých systémů, vizualizaci vydolovaných znalostí a získaných metrik. Osmá kapitola navrhuje možná řešení pro další vývoj aplikace Data Miner. Závěrečná kapitola shrnuje jednotlivé požadavky a hodnotí dosažené výsledky.

2 Získávání znalostí z databází

Můžeme říci, že *získávání znalostí z databází* je extrakce (neboli „dolování“) zajímavých (netriviálních, skrytých, dříve neznámých a potenciálně užitečných) modelů dat a vzorů z velkých objemů dat. Tyto modely a vzory reprezentují znalosti získané z dat. [1]

Kde *netriviální* jsou takové modely, které nejsou dostupné obyčejným dotazem nad databází, ale je nutné použít náročnější postup. *Skryté* znamená, že tyto vzory nejsou na první pohled odhalitelné a musí se poměrně složitě hledat. Aby byly modely užitečné, musí být *dříve neznámé*, známá fakta a skutečnosti nemá smysl hledat. *Potenciálně užitečné* jsou myšleny takové modely a vzory, které nám pomohou se rozhodnout ohledně nějakého postupu, rozhodnutí nebo třeba dokazované hypotézy.



Obr. 2.1: Proces získávání znalostí z dat (převzato z [16])

Je to proces poměrně velice složitý, skládající se z několika kroků (Obr 2.1), jejichž správné provedení je důležité pro dobrý výsledek. Jedná se o procesy:

- **Čištění dat –** proces, při kterém se zpracovávají chybějící a nekonzistentní data a odstraňuje se šum v datech.
- **Integrace dat –** data, která často pocházejí z více zdrojů, je potřeba sjednotit. Bývá často zároveň s čištěním, protože nekonzistence je většinou způsobena právě daty z více zdrojů.
- **Výběr dat –** né všechna data jsou pro danou úlohu potřebná. V této fázi vybereme ty sloupce, dimenze z relační databáze nebo datového skladu, které jsou pro řešení úlohy podstatné.
- **Transformace dat –** převod dat do podoby vhodné pro dolování, například normalizace vstupu nebo odstranění extrémních hodnot.
- **Dolování dat –** tento náročný krok je jádrem celého procesu, aplikuje metodu a algoritmus na data připravená v předchozích krocích a vytváří vzory nebo modely dat, které jsou podkladem pro další zpracování.
- **Hodnocení modelů a vzorů –** snahou v tomto kroku je najít ty skutečně zajímavé vzory, které budou vhodné pro prezentaci.

- **Pochopení a prezentace znalostí –** velice důležitým krokem je interpretace získaných znalostí způsobem, který je vhodný pro širokou škálu lidí i bez technického vzdělání, většinou vhodnou vizualizační technikou (grafy, tabulky, …).

První čtyři kroky se nazývají předzpracování dat. Data, která jsou většinou uložena v klasických relačních databázích nebo v datových skladech, nejsou často v takovém stavu, abychom je mohli ihned použít pro dolování dolovacími metodami. Proto je nezbytné taková data upravovat. Mezi nejčastější nedostatky patří chybějící údaje, nekonzistentnost dat, data jsou příliš odlehlá od ostatních nebo nejsou správná. Pokud bychom dolovacímu algoritmu předali neupravená surová data, jistě bychom dostali špatné výsledky, jejichž interpretace by mohla být velice zavádějící.

Na obrázku 2.1 je systematicky znázorněn proces získávání znalostí z databází. Je z něj patrná již zmiňovaná závislost jednotlivých kroků na dobrém výsledku celého procesu.

2.1 Typy dolovacích metod

Existují různé druhy dolovaných informací a znalostí, a proto je nutné mít také několik druhů dolovacích metod, kde každá metoda může být řešena jiným algoritmem. Typy dolovacích metod lze rozdělit na dvě základní skupiny:

- **Prediktivní** – tyto úlohy se na základě vstupních dat snaží předpovědět budoucí hodnoty. Jedná se deduktivní chování, kdy se snažíme například zjistit, zda je nový zákazník schopen splácet úvěr. Klasifikace je jednou z prediktivních úloh.
- **Deskriptivní** – popisné metody, které se snaží charakterizovat vlastnosti dat. Jednou z deskriptivních metod je asociační analýza.

2.1.1 Klasifikace a predikce

Jedná se o prediktivní typy dolovacích metod. Cílem *klasifikace* je nalézt model dat, kterým bychom rozlišili a popsali třídy dat. Tento model poté použijeme na data, u kterých neznáme jejich zařazení, pro jejich přiřazení do správné třídy. Postupujeme v několika krocích:

- **Učení** – na trénovacích datech, u kterých známe zařazení, se vytvoří klasifikační model
- **Testování** – klasifikační model se musí vyhodnotit. Určuje se četnost případů, ve kterých model klasifikoval třídu správně. Děj probíhá se na známých datech. Pokud testování dopadne úspěšně, je model připraven k použití, pokud ne, musí se upravit.
- **Aplikace** – aplikace modelu na data, jejichž třídu chceme určit.

Klasifikace se používá pro určení diskrétní (kategorické) hodnoty tříd (teplé, horké, studené,…). Tímto tématem se podrobněji budeme zabývat ve 4. kapitole. Pro hodnoty spojitých atributů používáme *predikci*. Nejčastěji se jedná o numerické hodnoty (např. teplotu). K predikci se často využívá metoda support vector machine (SVM) nebo regrese.

2.1.2 Popis konceptu/tříd

Data mohou být spojována s určitou třídou nebo konceptem, poté je možné tyto třídy a koncepty popsat určitým jednoduchým a přesným způsobem. Popis lze vytvářet dvojím způsobem:

- **Charakterizace dat** – je souhrnný popis vlastností třídy. Výsledkem bývá charakteristika např. určitého typu zákazníků nebo určitého druhu zboží. Hledá hodnoty atributů, které jsou podobné s vlastnostmi této třídy.
- **Diskriminace dat** – nepopisuje třídu obecně, ale snaží se ji porovnávat s jednou nebo více jinými třídami. Hledá hodnoty atributů, které se co nejvíce odlišují.

2.1.3 Frekventované vzory

Jsou to vzory, které se v datech vyskytují často. Tyto úlohy ukazují zajímavé spojitosti mezi atributy, které se používají např. pro analýzu nákupního košíku. Jedná se o vyhledávání asociací a korelací. Asociační analýza je procesem, při které hledáme asociační pravidla v takovém tvaru, že nějaký atribut podmiňuje jiný atribut. Např. u analýzy nákupního košíku se jedná o pravidla, že pokud si zákazník koupí *rohlíky*, koupí si i *sýr*. Takové pravidlo by vypadalo:

kupuje(X, rohliky) => kupuje(X, sýr) [podpora = 2%, spolehlivost = 43%]

Důležitými údaji jsou podpora a spolehlivost, které určují významnost zastoupení frekventovaného vzoru v datech, která analyzujeme. *Podpora* značí, v kolika procentech analyzovaných nákupů byly společně tyto položky zakoupeny. V našem případě si zákazníci koupili zároveň rohlíky a sýr ve 2% nákupů. *Spolehlivost* vyjadřuje počet zastoupení položek na pravé straně pravidla vzhledem k výskytu položek na levé straně pravidla. V našem případě si ve 43% zákazník koupí rohlíky a hned taky sýr.

Pravidla nemusí mít na levé straně pouze jednu jedinou podmínku (tzv. jednodimenzionální pravidlo), ale může jich tam být i několik (tzv. multidimenzionální pravidlo).

2.1.4 Shluková analýza

Shlukování rozděluje jednotlivé objekty do tříd na základě podobnosti. Hledá takové objekty, které jsou si nejvíce podobné a spolčuje je do tříd. Snaží se zároveň, aby se jednotlivé třídy od sebe co nejvíce lišily.

2.1.5 Analýza odlehlých objektů

Použití analýzy odlehlých objektů nespočívá v hledání vzorů, které se vyskytují velice často, ale v hledání vzorů, které jsou od ostatních něčím odlišné, tzv. odlehlé objekty. Metoda se využívá např. při hledání podezřelých bankovních transakcí nebo při odhalování neobvyklého chování uživatelů, abychom předcházeli nebezpečným situacím.

Úprava dat pro tuto metodu nesmí provádět úpravy na vyhlazení odlehlých hodnot nebo použit normalizaci, protože by se tím odstranila nejdůležitější část dat a výsledek takto chybně upravených dat by neměl žádnou vypovídající hodnotu.

2.1.6 Evoluční analýza

Sleduje chování jednotlivých atributů v čase. Zaměřuje se na intervaly, které jsou mezi jednotlivými opakováními nebo rychlost, s jakou se daný atribut mění. Využití evoluční analýzy je především při odhadu ceny akcií na burze.

3 Klasifikace

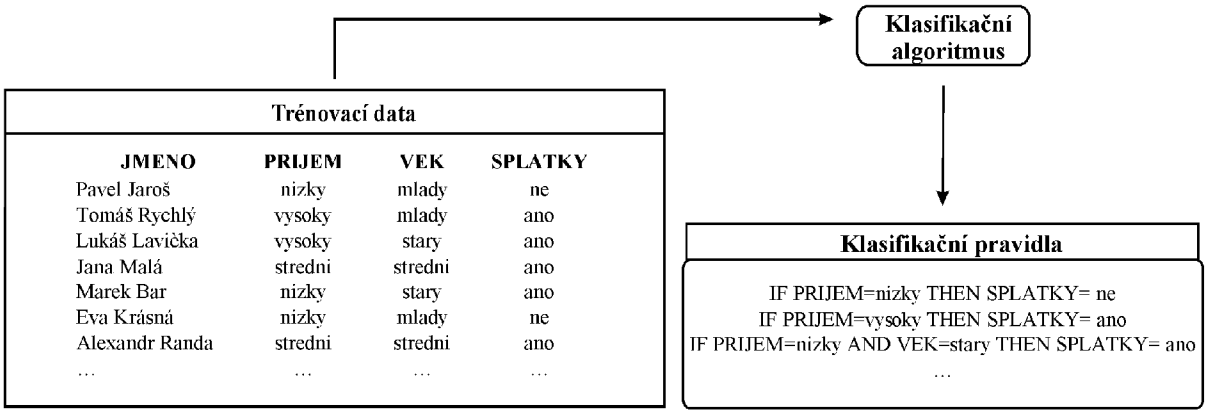
Jedná se o metodu, která se snaží přiřadit objekt do třídy na základě vlastností tohoto objektu. Pro klasifikaci je nutné, aby cílový atribut, který chceme klasifikovat, nabýval diskrétních hodnot. Nejjednodušší samozřejmě je, aby tato hodnota byla binární (např. malé, velké nebo ano, ne), ale není to podmínkou. Mezi nejčastější metody, které se pro klasifikaci používají, patří rozhodovací strom, neuronová síť nebo bayesovské sítě.

Jako příklad by mohl být obchod, který se snaží rozdělit zákazníky do dvou skupin. Jedna skupina bude obsahovat zákazníky, kteří budou bezproblémoví, a firma jim poskytne možnost nakupovat na splátky. Druhá skupina bude obsahovat takové zákazníky, kteří se pro splátkový prodej budou jevit jako rizikoví, a proto jim obchod nebude poskytovat možnost nakupovat na splátky. Tedy cílový atribut se jmenuje *SPLATKY* a může nabývat hodnot *ano* a *ne*.

Dále se v této kapitole budeme zabývat problematikou klasifikace. Popíšeme její hlavní fáze, které budeme vysvětlovat. Pro lepší názornost, použijeme již zmíněný příklad prodejny.

3.1 Fáze klasifikace

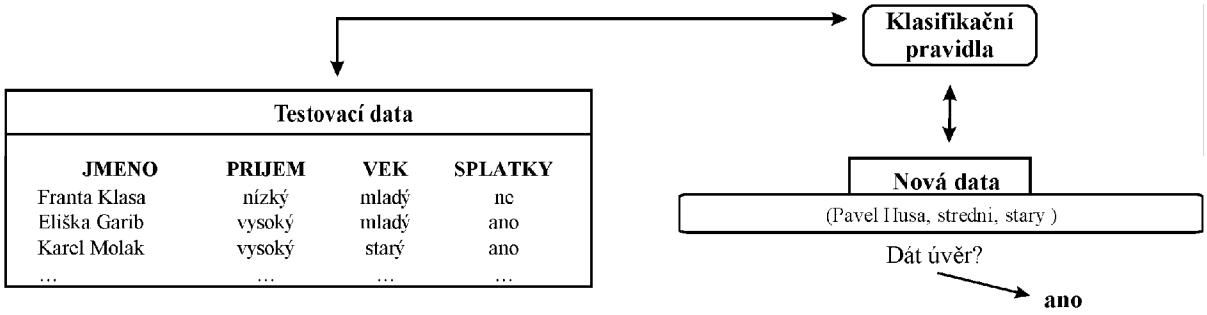
Klasifikace je rozdělena do fáze *učení (trénování)*, fáze *testování* a fáze *aplikace*. Během první fáze jsou z databáze vybrány vzorky, na kterých se bude trénovací mechanizmus učit (tzv.trénovací množina). U těchto dat víme, do které třídy patří, a jsou tedy vhodným vstupem pro klasifikační mechanizmus. V případě klasifikace pomocí rozhodovacího stromu, je poté z této trénovací množiny potřeba zjistit klasifikační pravidla, pomocí kterých lze s jistou pravděpodobností klasifikovat objekt do tříd (Obr 3.1).



Obr. 3.1: Tvorba klasifikačních pravidel (podle [2])

Druhým krokem je fáze testování (Obr 3.2), kde se vyhodnocuje, s jakou přesností klasifikační mechanismus funguje. Vybere se množina objektů, u kterých je známá třída, které jsou ale jiné než vzorky u trénovací množiny. Prvky testovací množiny jsou naučeným klasifikačním mechanizmem přiřazovány do jednotlivých tříd. Vzhledem ke znalosti, kam který prvek patří, se zjistí, jak dobře je klasifikační algoritmus naučený.

Z výsledků testování se určí, jestli je klasifikační mechanizmus (klasifikátor) dostatečně dobře naučen a připraven pro použití pro klasifikaci nebo zda se má nějakým způsobem upravit .



Obr. 3.2: Klasifikace a aplikace (podle [2]).

3.2 Příprava dat pro klasifikaci

Příprava dat před samotným procesem klasifikace může pomoci zlepšit afektivnost, přesnost a stabilitu. Jedná se především o :

- **Čištění dat** – tento krok by měl odstranit nesprávná a chybějící data. Většina klasifikačních algoritmů zahrnuje i metody pro zacházení se zašuměnými a chybějícími daty, ale tento krok může pomoci snížit nepřesnost v průběhu učení.
- **Analýza relevance** – v datech může být velice mnoho redundantních údajů. Pro nalezení atributu, který je závislý na jiném, může sloužit korelační analýza. Například pokud je silná korelace mezi dvěma atributy, je pravděpodobné, že můžeme jeden z nich odstranit a tím zmenšíme množství dat, ale nesnížíme tím jejich informační hodnotu. Další nerelevantní atributy jsou atributy zcela zbytečné (v našem příkladě je to např. jméno).
- **Transformace** – data mohou být transformována pomocí normalizace. Normalizace převede většinou spojité atributy do intervalu od -1 do 1 nebo od 0 do 1. Další možností transformace dat je diskretizace. Tento proces převede spojitý atribut do několika různých intervalů a tím vytvoří diskrétní atribut. Diskretizace atributů je nutná pro metodu rozhodovacího stromu.

Podrobný popis čištění, analýzy relevance, redukce a transformace lze nalézt v [2].

3.3 Algoritmy klasifikace

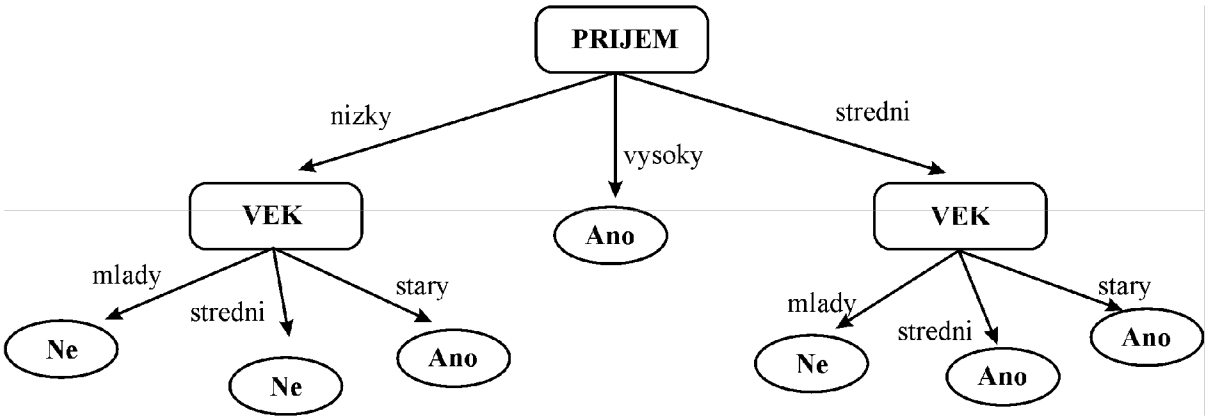
Existuje několik různých metod pro klasifikaci. Každá má jiné parametry a každá je vhodná pro jiný typ dat a znalostí, které chceme získat. Pokud na některých datech přináší dobré výsledky klasifikace pomocí neuronových sítí, tak na jiných datech může být nejlepší metoda rozhodovací strom. Proto je dobré vědět, jaké mají metody vlastnosti. Existuje několik metrik, které rozhodují o kvalitě metody:

- **Přesnost** – udává podíl nových objektů, které byly přiřazeny do správné třídy. Tato metrika je udávána v procentech úspěšných pokusů.
- **Rychlost** – náročnost výpočtu a používání klasifikačních pravidel.
- **Stabilita** – schopnost vytvořit správný model i pro veliké množství vstupních dat.
- **Robustnost** – odolnost proti šumu a chybějícím hodnotám.
- **Interpretovatelnost** – jednoduchost pochopení výstupních dat, některé algoritmy mají na výstupu jednoduchá pravidla, jiná složitější objekty.

Budeme se především zabývat metodou rozhodovacího stromu, která je cílem této práce. Ostatní metody zmíníme jen okrajově.

3.3.1 Rozhodovací strom

Rozhodovací strom je diagram se stromovou strukturou, obecně n-nární, kde nejvrchnější uzel reprezentuje kořen stromu. Vnitřní uzly reprezentují testování atributu, hrany reprezentují výsledek testu a listové uzly označují třídu, do které bylo klasifikováno (Obr. 3.3).



Obr. 3.3: Rozhodovací strom z příkladu

Klíčovým bodem tvorby rozhodovacího stromu je výběr atributu „s nevyšší rozhodovací schopností“, který tvoří na začátku kořenový uzel stromu. Pokud se nám podaří takový atribut nalézt, je proces určení správné třídy pro daný objekt klasifikace mnohem přesnější a výpočetně méně náročný.

3.3.1.1 Výběr atributu s největší rozhodovací schopností

Pro výběr vhodného atributu s největší rozhodovací schopností je potřeba uvážit některé vlastnosti atributů, které mají vliv na množství informace, které atribut nese. Pokud některý atribut obsahuje hodnotu s pravděpodobností 1, nenese pro nás žádnou informaci. Je tedy zřejmé, že hodnota atributu, která má méně pravděpodobný výskyt, nese větší množství informace.

Informační zisk (Information gain)

Tato metoda, použita v **algoritmu ID3**, hledá atribut *S* s největší hodnotou *Gain(A)*. Hodnotu Gain(A) vypočteme:

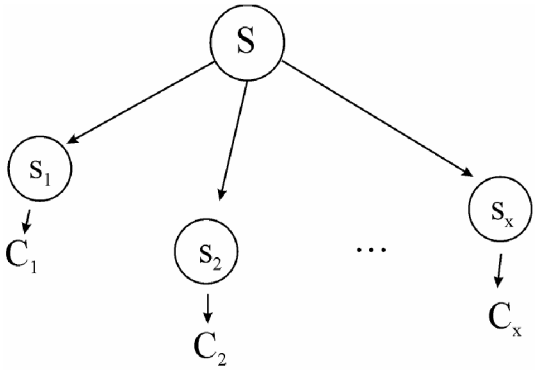
$$Gain(A) = Info(S) - Info_A(S) \qquad (3.1)$$

Prvky množiny kde *S*, *S* je množina všech vzorků trénovacích dat, které jsou klasifikovány do tříd C_1, C_2, až C_x, kde *x* značí počet tříd množiny *S*. Symboly $|s_i|$ označují počet prvků množiny *S*, které jsou přiřazeny do třídy C_i . Hodnotu *Info(S)* vypočteme:

$$Info(S) = -\sum_{i=1}^{x} p_i \log_2(p_i) \qquad (3.2)$$

$$p_i = \frac{|s_i|}{|S|} \tag{3.3}$$

Kde hodnota p_i značí pravděpodobnost, že náhodný vzorek z *S* bude klasifikován do třídy C_i. $|S|$ je počet prvků množiny *S* a *i = 1 … x*. Na obrázku 3.4 je ilustrován výpočet *Info(S)*. Kde s_i hodnota z množiny S, podle které bylo provedeno klasifikování do třídy C_i.



Obr. 3.4: Ilustrace výpočtu *Info(S)*

Pro *Info(S)*, tedy pro označení průměrné hodnoty informace množiny S, se někdy používá pojem **entropie** množiny S (H(S)). Entropie je maximální pro rovnoměrné rozložení p_i, tedy pro $p_1 = p_2 = \ldots = p_x$, což se dá vyjádřit jako:

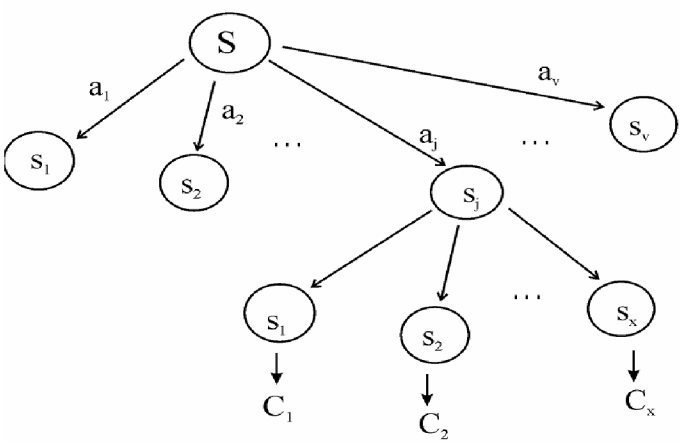
$$H(S) = \log_2 x \tag{3.4}$$

Kde *x* je počet tříd množiny S. Naopak minimální entropie je, pokud nějaké $p_i = 1$, kde $H(S) = 0$.

Nyní předpokládejme, že budeme chtít rozdělit množinu S pomocí atributu A, kde A obsahuje *v* různých hodnot (a_1, a_2, … ,a_v). Atribut A může být použit pro rozdělení množiny S do *v* částí (s_1, s_2, … ,s_v), kde s_j obsahuje n-tice z S, které odpovídají výsledku dělení podle a_j .(Tento postup je znázorněn na obr. 3.5)

$$Info_A(S) = \sum_{j=0}^{v} \frac{|s_j|}{|S|} \cdot Info(s_j) \tag{3.5}$$

Kde $\frac{|s_j|}{|S|}$ znamená váhu j-té části a *v* je počet tříd atributu A. $Info_A(S)$ je tedy očekávaná informace pro klasifikaci, kterou získáme z množiny S, pokud zvolíme nejlepším atributem A.



Obr. 3.5: Ilustrace výpočtu $Info_A(S)$

Gain ratio

Jednou z nevýhod při výpočtu informačního zisku je, že funkce $Info_A(S)$ je závislá na počtu hodnot atributu A. Lze říci, že čím většího počtu hodnot nabývá A, tím menší hodnoty nabývá $Info_A(S)$, pak v extrémním případě může $Info_A(S) = 0$ pro unikátní atribut A. Tyto atributy jsou vybírány před ostatními, protože jejich hodnota Gain(A) je největší, ale skutečná rozhodovací schopnost je malá.

Snahou o vylepšení je výpočet GainRatio(A), který využívá **algoritmus C4.5**. Tato hodnota je upravenou hodnotou Gain(A) a tím kompenzuje možnou chybu.

$$SpliInfo_A(S) = -\sum_{j=1}^{y} \frac{s_j}{|S|} \log_2 \left(\frac{s_j}{|S|} \right) \tag{3.6}$$

$SplitIfo_A(S)$ reprezentuje potenciální informaci tvořenou dělením trénovacích dat S do y částí, kde y je počet tříd atributu A.

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(S)} \tag{3.7}$$

Toto nahrazení má za následek normalizaci hodnoty Gain(A). Jako výsledek se bere největší hodnota GainRatio(A) pro atribut A.

Gini index

Použijeme-li podobnou notaci jako u předchozího výkladu, pak:

$$Gini(S) = 1 - \sum_{i=1}^{y} p_i^2 \tag{3.8}$$

Výhodou je, že tato metoda vytváří rozhodovací strom, který je v binární podobě (tedy má každý nelistový uzel dva následníky). Při vytváření stromu se nejprve z atributu A vytvoří všechny podmnožiny s_j, poté se zeptáme, které prvky atributu A patří do s_j, která byla vybrána jako rozhodovací.

$$Gini_A(S) = \sum_{j=0}^{y} \frac{s_j}{S} Gini(s_j) \qquad (3.9)$$

Pro dělení stromu je vybrán takový atribut A s množinou s_j, která má největší hodnotu $\Delta Gini(A)$.

$$\Delta Gini(A) = Gini(S) - Gini_A(S) \qquad (3.10)$$

3.3.1.2 Ořezání stromu

Vlivem chyb v datech vznikají při vytváření stromu větve, které jsou nežádoucí. Strom s větším množstvím větví nám většinou neposkytuje větší přesnost klasifikace, ale dělá pouze složitější výpočet. Pro odstranění nežádoucích větví se využívá metod:

- **Prepruning** – jedná se o metodu eliminace větví, která při vytváření rozhodovacího stromu nevytváří nežádoucí větve. Atribut se dále netestuje, ukončí se listem, který se klasifikuje do třídy, která má zde nejvíce zástupců. Tato metoda je výpočetně méně náročná.
- **Postpruning** – je metoda, kde se nejdříve vygeneruje celý rozhodovací strom a až potom, se odstraní nežádoucí větve. Metoda je oproti prepruningu výpočetně náročnější, ale přesnější.

Většinou je výběr metody řešen tak, že se použije kombinace obou dvou metod, protože obě metody jsou protichůdné a každá má svoje výhody i nevýhody.

3.3.2 Bayesovská klasifikace

Jedná se o metodu klasifikace založenou na statistických výpočtech. Pro každý vzorek se určuje do jakých tříd s jakou pravděpodobností patří. Poté se vzorek klasifikuje do třídy s největší vypočtenou pravděpodobností. Při klasifikaci se využívá podmíněné pravděpodobnosti a Bayesův vzorec.

- **Podmíněná pravděpodobnost** – pokud máme dva různé jevy X a Y, pak P(X|Y) je podmíněná pravděpodobnost X za podmínky výskytu jevu Y. Je to hodnota, která nám říká, jaká je pravděpodobnost že nastane X, pokud víme, že nastal jev Y.

$$P(X \mid Y) = \frac{P(X \cap Y)}{P(Y)} \qquad (3.11)$$

  Kde $P(X \cap Y)$ je pravděpodobnost, že nastane X a současně Y, P(Y) je pravděpodobnost jevu Y.
- **Bayesův vzorec** – vzorec je odvozen z podmíněné pravděpodobnosti a je důležitý pro přiřazování do tříd.

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)} \qquad (3.12)$$

Mezi výhody Bayesovské klasifikace patří jednoduchá implementace a dobré výsledky v řadě případů, pokud jsou atributy na sobě nezávislé. Potřeba nezávislých atributů je velkou nevýhodou, protože ve skutečnosti bývají atributy na sobě závislé.

3.3.3 Neuronové sítě

Princip neuronových sítí je velice podobný neuronovým sítím ve skutečné nervové soustavě. Síť je tvořena neurony, které mají několik vstupů a výstupů. Jedná se o síť neuronů složenou s několika vrstev. První vrstva je vrstva vstupní, další jsou vrstvy skryté, kterých může být i několik, ale minimálně jedna a poslední vrstva je vrstva výstupní.

Ke klasifikaci se často používá neuronová síť Backpropagation, ve které při učení hledáme vhodné nastavení vstupů tak, že trénovací hodnota projde sítí a ta určí výslednou třídu. Poté porovnáme výsledek trénování a skutečné třídy a chybu klasifikace šíříme sítí zpět od výstupní vrstvy k vstupní vrstvě a přenastavujeme parametry jednotlivých neuronů.

Mezi výhody neuronových sítí patří dobré výsledky. Nevýhodou je velká časová náročnost trénování.

3.3.4 Support Vector Machines (SVM)

Je to matematický model pro klasifikaci numerických atributů. V základní verzi klasifikuje pouze do dvou tříd. Metoda hledá nadrovinu, která rozdělí prostor dat na dva podprostory. Rozdělující nadrovina by měla mít takové vlastnosti, aby vzdálenost různých tříd byla co největší. Vzdálenost dvou tříd se nazývá *margin*. Námi hledaná největší oddělující rovina se nazývá maximal marginal hyperplane (MMH). Pokud zavedeme rozšíření, je SVM schopno klasifikovat i s lineárně neoddělitelnými daty.

SVM je metoda klasifikace, která se v poslední době stává velice častou. Mezi výhody lze zařadit přesnost výsledků, dobrá interpretovatelnost a není náchylná k přeučení.

3.3.5 Další metody klasifikace

Existuje ještě několik dalších metod a algoritmů klasifikace. Výše uvedené jsou nejčastěji zastoupené v dolovacích modulech. Mezi další patří klasifikace naložené na:

- K-nejbližším sousedovi
- Fuzzy množinách
- Genetické algoritmech
- Rozhodovacích pravidlech

4 Aplikace vyvíjená na FIT

Na FIT VUT v Brně se vyvíjí systém dolování dat, který umožňuje přípravu a definici dat. Dolování dat je zde prováděno připojitelnými moduly, které dolují a prezentují data. Aplikace je tvořena architekturou klient-server, kde server tvoří databázový server Oracle (Data Mining Engine - DME) s platformou Oracle Data Mining (ODM). Aplikace je tvořena v jazyce Java v prostředí NetBeans, které je použito jako modulární základ. Server zajišťuje samotné dolování dat, klient zajišťuje definici dat, přípravu a jejich transformaci. Pro ukládání metadat a procesu dolování je zde použit jazyk DMSL, který je popsán v následující kapitole 4.1.3

Aplikaci lze rozdělit na dvě hlavní části - jádro a moduly. Jádro je v podstatě tvořeno samotným jádrem aplikace, které vytvořil Ing. Doležal, a grafickou nadstavbou, kterou vytvořil Ing. Gálet [9]. Obě dvě tyto komponenty byly upraveny a spojeny do jednoho celku Ing. Krásným [3]. Prvním modulem této aplikace, který vytvořil Ing. Mader, je modul dolování dat metodou frekventovaných vzorů a asociační analýza. Je popsán v jeho diplomové práci [8].

4.1 Použité technologie

4.1.1 Oracle Data Mining

Firma Oracle má jako součást databázového systému i podporu pro získávání znalostí z databází, která se nazývá Oracle Data Mining (ODM). Tato podpora je dostupná u databázových systémů od verze 9iR2, která je součástí databázového systému firmy Oracle od roku 2002. V současné době je poslední verze 11gR1 z roku 2007. Obsahuje mnoho algoritmů pro klasifikaci, regresi, predikci, asociační analýzu, detekci odlehlých hodnot, asociační analýzu, shlukování a jiné specializované techniky. Tyto algoritmy poskytují možnosti pro tvorbu, správu a zlepšení úspěšnosti dolovacího procesu. Součástí řešení firmy Oracle jsou i funkce předzpracování a úprav dat před dolováním (diskretizace, normalizace, …). ODM implementuje všechny varianty dolovacích procesů do jádra relačního databázového systému, čímž odstraňuje množství operací spojených s přenášením dat do specializovaných dolovacích serverů. Všechna data potřebná k dolování, jakož i dolovací model a výsledky, jsou uloženy v relačních tabulkách. Uživatel definuje data, nad kterými chce aplikovat dolovací mechanismus, nastaví parametry modelu, které se uloží na server, a spustí dolovací proces [10].

ODM podporuje PL/SQL, které podporuje vytvoření, mazání, použití, testování, export a import modelů pomocí nativního balíku DBMS_DATA_MINING. ODM také podporuje Java API. První implementace knihovny pro Java API na Oracle 9iR2 byla omezena licencí pouze pro Oracle. Od verze Oracle 10.2 je pro komunikaci a vývoj dolovacích aplikací implementován Java API standard Java Data Mining (JDM) v. 1.0 jako JSR 73. V roce 2006 byl specifikován nový standard JDM v. 2.0 pod označením JSR 247. JDM definuje rozhraní, které by měl dolovací systém poskytovat, aby byla zajištěna nezávislost aplikace na konkrétním nástroji pro data mining. K rozhraní Java Data Mining pro ODM je na stránkách firmy Oracle kvalitní dokumentace v podobě *JavaDoc* [14] a k dispozici jsou také vzorové příklady [17].

4.1.1.1 Funkce ODM

Oracle Data Mining poskytuje řadu funkcí pro podporu získávání znalostí. Tyto funkce jsou obsaženy ve verzi 10gR2 i 11gR1. Budeme se zde zabývat pouze těmi, které jsou užitečné při klasifikaci.

Příprava a transformace

- Ořezání, vyhlazení, diskretizace ,normalizace

Podporované metody klasifikace

Metody pro klasifikaci, které podporuje ODM a jejich srovnání.

- Rozhodovací strom (RS)
- Bayesovská klasifikace (NB)
- Neuronové sítě (NS)
- Support vector machine (SVM)

	NB	NS	SVM	RS
Rychlost	velmi rychlé	rychlé	rychlé s aktivním učením	rychlé
Přesnost	dobrá	dobrá	výborná	dobrá
Interpretovatelnost	žádná pravidla (černá skříňka)	pravidla pouze v jednoduchých případech	žádná pravidla (černá skříňka)	pravidla
Interpretace chybějících hodnot	chybějící hodnota	chybějící hodnota	řídká data	chybějící hodnota

Tab. 4.1: Porovnání klasifikačních metod v ODM (podle [7}).

Metriky pro vyhodnocení modelu

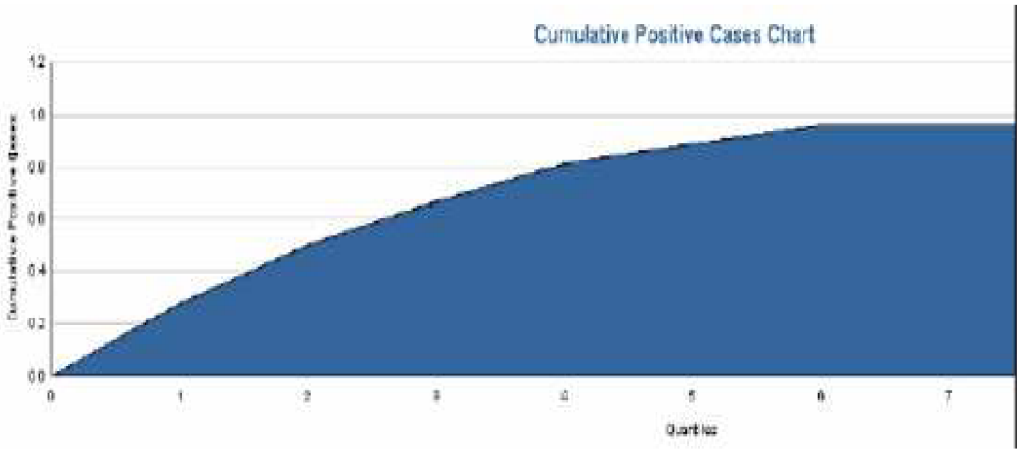
Pro všechny metody klasifikace, které jsou podporovány ODM, jsou k dispozici metriky pro vyhodnocování přesnosti těchto metod. Podrobně jsou popsány v [12].

- **Tabulka přesnosti** (**Confusion Matrix**) – je tabulka, která udává v kolika případech byla klasifikační metoda úspěšná a v kolika neúspěšná, vzhledem k testovaným datům (Tab. 4.1). Je to matice *n* x *n*, kde *n* je počet tříd, do kterých je cílový atribut klasifikován. Z hodnot v matici je možné vypočítat chybovost modelu a jeho přesnost. Kde přesnost je poměr správě klasifikovaných ku všem případům a chybovost je počet chybně klasifikovaných ku všem případům.

		předpovídaná třída	
		SPLATKY = ano	SPLATKY = ne
skutečná třída	SPLATKY = ano	130	10
	SPLATKY = ne	17	267

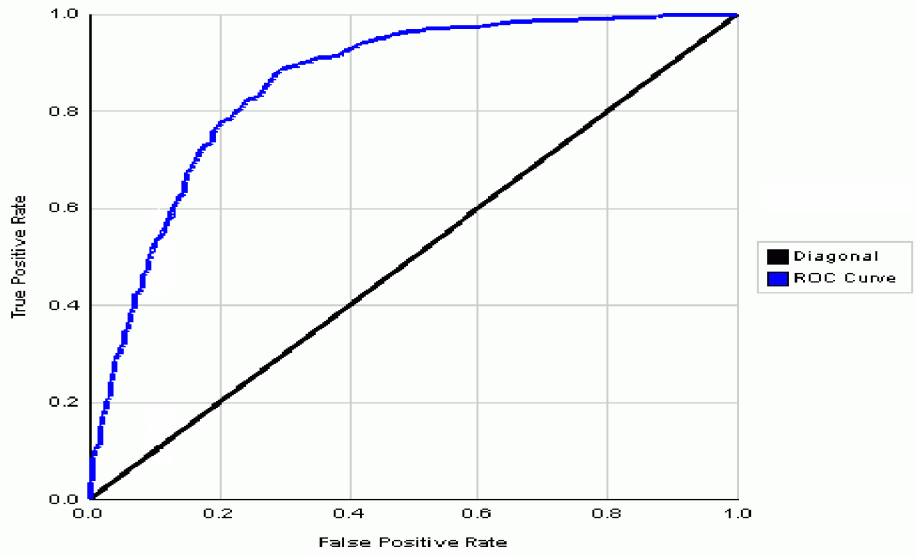
Tab 4.2: Příklad tabulky přesností

- **Graf navýšení** (**Lift**) – měří míru, o kolik je klasifikační model lepší než náhodně generovaná předpověď. Lift lze použít pouze při klasifikaci do binárních tříd a potřebuje stanovit preferovanou hodnotu atributu. Je počítán pro kvantily (*quantiles*), kde každý obsahuje stejný počet prvků cílového atributu. Pro vizualizaci se nejčastěji používá 2D graf a typický počet kvantilů je 10.



Obr. 4.1: Příklad Lift grafu [12]

- **ROC křivky** (**Receiver Operating Characteristics**) – metrika podobná Liftu, která může být použita pro zjištění, jak je model schopný dělat rozhodnutí. Tedy jak je model schopen pozitivně nebo negativně klasifikovat. Vyžaduje klasifikaci do binárních tříd a potřebuje stanovit preferovanou hodnotu atributu. Zobrazuje se jako 2D graf, kde na ose *x* je podíl chybně predikovaných a na ose *y* je podíl správně predikovaných hodnot cílového atributu. Body na křivce se jmenují dělící nebo prahové (*treshold*) a reprezentují poměr mezi správně předikovanými a špatně předikovanými. Důležitým parametrem je velikost plochy pod křivkou, tato hodnota by měla být co největší.



Obr. 4.2: Příklad ROC křivky [11]

Cenová matice (Cost Matrix)

Cenová matice je tabulka *n* x *n*, která definuje cenu spojenou s predikovanou a skutečnou hodnotou klasifikovaného prvku. Číslo *n* je počet tříd, do kterých je cílový atribut klasifikován. Řádek tabulky reprezentuje skutečnou hodnotu a sloupec predikovanou hodnotu. Políčko tabulky je cena, kterou bude ohodnoceno rozhodnutí klasifikovat prvek do třídy reprezentované sloupcem, ve kterém se políčko nachází. Políčka na hlavní diagonále tabulky znamenají, jakou cenou ohodnotíme ty prvky, které byly klasifikovány správně. Proto je jejich hodnota nula. Pokud je cenová matice nastavena pro vyhodnocování, pak nejlepší výsledek klasifikace je s nejmenší cenou.

Například chceme zdůraznit, že pokud budeme špatně klasifikovat zákazníka a povolíme mu nakupovat na splátky, ale zákazník nebude schopen splátky splácet, je to pro nás větší škoda, než pokud nedáme možnost nakupovat na splátky zákazníkovi, který je schopen splácet. V takovém případě nastavíme v cenové matici pro první možnost větší hodnotu.

		předpovídaná třída	
		SPLATKY = ano	SPLATKY = ne
skutečná třída	SPLATKY = ano	0	1
	SPLATKY = ne	**2**	0

Tab. 4.3: Příklad cenové matice

4.1.1.2 Formáty dat atributů v ODM

ODM nepodporuje všechny druhy typů dat, které podporuje databázový systém Oracle. Každý atribut (sloupec tabulky) , který bude používán ODM, musí být jedním z následujících typů.

- INTEGER, NUMBER, FLOAT
- VARCHAR2, CHAR

Podrobnosti o jednotlivých datových typech jsou v [11].

4.1.2 NetBeans

NetBeans je open-source projekt zaštiťovaný a sponzorovaný firmou Sun Microsystems. V současné době je ve verzi 6.5. Celý projekt je rozdělen do dvou částí:

- Vývojové prostředí NetBeans (NetBeans IDE)
- Vývojová platforma NetBeans (NetBeans Platform)

NetBeans IDE je napsáno v jazyce Java a je postaveno na stejnojmenné platformě. Primárně je určeno pro vývoj aplikací v jazyce Java, ale může podporovat i další programovací jazyky. V Javě podporuje všechny 3 hlavní platformy - J2SE, J2EE a J2ME. Obsahuje také RAD nástroje pro tvorbu vzhledu aplikace. Kromě toho také existuje velké množství modulů, které toto vývojové prostředí rozšiřují. Vývojová platforma NetBeans Platform je modulární a rozšířitelný základ pro použití při vytváření rozsáhlých aplikací. Nezávislí dodavatelé softwaru poskytují zásuvné moduly pro integraci do této platformy. Tyto moduly slouží pro vývoj jejich vlastních nástrojů a řešení. Obě části NetBeans jsou bezplatně šířený produkt, který je možné používat bez jakýchkoliv omezení. Je použitelný na operačních systémech Windows, Linux, Mac OS X a Solária [6].

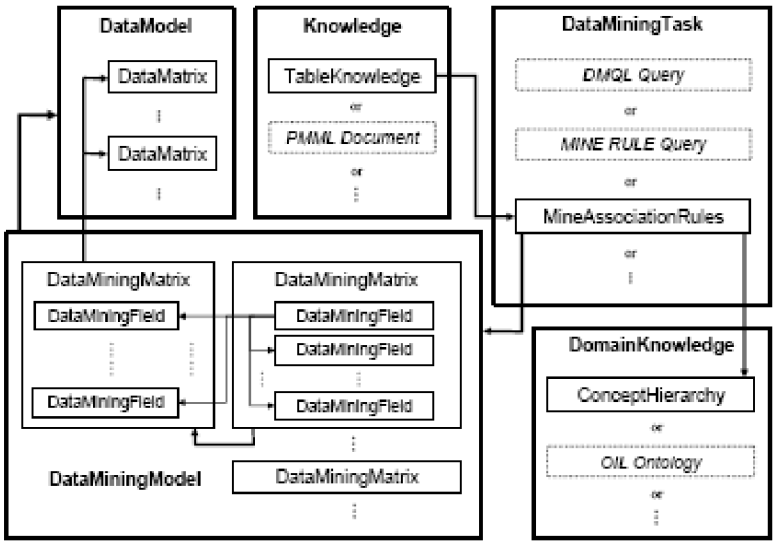
Dřívější řešitelé používaly, při vytváření aplikace Data Mineru NetBeans Platform, protože potřebovali využívat možnosti přídavných modulů. Vývojové prostředí NetBeans IDE 6.5 obsahuje i tuto vývojovou platformu a není tedy nutné pro zajištění modulárnosti projektu využívat NetBeans Platform.

4.1.3 DMSL

DMSL slouží v systému pro ukládání metadat a průběhu procesu dolování. Je to jazyk založený na struktuře jazyka XML, který byl vytvořen jako součást disertační práce na FIT VUT Petrem Kotáskem [5]. Je to jazyk, který se zabývá celým procesem získávání znalostí z databází, definuje vstupní data, transformace apod. Používá se především pro ukládání metadat, tedy popisuje parametry jednotlivých atributů, parametry dolovacích modulů nebo třeba výsledný model některého modulu. Pro potřebu aplikace byly některé jeho části omezeny nebo upraveny.

4.1.3.1 Struktura

Jazyk DMSL tvoří elementy, které jsou definovány v DTD, podobně jak je tomu u jiných jazyků vycházejících z XML. Pro potřebu naší aplikace je jazyk DMSL vybaven elementem pro každou část procesu získávání znalostí (Obr. 4.3).



Obr. 4.3: Struktura DMSL (převzato z [5])

Hlavní elementy:

- **DataModel** – slouží k definici vstupních dat, skládá se z datových matic (DataMatrix), může odkazovat na definice funkcí (FunctionPool).
- **DataMiningModel** – popisuje, které dolovací matice (DataMiningMatrix) se mají používat, jak je spojovat, umožňuje aplikaci transformací. Musí se odkazovat na právě jeden DataModel a může se odkazovat na jeden FunctionPool.
- **DomainKnowledge** – popisuje znalosti o vstupních datech, element s volnou syntaxí, není používán v aplikaci, ale je ponechán pro případný další rozvoj.
- **DataMiningTask** – specifikuje dolovací úlohu, tento objekt by měl obsahovat všechny parametry potřebné pro spuštění dolovací úlohy.
- **Knowledge** – popisuje znalosti získané dolováním, jako je např. model dat, jednotlivé elementy testování, … .

Zvláštními elementy jsou :

- **FunctionPool** – slouží k uchování definic funkcí, které jsou používány při čištění a transformaci. Má jednoznačný název, na který se odkazují jednotlivé elementy.
- **Header** – nepovinný, obsahuje informace o projektu (např. kdo a kdy jej vytvořil)

Z těchto elementů je poskládaný hlavní element DMSL, který je podle definice ve tvaru:

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

Z této definice je patrné, že FunctionPool, DataModel, DataMiningModel, DomainKnowledge, DataMiningTask a Knowledge musí být v elementu DMSL obsaženy jednou nebo vícekrát a element Header nemusí být obsažen vůbec.

4.1.3.2 DataMiningTask, Knowledge

Budeme-li vytvářet modul pro dolování dat, budou nás především zajímat elementy DataMiningTask a Knowlege, protože specifikují dolovací úlohu a vydolované znalosti. Jejich struktura je popsána následovně:

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


<!ELEMENT Knowledge ANY >
<!ATTLIST Knowledge           name              CDATA #REQUIRED
                              language          CDATA #REQUIRED
                              languageVersion   CDATA #IMPLIED
                              type              CDATA #IMPLIED >
```

V původním DMSL není atribut *name*, protože se pravděpodobně nepočítalo s použitím více dolovacích metod. Atribut byl přidán jako povinný (podrobněji v [3]), další povinný je atribut *Language*, ostatní jsou nepovinné. Význam jednotlivých atributů:

- name – jméno, slouží pro propojení elementu Knowlege a DataMiningTask a pro určení, které znalosti patří kterému modulu.
- language – jazyk, ve kterém je dotaz nebo znalost napsána.
- languageVersion – verze jazyka.
- type – typ dotazu nebo znalosti.

Jak je vidět, tak množství a jména elementů, která mají tyto elementy obsahovat, nejsou v původním dokumentu uvedena a je potřeba je definovat pro každou dolovací úlohu samostatně. Obecný návrh těchto elementů není možný, protože každá dolovací úloha má jiné vstupní parametry a jiné výsledné znalosti. Popis rozšíření a úpravy těchto elementů pro klasifikaci rozhodovacím stromem je popsána v kapitole 5.4.

4.2 Současný stav aplikace

Aplikace – Data Miner, která se na VUT FIT vyvíjí v rámci diplomových a bakalářských prací se, zaměřuje na získávání znalostí z dat. Tato aplikace pracuje nad databázovým systémem firmy Oracle. Tento systém obsahuje modul Oracle Data Mining (kapitola 4.1.1.), který má v sobě zabudovanou podporu pro dolování z dat. Data Miner využívá jazyk DMSL pro specifikaci celé dolovací úlohy (vstupních dat, transformací, redukcí, specifikaci dolovacích metod atd.) a ukládání výsledných znalostí.

Jako první začal na projektu dolovacího systému, který spolupracuje s databází Oracle, pracovat Ing. Doležal, který implementoval základní prvky systému. Jako implementační jazyk byla zvolena Java. Jeho projekt byl zaměřen hlavně na vývoj základu (jádra) celého systému, který by

umožňoval provádět základní kroky dolovacího procesu. Jako další se na vývoji podílel Ing. Gálet [9], který nad vytvořeným jádrem vytvořil aplikaci Data Miner postavenou na NetBeans Platform. Této aplikaci vytvořil nové grafické uživatelské rozhraní. Hlavní změnou byl systém definice dolovací úlohy. Jednotlivé kroky dolovací úlohy byly reprezentovány komponentami, které se mezi sebou propojovaly, a tím se určilo pořadí dolovacích kroků aplikace. Toto grafické rozhraní je využíváno v současné aplikaci s několika změnami jednotlivých komponent grafu.

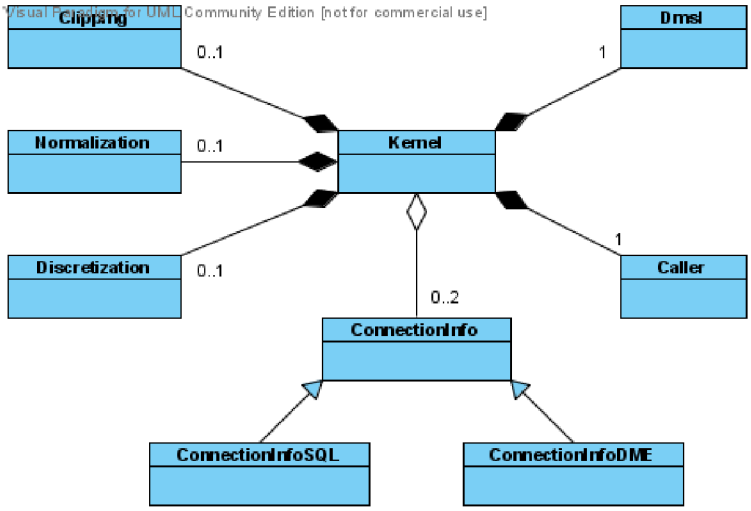
Dalším, který se podílel na vytváření aplikace, byl Ing. Krásný [3], který zásadně přestavěl jádro systému s důrazem na snadnou modifikovatelnost a použitelnost. Došlo ke sloučení několika komponent z důvodu zpřehlednění dolovacího procesu a byla rozšířena funkčnost jádra pro lepší spolupráci komponent v grafu. Bylo zde věnováno velké úsilí pro použitelnost připojování nových modulů, kterými měly být samotné dolovací moduly, protože původní řešení nezapadalo do konceptu. Před touto úpravou se o moduly staralo jádro, které je mělo implementováno jako externí třídu. Tato koncepce byla změněna do stávající podoby, a to tak, že modul je implementován jako jakákoliv jiná standardní komponenta grafu. Zavedl také, že budou všechny informace potřebné pro proces předzpracování, dolování i prezentaci výsledků ukládány výhradně v dokumentu DMSL.

První dolovací modul přidal do aplikace Ing. Mader [8]. Prozkoumal možnosti připojení nových modulů a tento postup zdokumentoval. Připojil modul pro získávání asociačních pravidel. Provedl menší úpravy v jádře systému. Tato práce vychází z jeho poznatků.

V době psaní této práce se pracuje na opravách chyb jádra, přidávání a úpravě prvků dolovacího procesu, které nebyly opraveny Ing. Krásným, v podobě diplomové práce Bc. Šebka. Dalším, kdo současně pracuje na aplikaci Data Miner, ale na jiném dolovacím modulu (pro Bayesovskou klasifikaci) je Bc. Kmoščák a na modulu pro predikci metodou SVM, Bc. Havlíček.

4.2.1 Jádro systému

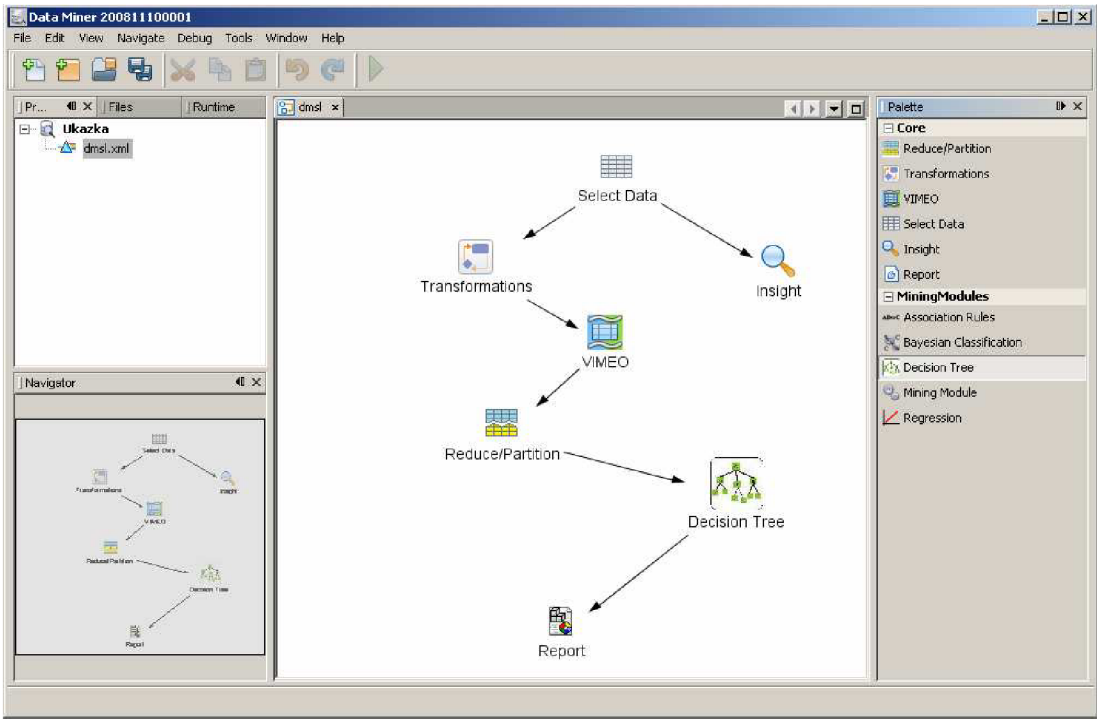
Základní třídou jádra je třída *Kernel.java* (Obr 4.4). Tato třída umožňuje přistupovat ke třídám pro práci s DMSL souborem (třída *Dmsl*), přístup ke zkompilovaným interním funkcím (třída *Caller*) a s připojením k serveru (třída *ConectionInfo*). Instance tříd Clipping, Normalization a Discretization se vytvářejí při volání metody obsluhující spuštění běhu komponenty Transformations. Zajišťují inicializaci parametrů pro server a jejich odeslání.



Obr. 4.4: Diagram tříd Kernel.java podle [3]

4.2.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je rozděleno do několika částí (Obr 4.5). V levé části je průzkumník projektů, kde je tučně zvýrazněn ten projekt, který je aktuální. Pod ním je celkový pohled na dolovací úlohu. Zde se počítalo, že graf bude s přibývajícími komponentami větší a bude tato přehledová část využívána. Prostřední část, tzv. pracovní plocha, zaujímá prostor pro vytváření dolovací úlohy. V pravé části je umístěna paleta s jednotlivými komponentami dolovacího procesu. Komponenty lze jednoduše tažením přemístit na pracovní plochu a propojením šipkami z jedné komponenty do druhé, vznikne graf dolovací úlohy.



Obr. 4.5: GUI aplikace Data Miner

Komponenty

- **Select Data –** Komponenta zajišťuje výběr vstupních dat. Obsahuje záložky pro výběr zdroje dat. V záložce Columns Selection může uživatel libovolně vybírat sloupce, ve kterých jsou data, která mají být použita v dolovací úloze. Je zde také záložka, sloužící ke vkládání podmínek pro spojování tabulek.
- **Insight –** Spuštěním komponenty výběru dat nebo komponent fáze předzpracování vzniká v databázi fyzická tabulka, která představuje výstupní data těchto komponent. Po připojení ke komponentám umožňuje komponenta *Insight* zobrazení obsahu těchto tabulek. Zobrazuje také statistické údaje.
- **Transformations –** Poskytuje možnost aplikace ODM transformací clipping, normalization a discretization na vstupní data. Také umožňuje nezahrnout do výstupu vstupní sloupce a pomocí funkcí odvodit sloupce nové. Zajišťuje přístup k editacím funkcí.
- **Vimeo –** Tato komponenta funguje jako datový filtr. Všechny výstupní sloupce jsou stejné jako vstupní, pouze je k nim přiřazena nějaká *vimeo* funkce. Neumožňuje sloupce přejmenovávat, od toho je komponenta Transformations. Zajišťuje přístup k editacím funkcí.

- **Reduce/Partition –** Tato komponenta byla přidána z důvodu potřeby takto rozdělit data pro klasifikační a prediktivní dolovací moduly. Dokáže rozdělit vstupní data na trénovací a testovací část podle zvoleného poměru. Také dovede snížit počet záznamů ve vybraných datech.
- **Decision Tree –** Reprezentuje dolovací modul, který získává znalosti pomocí klasifikační metody rozhodovacího stromu. Aplikace obsahuje ještě několik dalších modulů.
- **Report -** Zajišťuje zobrazení výsledků. Má specifické rozhraní pro každý modul, odpovídající požadavkům prezentace výsledků dané dolovací úlohy.

V této diplomové práci byla řešena komponenta Decision Tree (kapitola 5.5.2 a 5.5.3) a Report (kapitola 5.5.4).

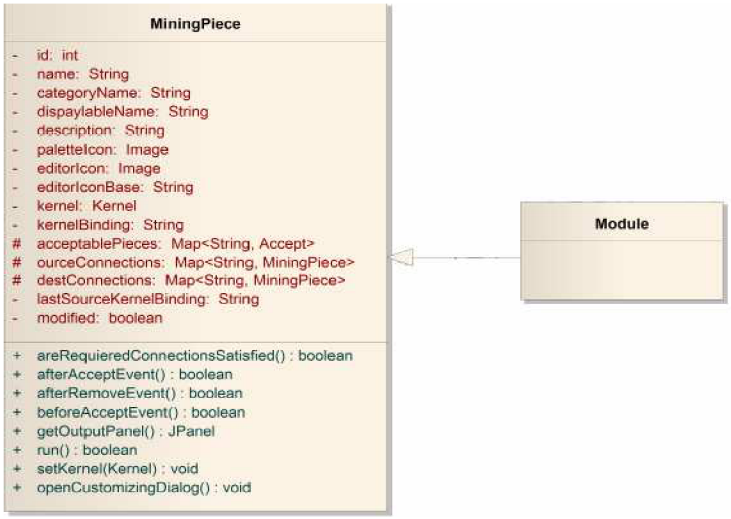
5 Implementace modulu rozhodovacího stromu

Pro implementaci modulu rozhodovacího stromu je potřeba vědět, jak se nový modul vytváří a jak se má správně začlenit do aplikace. Využití podpory Oracle Data Mining s sebou nese některé požadavky na parametry dolovací úlohy a na vstupní data. Závislost na ODM nám specifikují omezení výstupního modelu. Všechny tyto okolnosti je potřeba akceptovat a přizpůsobit jim aplikaci. Je potřeba se dále zamyslet nad strukturou DMSL dokumentu, který je jediným úložištěm metadat pro celý proces získávání znalostí z dat, tedy i výstupních hodnot dolování.

Tato kapitola se zabývá všemi těmito aspekty přípravy tvorby dolovacího modulu. Je zde i popis aplikace modelu na nová data, ale tato část není, po dohodě s vedoucím, součástí implementace. Stručně popisuje implementaci modulu klasifikace pomocí metody rozhodovacího stromu a v závěru se soustřeďuje na prezentaci vydolovaných znalostí.

5.1 Způsob začlenění nového modulu

Způsob začlenění nového modulu je podrobně popsána v [8], z tohoto důvodu zde bude zmíněn jen základní princip. Pro jakýkoliv nový modul je nezbytné implementovat rozhraní *MiningPiece* (Obr 5.1), protože pouze třída, která dědí vlastnosti tohoto rozhraní, se dá poté vložit do dolovacího procesu jako nová komponenta (prvek grafu).



Obr. 5.1: Třída MiningPiece

Pro správnou funkčnost nového modulu je zapotřebí implementovat všechny metody tohoto rozhraní. Zde jsou uvedeny pouze ty nejdůležitější. Tedy ty, které se nějakým způsobem přímo podílejí na dolovacím procesu.

- **openCustomizingDialog() –** vytváří a otevírá panel, do kterého se zadávají parametry modulu. Tato metoda je volána při otevření panelu.
- **getOutputPanel() –** vytváří a otevírá panel standardní komponenty Report, ve kterém se budou zobrazovat výsledky dolovacího procesu.

- **run() –** spouští akci komponenty. V případě dolovacího modulu v této metodě probíhá samotný proces aplikace algoritmu získání znalostí na vstupní data.

5.2 Rozhraní ODM

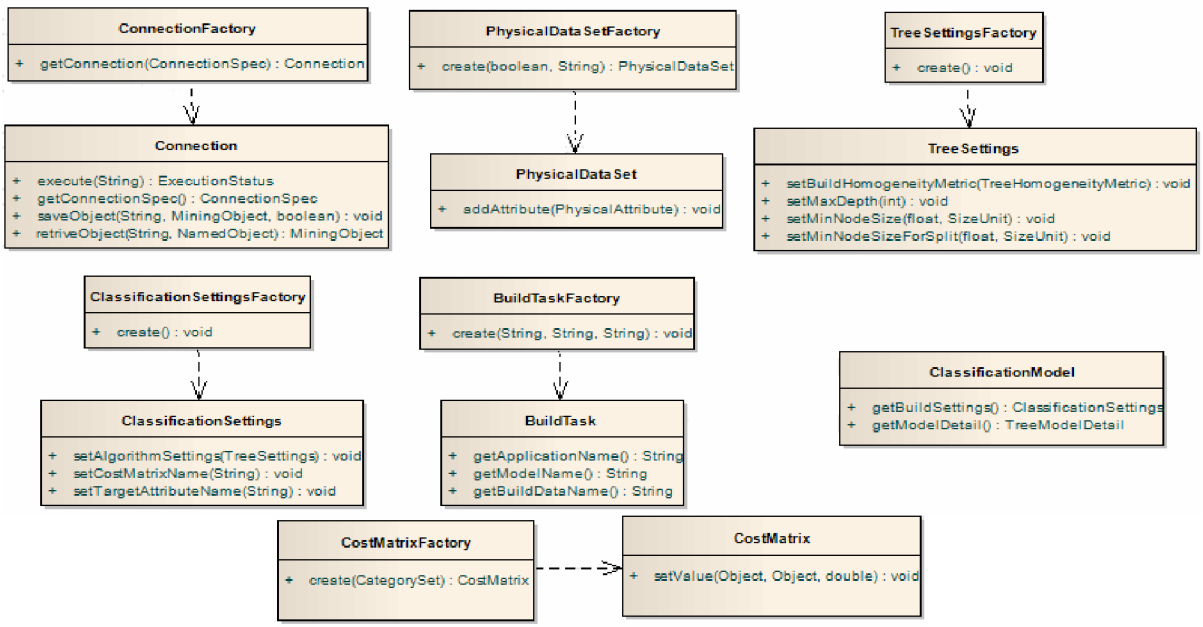
Pro vytvoření a spuštění dolovací úlohy je nutné implementovat některé části rozhraní ODM. Podrobně jsou všechny části popsány v [11]. Celý proces je rozdělen do tří částí, kde každá z částí reprezentuje část procesu klasifikace. První část je proces vytváření modelu (učení), druhá část obsahuje testování vytvořeného modelu a tvorbu vyhodnocovacích metrik a poslední část je aplikace modelu na nová data. O vytváření instancí potřebných tříd se stará metoda create() příslušné Factory. Celý proces začíná připojením k data mining enginu pomocí objektu třídy *Connection*. Přes tento objekt probíhá ukládání objektů na server, spuštění úlohy, získání vytvořeného dolovacího modelu a získání metrik z průběhu testování.

5.2.1 Vytvoření modelu

Při vytváření modelu rozhodovacího stromu je nejdříve potřeba vytvoření objektu *PhysicalDataSet*, který popisuje strukturu dat určených k naučení. Dalším krokem je sestavení rozhodovacího stromu, které se provádí v objektu *TreeSettings*. V tomto objektu probíhá nastavování všech parametrů stromu. Mezi nejdůležitější patří:

- **setBuildHomogeneityMetric** – metoda pro volbu významnosti atributů. ODM podporuje pouze dvě varianty Gini a Entropy.
- **setMaxDepth –** nastavení maximální hloubky stromu. Maximální hodnota je 20.
- **setMinNodeSize –** minimální povolený počet prvků v uzlu. Udává se buď počtem prvků nebo procenty z celku.
- **setMinNodeSizeForSplit -** minimální povolená velikost uzlu pro rozdělení. Udává se buď počtem prvků nebo procenty z celku.

Pro nastavení klasifikačního modelu slouží objekt třídy *ClassificationSettings*. Tomuto objektu se musí nastavit parametry dolovací metody, jméno cenové matice (pokud je použita) a jméno cílového atributu. Dolovací úloha je reprezentována třídou *BuildTask*, které je při vytváření potřeba nastavit specifikaci dat, specifikaci nastavení a jméno modelu. Po provedení úlohy se na serveru vytvoří model, který reprezentuje danou úlohu. Pro získání modelu ze serveru je potřeba volat metodu *retrieveObject* třídy *Conection*. Z něj je poté možno získat pravidla pro rozhodovací strom. Na obr 5.2 jsou základní třídy ODM pro vytvoření modelu rozhodovacího stromu.



Obr. 5.2 : Třídy ODM pro vytvoření modelu rozhodovacího stromu

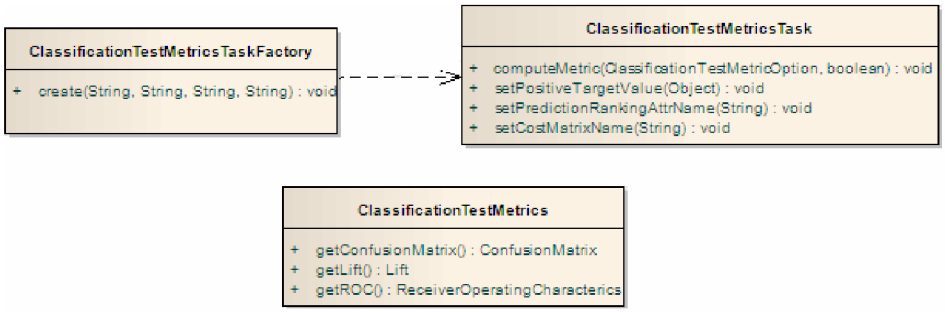
Výsledný model je uložen ve třídě *TreeModelDetail*. Tato třída je specifikována ve standardu Java Data Mining. Model je zde uložen jako stromová struktura (binární strom) obsahující identifikaci uzlu, počet následovníků, predikovanou hodnotu, počet takto klasifikovaných případů a pravidlo, ve formátu *atribut* pak *znak nerovnost(<= nebo >)* a *hodnota*. Některé uzly mohou obsahovat také náhradní pravidla (*surogates*), která reprezentují pravidla se stejným rozhodovacím výsledkem. Listové uzly obsahují navíc ještě informace o podpoře a spolehlivosti.

5.2.2 Testování modelu

Pro testování modelu se musí vytvořit instance třídy *PhysicalDataSet*, která popisuje strukturu dat určených k testování. Dále musí být vytvořen objekt třídy *ClassificationTestMetricsTask*, který umožňuje nastavení několika parametrů pro testování:

- **computeMetrics** – nastavení požadovaných testovacích metrik (ROC, Lift, tabulka přesnosti)
- **setNumberOfLiftQuantiles** – nastavení počtu kvantilů (*quantiles*) v metrice Lift
- **setCostMatrix** – nastavení cenové matice (pokud je použita)
- **setPositiveTargetValue** - preferovanou hodnotu testovaného atributu

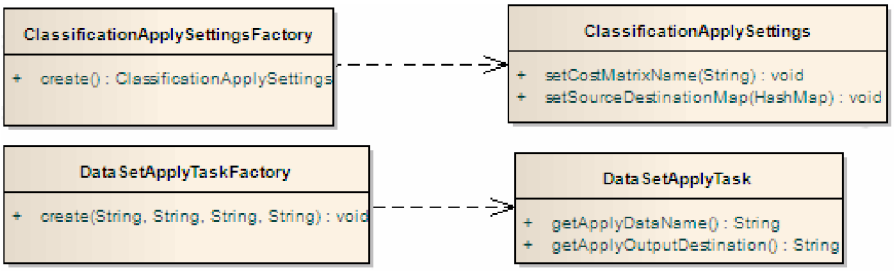
Výstupem je objekt třídy *ClassificationTestMetrics*, který se získá voláním metody *retrieveObject* třídy *Conection*. Z tohoto objektu lze získat všechny metriky, které byly specifikovány. Obrázek 5.3 ukazuje základní třídy pro vytvoření testovacích metrik.



Obr. 5.3: Třídy pro vytvoření testovacích metrik

5.2.3 Aplikace modelu na nová data

Pro testování modelu se musí vytvořit instance třídy *PhysicalDataSet*, která popisuje strukturu dat určených k aplikaci. Dále musí být vytvořen objekt třídy *ClassificationApplySettings*, které se nastaví predikovaný atribut. Aplikační proces je reprezentován třídou *DataSetApplyTask*, které je při vytváření potřeba nastavit specifikaci dat, na která bude model aplikován, naučený klasifikační model, nastavení aplikačního mechanismu a jméno tabulky, do které se uloží výsledek. Výsledek je dostupný v databázi a na jeho získání stačí dotaz v jazyce PL\SQL.



Obr. 5.4: Třídy pro aplikaci

5.3 Vstupní data

Jako vstupní data slouží standardní relační tabulka, kde každý prvek by měl být reprezentován jedním řádkem a každý atribut jedním sloupcem tabulky. Sloupec tabulky musí být v jednom z typů dat, které ODM podporuje (4.1.1.2).

V tabulce by měl být predikovaný atribut diskretizován. Pokud bychom měli klasifikovat do velkého množství tříd, budou výsledky metody nepřesné. Vstupní tabulka může obsahovat i nadbytečné atributy, protože při vytváření modelu použije ODM jen ty, které mají nějaký vliv na výsledný model.

Je potřeba vědět, jak zacházet s hodnotami atributů, které by mohly ovlivnit tvorbu modelu. Mezi takové patří především odlehlé nebo NULL hodnoty.

Odlehlé hodnoty

Odlehlé hodnoty jsou takové, které se výrazně liší od ostatních. Tyto hodnoty nemají vliv na tvorbu modelu pro rozhodovací strom.

NULL hodnoty

NULL hodnoty jsou u metody rozhodovacího stromu reprezentovány jako chybějící hodnoty.

5.4 Úprava DMSL

Tato kapitola se věnuje elementům DataMiningTask a Knowledge DMSL dokumentu, které byly přizpůsobeny potřebě modulu rozhodovacího stromu. Základní struktura těchto elementů je popsána v kapitole 4.1.3.2. Jejich přesná struktura není obecně specifikována z důvodu odlišnosti dolovacích metod. Úplná specifikace úprav DMSL dokumentu je v příloze A.

5.4.1 DataMiningTask

DataMiningTask specifikuje dolovací úlohu a její parametry. Definice elementu pro rozhodovací strom má podobu:

```
<!ELEMENT DecisionTree (UseMatrix, InputDataType, Parameters, CostMatrix,
ConfMatrix, Lift, ROC)>
<!ATTLIST DecisionTree  splitCriter (Gini|Entropy) #REQUIRED>
```

Element *DecisionTree* reprezentuje úlohu klasifikace založené na rozhodovacím stromu. Je v něm uložena informace o dělícím kritériu, který bude použit *splitCriter*. Jak bylo naznačeno v kapitole 5.2.1, může tento atribut nabývat pouze hodnot *Gini* nebo *Entropy*. Obsahuje další elementy *UseMatrix, InputDataType, Parameters*.

```
<!ELEMENT UseMatrix>
<!ATTLIST UseMatrix     testMatrix CDATA #REQUIRED>
                        trainMatrix CDATA #REQUIRED>
```

Element *UseMatrix* definuje tabulky, které budou použity pro dolování. Atribut *testMatrix* obsahuje jméno tabulky pro testování a atribut *trainMatrix* jméno tabulky pro trénování modelu.

```
<!ELEMENT InputDataType>
<!ATTLIST InputDataType       tableKey CDATA #REQUIRED>
                              prefVal CDATA #REQUIRED>
                              target CDATA #REQUIRED>
```

Element *InputDataType* definuje atribut pro dolování *target*, jeho preferovanou hodnotu *prefVal* a atribut reprezentující primární klíč tabulky, která bude použita pro trénování a testování modulu - *tableKey*.

```
<!ELEMENT Parameters (MaxDepth, MinNodeSizeP, MinNodeSizeC, MinSplitRecP,
MinSplitRecC)>

<!ELEMENT MaxDepth (#PCDATA)>

<!ELEMENT MinNodeSizeP (#PCDATA)>

<!ELEMENT MinNodeSizeC (#PCDATA)>

<!ELEMENT MinSplitRecP (#PCDATA)>

<!ELEMENT MinSplitRecC (#PCDATA)>
```

Element *Parameters* obsahuje elementy:

- *MaxDepth* – reprezentuje maximální hloubku stromu
- *MinNodeSizeP* – minimální povolená velikost uzlu. Udává se procentuelně z celkového počtu.
- *MinNodeSizeC* – minimální povolená velikost uzlu. Udává se absolutním počtem prvků.
- *MinSplitRecP* – minimální povolená velikost uzlu pro rozdělení. Udává se procentuelně z celkového počtu.

- *MinSplitRecP* – minimální povolená velikost uzlu pro rozdělení.Udává se absolutním počtu prvků

```
<!ELEMENT CostMatrix (Rows*)>
<!ATTLIST CostMatrix          name CDATA #REQUIRED>
                              type (INT|STRING|DATE|REAL) #IMPLIED>

<!ELEMENT Rows (Value+)>
<!ATTLIST Rows          value CDATA #REQUIRED>

<!ELEMENT Value (#PCDATA)>
<!ATTLIST Value         source CDATA #REQUIRED>
                        target CDATA #REQUIRED>
```

Element *CostMatrix* obsahuje atribut *name*, který jej identifikuje. Pokud cenová matice není použita, atribut obsahuje *null* a pak je element prázdný. Atribut *type* je nepovinný a reprezentuje typ pedikovaného atributu. *Rows* je element, který v atributu *value* uchovává třídy atributu a obsahuje elementy *Value*, které reprezentují jednotlivé prvky cenové matice patřící k této třídě. Atribut *source* ukazuje skutečnou třídu atributu a *target* představuje třídu, do které byl atribut klasifikován.

```
<!ELEMENT ConfMatrix>
<!ATTLIST ConfMatrix          use (true|false) #REQUIRED>

<!ELEMENT ROC>
<!ATTLIST ROC           use (true|false) #REQUIRED>
```

ConfMatrix obsahuje atribut *use*, který nese informaci, zda budeme požadovat vypočet tabulky přesnosti. Podobný význam má element *ROC* a jeho atribut *use*.

```
<!ELEMENT Lift (Count?)>
<!ATTLIST Lift          use (true|false) #REQUIRED>

<!ELEMENT Count (#PCDATA)>
```

Element *Lift*, který obsahuje stejný atribut *use*, obsahuje navíc element Count, který zaznamenává pro kolik kvantilů bude metrika počítána.

5.4.2 Knowledge

Element *Knowledge* reprezentuje vydolované znalosti. V případě, že se jedná o klasifikační algoritmus, můžeme elementy rozdělit do dvou skupin. První jsou elementy pro trénování (sestavení) modelu, v našem případě označené *Build*, následuje element *Test,* pro uložení výsledku testovacích metrik.

Element Build

```
<!ELEMENT Build (Parameters, Attributes, TreeModel)>
<!ATTLIST Build         buildTable CDATA #REQUIRED>
                        name CDATA #REQUIRED>
                        date CDATA #IMPLIED>
                        target CDATA #REQUIRED>
```

Build obsahuje atribut *buildTable*, který obsahuje jméno trénovací tabulky, *date* obsahuje datum a čas vytvoření modelu, atribut *name* je jméno dolovací úlohy a *target* je cílový atribut, který chceme klasifikovat. Element *Parameters*, který obsahuje parametry dolovací úlohy a je totožný se stejnojmenným parametrem v elementu *DataMiningTask*.

```
<!ELEMENT Attributes (Attribute+)>

<!ELEMENT Attribute>
<!ATTLIST Attribute           name CDATA #REQUIRED>
                              type CDATA #REQUIRED>
```

V elementu *Attributes* jsou uložena jména sloupců trénovací tabulky (element *Atribute*), které dolovací úloha potřebovala k naučení. Atribut *name* reprezentuje jméno sloupce a v atributu *type* je uložen typ hodnot atributu (double, …).

```
<!ELEMENT TreeModel (Nodes+)>
<!ATTLIST TreeModel     leavNode NUMBERS #REQUIRED>
                        numNodes NUMBERS #REQUIRED>
                        treeDepth CDATA #REQUIRED>

<!ELEMENT Nodes (Root, Node+)>
```

TreeModel reprezentuje element rozhodovacího stromu, kde *Nodes* jsou uzly stromu.

- *leavNode* – počet listových uzlů stromu
- *numNodes* – celkový počet uzlů
- *treeDepth* – hloubka stromu

```
<!ELEMENT Root >
<!ATTLIST Root    caseCount INT-NUMBER #REQUIRED>
                  id CDATA #REQUIRED>
                  prediction CDATA #REQUIRED>
```

Element *Root* je kořenový uzel stromu, který obsahuje údaj o počtu případů při trénování (*caseCount*), identifikaci uzlu (*id*) a hodnotu predikovaného atributu (*prediction*).

```
<!ELEMENT Node (Predicate, Surrogate*, Rule?)>
<!ATTLIST Node    caseCount NUMBERS #REQUIRED>
                  id CDATA #REQUIRED>
                  level NUMBERS #REQUIRED>
                  chilCount NUMBER #REQUIRED>
                  parent CDATA #REQUIRED>
                  prediction CDATA #REQUIRED>
```

Node reprezentuje uzel stromu. Význam atributů je stejný jako u elementu *Root*. Navíc obsahuje atribut pro uložení počtu synovských uzlů – *chilCount*, odkaz na svého předchůdce – *parent* a úroveň stromové struktury, ve které se tento uzel nachází - *level*.

```
<!ELEMENT Predicate>
<!ATTLIST Predicate     flag(lessOrEqual|grtrOrEqual|greater|less|equal)
                        #REQUIRED>
                        name CDATA #REQUIRED>
                        value CDATA #REQUIRED>

<!ELEMENT Surrogate>
<!ATTLIST Surrogate     flag(lessOrEqual|grtrOrEqual|greater|less|equal)
                        #REQUIRED>
                        id NUMBERS #REQUIRED>
                        name CDATA #REQUIRED>
                        value CDATA #REQUIRED>
```

Predicate reprezentuje pravidlo, podle kterého byl strom rozdělen do části podstromu, ve které se nachází. Atribut *flag* reprezentuje znaménko nerovnosti, *name* je jméno sloupce tabulky, *value* je hodnota pravidla. *Surrogate* je pravidlo, které má pro dělení stromu stejnou rozhodovací schopnost jako *Predicate*. Tomu odpovídají i stejné parametry. Protože elementů *Surrogate* může být obecně více, byl do tohoto elementu přidán atribut *id*, který jej identifikuje.

```
<!ELEMENT Rule (Consequent)>
<!ATTLIST Rule          confidence NUMBERS #REQUIRED>
                        support NUMBERS #REQUIRED>

<!ELEMENT Consequent>
<!ATTLIST Consequent    name CDATA #REQUIRED>
                        value CDATA #REQUIRED>
```

Pokud je *Node* uzlem listovým, obsahuje element *Rule*, který reprezentuje klasifikační pravidlo. Atribut *confidence* označuje spolehlivost a *support* značí podporu tohoto pravidla. Oba tyto termíny jsou podrobněji popsány v kap. 2.1.3. Element Consequent obsahuje jméno cílového atributu (*name*) a klasifikovanou třídu (*value*).

Element Test

```
<!ELEMENT Test (ConfusionMatrix, Lift, Roc)>
<!ATTLIST Test          testTable CDATA #REQUIRED>
                        name CDATA #REQUIRED>
```

V elementu *Test* jsou uloženy výsledky testování modelu. Atribut *testTable* obsahuje jméno tabulky s testovacími daty a *name* je jméno testovací části procesu. Obsahuje elementy *ConfusionMatrix*, *Lift* a *Roc*, ve kterých je uložena většina parametrů pro testovací metriky (metriky jsou popsány v 4.1.1.1), které je možné z ODM získat.

```
<!ELEMENT ConfusionMatrix (Item+)>
<!ATTLIST ConfusionMatrix     accuracy NUMBERS #REQUIRED>
                              error NUMBERS #REQUIRED>

<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item          actual CDATA #REQUIRED>
                        predicate CDATA #REQUIRED>
```

ConfusionMatrix reprezentuje tabulku přesností, kde *accuracy* je přesnost modelu a *error* je chybovost modelu. Tento element může obsahovat několik položek elementu *Item*, který představuje počet n-tic testovací tabulky, které byly klasifikovány do třídy *predicate*, a skutečná hodnota měla být *actual*. Atribut *actual* představuje řádek této tabulky a *predicate* sloupec.

```
<!ELEMENT Lift (Quantile+)>
<!ATTLIST Lift          cases NUMBERS #REQUIRED>
                        numOfQuantiles NUMBERS #REQUIRED>
                        positiveCases NUMBERS #REQUIRED>
                        positiveTargetValue CDATA #REQUIRED>
                        targetName CDATA #REQUIRED>
```

Lift reprezentuje stejnojmennou testovací metriku. Atribut *cases* slouží pro uložení počtu případů, *numOfQuantiles* reprezentuje počet počítaných kvantilů, positiveCases je počet správně predikovaných n-tic, positiveTargetValue je preferovaná hodnota cílového atributu a targetName je cílový sloupec.

```
<!ELEMENT Quantile>
<!ATTLIST Quantile      cumLift NUMBERS #REQUIRED>
                        cumNonTargets NUMBERS #REQUIRED>
                        cumRecordP NUMBERS #REQUIRED>
                        cumTargetDensity NUMBERS #REQUIRED>
                        cumTargets NUMBERS #REQUIRED>
                        id CDATA #REQUIRED>
                        quantLift NUMBERS #REQUIRED>
                        quantTargetCount NUMBERS #REQUIRED>
                        quantTotalCount NUMBERS #REQUIRED>
                        targetDensity NUMBERS #REQUIRED>
```

Quantile reprezentuje jeden kvantil pro výpočet testovací metriky Lift.

- *id* – pořadové číslo kvantilu (*n)*
- *cumLift* – poměr *cumTargetDensity* kvantilu k celkové chybovosti všech testovacích dat
- *cumNonTargets* – součet špatně klasifikovaných prvků ze všech předešlých kvantilů
- *cumRecordP* – procento všech testovaných prvků ze všech předešlých kvantilů
- *cumTargetDensity* – součet všech *targetDensity* ze všech předešlých kvantilů
- *cumTarget* – součet správně klasifikovaných prvků ze všech předešlých kvantilů
- *quantLift* – poměr *targetDensity* kvantilu k celkové chybovosti všech testovacích dat
- *quantTargetCount* – počet správně klasifikovaných prvků v kvantilu
- *quantTotalCount* – počet všech prvků v kvantilu
- *targetDensity* – poměr správně klasifikovaných prvků jako dobré ku celkovému počtu prvků v kvantilu

```
<!ELEMENT Roc (Cadndidate+)>
<!ATTLIST Roc           areaUCurve NUMBERS #REQUIRED>
                        numTresholCand NUMBERS #REQUIRED>
```

Roc reprezentuje stejnojmennou testovací metriku. V atributu *areaUCurve* je velikost plochy pod ROC křivkou, vyjádřeno v procentech, *numTresholdCan* je počet dělících bodů na ROC křivce.

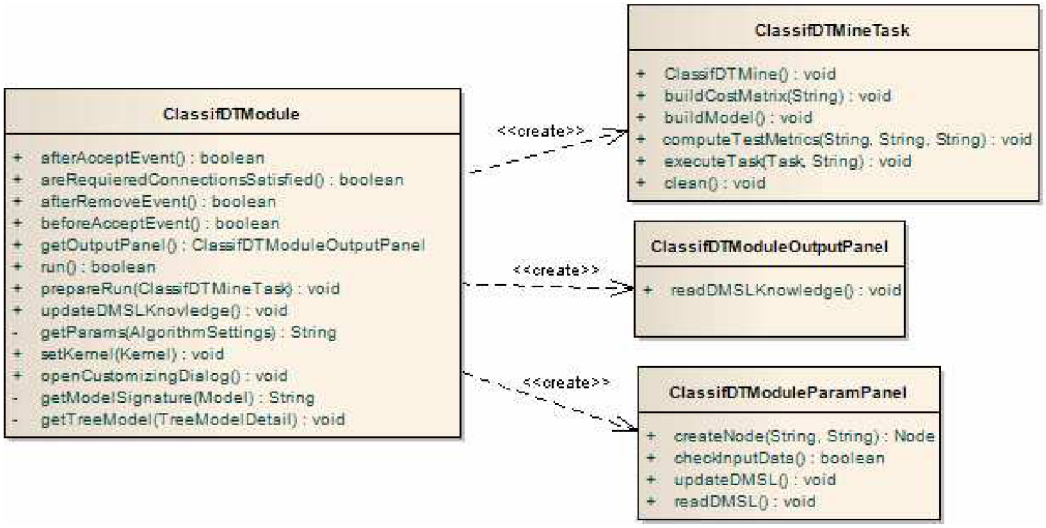
```
<!ELEMENT Cadndidate >
<!ATTLIST Cadndidate    falseNeg NUMBERS #REQUIRED>
                        falsePos NUMBERS #REQUIRED>
                        falsePosFrac NUMBERS #REQUIRED>
                        id CDATA #REQUIRED>
                        probability NUMBERS #REQUIRED>
                        trueNeg NUMBERS #REQUIRED>
                        truePos NUMBERS #REQUIRED>
                        truePosFrac NUMBERS #REQUIRED>
```

Candidate je element, který reprezentuje bod na ROC křivce. Atributy:

- *id-* identifikace
- *falseNeg* – počet prvků chybně klasifikovaných jako špatné
- *falsePos* - počet prvků chybně klasifikovaných jako správné
- *falsePosFrac - truePos/(truePos +falseNeg))*
- *trueNeg* - počet prvků dobře klasifikovaných jako špatné
- *truePos* - počet prvků dobře klasifikovaných jako správné
- *truePosFrac- (falsePos/(falsePos + trueNeg))*
- *probability* – pravděpodobnost, že bude tento bod vybrán jako nejlepší prahová hodnota

5.5 Implementace modulu

Pro implementaci nového modulu je zapotřebí implementovat rozhraní *MiningPiece*. Toto rozhraní implementuje třída *ClassifDTModule*, která zajišťuje komunikaci s jádrem. Je takovým rozcestníkem pro všechny funkce, které modul poskytuje. V konstruktoru této třídy je nastavení jména a popisu přes lokalizační bunde a také je zde nastavení cesty k ikonám modulu, které budou zobrazeny v aplikaci. Obrázek 5.5 ukazuje hlavní třídy dolovacího modelu, které se starají o zadávání parametrů, průběh dolování a zobrazení získaných znalostí.



Obr. 5.5: Třídy pro modul klasifikačního stromu

Třída *ClassifDTModule* se stará o spouštění akcí vyvolaných uživatelem a ukládání dat do DMSL dokumentu. Protože data nemohou být ukládána jinam než do DMSL dokumentu, musí mít uživatel možnost uložit data v jakémkoliv kroku dolovacího procesu.

5.5.1 Vložení modulu

Před vložením komponenty dolovacího modulu do grafu, DMSL dokument neobsahuje o modulu žádné informace, je tedy nutné ihned po vložení provést záznam o modulu do DMSL dokumentu. Toto uložení provádí metoda *afterAcceptEvent()*. Tím se v DMSL vytvoří element *DataMiningTask* a element *Knowledge*, který má nastaveny výchozí hodnoty.

```
<DataMiningTask language="XML" name="DM995_ClassifDTModule1">
      <DecisionTree splitCriter="Gini">
            <UseMatrix testMatrix="null" trainMatrix="null"/>
            <InputDataType  prefVal="null" tabKey="null" target="null"/>
            <Parameters >
                  <MaxDepth >7 </MaxDepth>
                  <MinNodeSizeP>0.05 </MinNodeSizeP>
                  <MinNodeSizeC>10 </MinNodeSizeC>
                  <MinSplitRecP>0.1 </MinSplitRecP>
                  <MinSplitRecC>20 </MinSplitRecC>
            </Parameters>
            <CostMatrix  name="null"></CostMatrix>
            <ConfMatrix  use="true"></ConfMatrix>
            <Lift  use="true">
                  <Count >10 </Count>
            </Lift>
            <Roc  use="true"></Roc>
      </DecisionTree>
</DataMiningTask>
<Knowledge  language="XML" name="DM995_ClassifDTModule1">
      <Build/><Test/>
</Knowledge>
```

Obr 5.6: Ukázka DMSL nového modulu rozhodovacího stromu bezprostředně po vložení do grafu dolovacího procesu

Pokud budeme chtít modul odstranit, odstraníme jej z grafu dolovacího procesu a zároveň jej musíme odstranit i z DMSL dokumentu. Pro odstranění slouží metoda *afterRemoveEvent()*. Pro případ, že bychom chtěli kontrolovat nějaký stav před vložením komponenty do grafu, je zde metoda *beforeAcceptEvent()* nebo pokud bychom chtěli kontrolovat nějaký stav před odstraněním komponenty z grafu, můžeme použít metodu *beforeRemoveEvent()*.

5.5.2 Panel parametrů

Panel parametrů se obsluhuje voláním metody *openCustomizingDialog()*. Tento panel slouží uživateli pro zadávání parametrů pro dolovací úlohu. Třída reprezentující panel parametrů, která je potomkem třídy *javax.swing.JPanel*, se jmenuje *ClassifDTModuleParamPanel*. Vzhledem k různorodosti dolovacích modulů, je třídě reprezentující panel parametrů předáván celý element DataMiningTask. Objekt třídy *ClassifDTModuleParamPanel* se stará o zobrazení parametrů při otevření panelu. Parametry jsou pomocí metody *readDMSL()* načteny a zobrazeny v prvcích panelu. Po ukončení editace jsou stiskem tlačítka OK všechny parametry zkontrolovány, zda jsou správně zadány, a uloženy zpátky do DMSL pomocí metody *updateDMSL()*. Pokud se vyskytne chyba při zadávání hodnot parametrů je zobrazena, upozorňující zpráva a parametry se do DMSL neuloží.

5.5.3 Spuštění modulu

Pro spuštění modulu slouží, v objektu třídy *ClassifDTModule*, metoda *run()*. Parametry pro dolovací úlohu jsou předávány dolovací úloze pomocí metody *prepareRun()*, která získá parametry z DMSL dokumentu a pomocí metod *set*, je nastaví v dolovací úloze. Dolovací úloha je reprezentována třídou *ClassifDTMineTask*. Samotné dolování se spouští metodou *ClassifDTMine()* objektu dolovací třídy.

Jako první je potřeba nastavit americké nebo anglické rozhraní pro databázi. Toto nastavení zabrání rozdílně interpretovat oddělovače desetinný míst (podrobněji v [8]). Data není potřeba nijak speciálně upravovat, protože metoda přijímá standardní relační tabulky. Před spuštěním samotného procesu dolování je potřeba odstranit tabulky, které mohly v databázi zůstat po předchozích úlohách a mohly by kolidovat s námi vytvořenými. Následuje proces vytvoření cenové matice, pokud je použita (*buildCostMatrix()*), vytvoření modelu (*buildModel()*), spočítání testovacích metrik (*computeTestMetrics()*) a odstranění nepotřebných tabulek (c*lean()*). Při implantaci byl nalezen menší nedostatek jádra aplikace, v podobě chybějícího balíčku `javax.datamining.supervised.classification`. Jedná se o Java třídy pro práci s nastavením dolovací úlohy, vytvořením modelu a testování pro metody klasifikace. Tento nedostatek byl odstraněn.

Získaný model a testovací metriky jsou vráceny do *ClassifDTModule*, kde je metoda *updateDMSLKnovledge()* uloží do DMSL. Při ukládání testovacích metrik jsou použity metody třídy *Metrics* z balíku `cz.vutbr.fit.dataminer.classifdtmodule.metrics`, která metodou *save* příslušné metriky uloží tabulku přesnosti, Lift a ROC křivku ve formátu podle definice v 5.4.2.

5.5.4 Prezentace výsledků

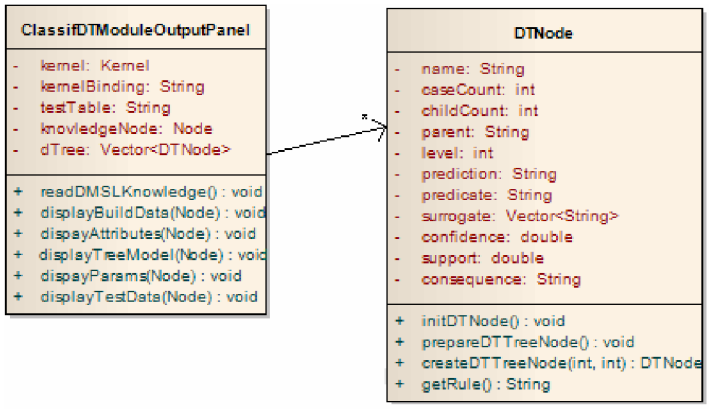
Prezentaci výsledků lze rozdělit do dvou částí. První část obsahuje prezentaci získaného naučeného modelu trénovacích dat. V našem případě je reprezentován objektem třídy *TreeModelDetail*. Každý dolovací algoritmus, podporovaný ODM, má tento model jiný a to i pro algoritmy klasifikace. Není tedy možné vytvořit univerzální prostředek pro jejich prezentaci. Druhá část výsledků obsahuje testovací metriky, které byly vytvořeny na základě dat z trénovací tabulky. Třída, kterou jsou

testovací metriky reprezentovány, se jmenuje *ClassificationTestMetrics*. Tyto metriky jsou pro algoritmy klasifikace stejné a je možné pro všechny algoritmy, které ODM podporuje, vytvořit jednotné zobrazení výsledků.

O prezentaci výsledků se stará výstupní panel, který je obsluhovaný metodou *getOutputPanel()*. Panel, reprezentován instancí třídy *ClassifDTModuleOutputPanel*, se vytvoří otevřením komponenty Report, která je v grafu dolovací úlohy napojena na modul. Tomuto objektu stačí z DMSL předat element Knowledge. Zavoláním metody *readDMSLKnowledge()* se data z DMSL rozdělí na část modelu a část metrik.

5.5.4.1 Zobrazení modelu

Zobrazení modelu zajišťuje metoda *displayBuildData()*, které je potřeba předat element Build elementu Knowledge z DMSL. Pomocí metod *dispayParams()* se zobrazí parametry a *displayAttributes()* zobrazuje atributy, které byly potřeba při vytváření modelu. Model dat je reprezentován rozhodovacím stromem, který je vykreslen na základě volání metody *displayTreeModel()*. Všem těmto metodám je předán odpovídající element z DMSL.

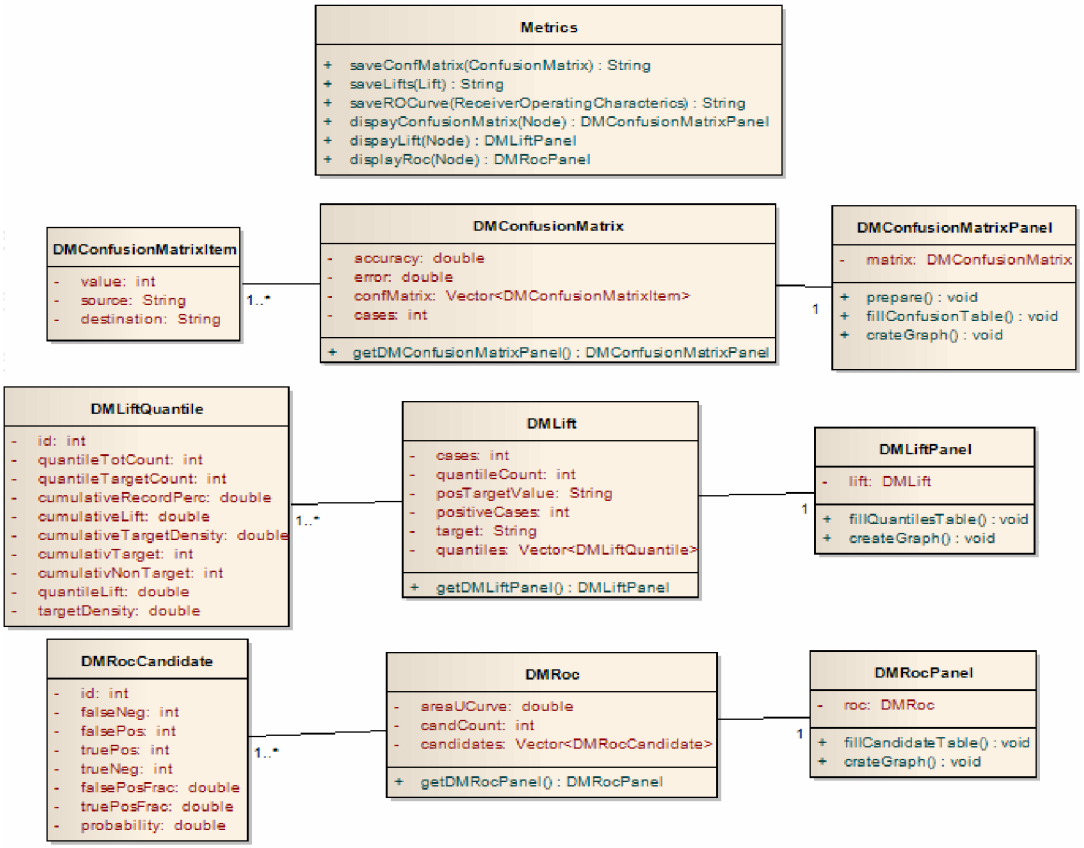


Obr. 5.7: Třídy tvořící zobrazení rozhodovacího stromu

Rozhodovací strom je složen z instancí třídy *DTNode*, které jsou propojeny do stromu. Požadované pravidlo vznikne spojením levé a pravé strany pravidla. Levá strana vznikne voláním metody *getRule()*, která poskládá atributy *predicate* od kořene stromu k uzlu (nad kterým je volána) a spojí je logickým operátorem AND. Pravou stranu pravidla tvoří atribut *consequence*.

5.5.4.2 Zobrazení testovacích metrik

Protože testovací metriky jsou v ODM pro všechny metody klasifikace stejné, byl navržen a implementován balík `cz.vutbr.fit.dataminer.classifdtmodule.metrics`. Tento balík obsahuje třídu *Metrics*, která slouží pro uložení vydolovaných dat do DMSL dokumentu a načtení data z DMSL. Metody třídy *Metrics* jsou implementovány jako statické, není tedy nutné aby byla vytvořena instance této třídy. Pokud je tato třída použita pro uložení vydolovaných metrik nebo jsou data metrik uložena ve formátu popsaném v 5.4.2, je možné je pomocí metod *display* příslušné metriky zobrazit ve výstupním panelu. Metodám je potřeba předat uzel z DMSL, který reprezentuju příslušnou metriku.



Obr. 5.8: Třídy pro zobrazení metrik klasifikace

Výstupem těchto tříd jsou panely JPanel standardní knihovny Swing, ve kterých jsou umístěny standardní komponenty. Pro vizualizaci metrik je použito 2D a 3D grafů. Tento postup byl zvolen z důvodu větší přehlednosti a názornosti. Protože původní aplikace s grafy nepočítala, bylo potřeba přidat do systému knihovny pro jejich vykreslování. Kreslení 2D grafů je zajišťováno knihovnou *jfreechart-1.0.12.jar* [18]. Tato knihovna byla vybrána, protože již byla použita panem Šebkem, který pracuje na úpravách jádra aplikace a tudíž byla již součástí aplikace. Dalším důvodem pro použití je snadná manipulace a některé implicitní funkce pro uživatele (např. zvětšování označené oblasti). Zdrojové kódy knihovny jsou licencovány jako open-source, avšak oficiální dokumentace je zpoplatněna. Grafy této knihovny byly použity pro vizualizaci metriky Lift a vykreslení ROC křivky.

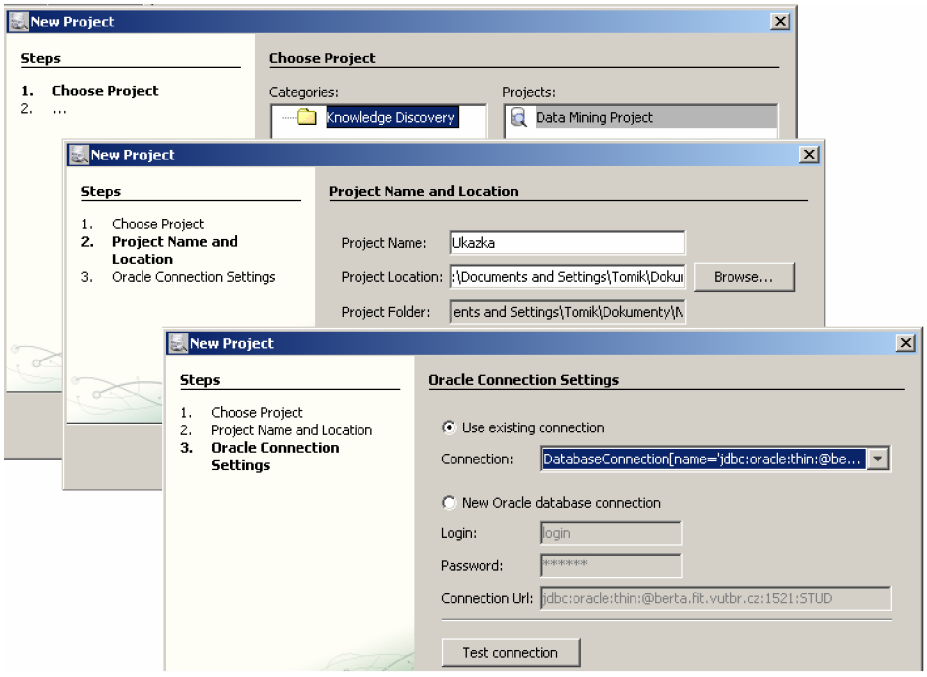
Pokud bychom chtěli převést do grafické podoby tabulku přesnosti a využili k tomu 2D grafu, bylo by toto řešení značně nepřehledné a matoucí. Z tohoto důvodu byl pro větší názornost použit 3D graf. Aplikace neobsahovala žádnou knihovnu pro vykreslování 3D grafu, a proto bylo nutné ji do systému přidat. Vzhledem k tomu, že výběr knihoven pro 3D grafy v jazyce Java je značně omezený, byla vybrána knihovna *jmathplot.jar* [13], která nejlépe splnila kritéria výběru (jednoduchost začlenění, přehlednost grafu, modifikovatelnost nastavení). Jako velice výhodná vlastnost se jeví možnost rotace celým grafem v prostoru. Použití je podrobně popsáno v dokumentaci . Knihovna je volně dostupná na internetu včetně dokumentace a krátkého tutoriálu [13].

6 Ukázka GUI aplikace

Tato kapitola ukazuje, jak vypadá celá aplikace a jak se vytváří nová dolovací úloha. Zaměříme se však na komponentu klasifikace pomocí rozhodovacího stromu, kterou se zabývá tato práce, a komponentu Report, která zobrazuje výsledek. Ukáže také některé komponenty, které nejsou vytvořeny na základě této práce, ale jsou nezbytné pro správný chod modulu. Pro ukázku jsou vybrána data z projektu *STULONG* [15]. (Jedná se o studii rizikových faktorů arterosklerózy u mužů středního věku. Data jsou organizována v jedné tabulce, která obsahuje 64 sloupců a zhruba 1400 řádků. Predikovaný sloupec *CHOLRISK* značí riziko vzniku arterosklerózy a nabývá hodnot 0,1,6. Tato databáze je určena k demonstraci možnosti aplikace metod získávání znalostí z databází v medicínské oblasti.)

6.1 Vytvoření nového projektu

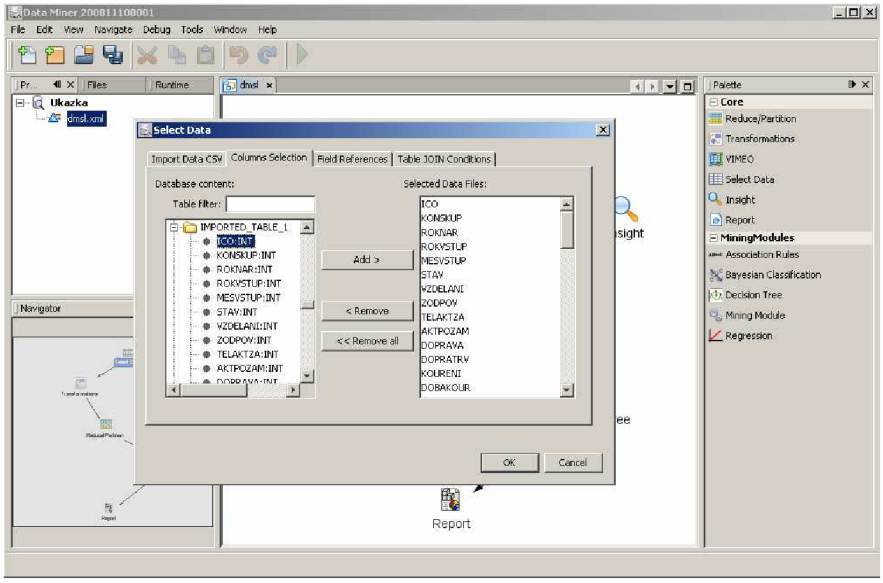
Po spuštění aplikace je potřeba vytvořit novou dolovací úlohu. Postup ilustruje obrázek 6.1. Tuto volbu můžeme provést v menu *File*, kde zvolíme položku *New Project* nebo stisknutím druhého tlačítka na panelu nástrojů, v sekci *File*. Otevře se okno s průvodcem vytvoření nového projektu. V prvním kroku vybereme v okénku *Categories* možnost *Knowledge Discovery* a v okénku *Project* možnost *Data Mining Project*. Další krok nás vyzve k zadání jména projektu a nastavení cesty jeho umístění. Poslední krok je věnován nastavení parametrů pro připojení k databázi. Toto nastavení je pro tento projekt uloženo a nebude muset být zadáváno při příštím otevření projektu. Navíc je uloženo i do aplikace a pokud budeme potřebovat nový projekt se stejným připojením, tak jej vybereme z nabídky existujících připojení. Stisknutím tlačítka dokončit je nový projekt přidán do průzkumníka projektů v levé horní části. Otevřeme projekt a v průzkumníku se objeví soubor *dmsl.xml*. Po otevření se zobrazí pracovní plocha a paleta s komponentami. Tím máme vytvořen nový prázdný projekt.



Obr. 6.1: Postup vytváření nového projektu

6.2 Vytvoření dolovací úlohy

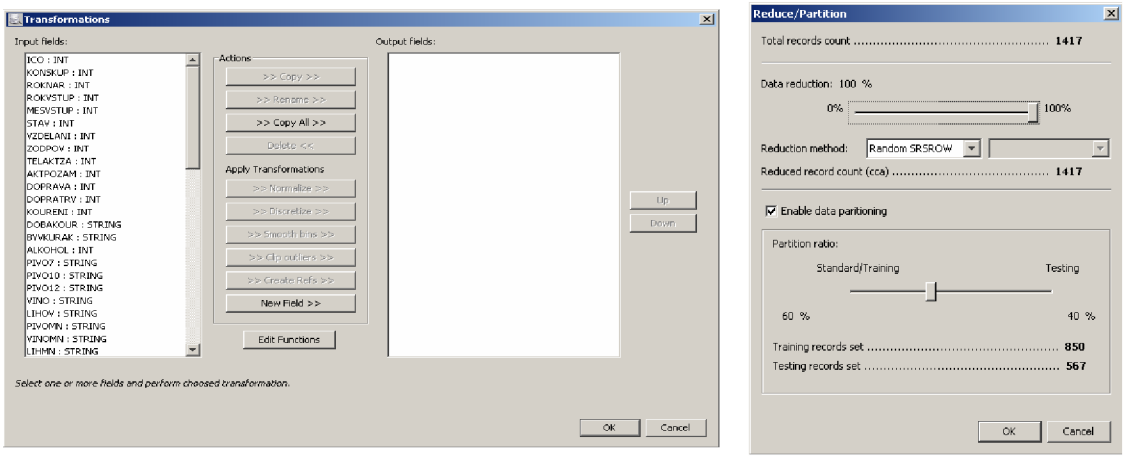
Dolovací úloha je v aplikaci znázorněna jako graf v pracovní ploše, složený z komponent a propojený šipkami. Komponenty je možné přidat pouhým přetažením z palety na plochu. Prvním prvkem, který je potřeba do pracovní plochy přidat, je *Select Data*. Zde je potřeba vybrat data nad kterými bude dolování probíhat. Pro naši ukázku zvolíme v záložce *Columns Selection* tabulku *DT_UKAZKA* a pomocí tlačítka *Add* přidáme všechny sloupce do dolovací úlohy (Obr 6.2).



Obr. 6.2: Vybrání dat v *Select Data*

Aby se proces výběru dokončil, je potřeba vyvolat kontextové menu nad komponentou a v něm zvolit položku *run*. Další důležitou komponentou jsou *Transformations* (Obr 6.3). Po vložení do pracovní plochy je nezbytné propojit *Select Data* s *Transformations* pomocí spojovací hrany (Propojovací hrana se vytváří stisknutím Ctrl + levé tlačítko myši a tažením od zdroje k cíli). Pro fungování našeho modulu není nezbytné tuto komponentu přidávat, je důležitá pokud potřebujeme upravit, odvodit nové nebo vynechat některé atributy vstupní tabulky. V našem případě zvolíme tlačítko *Copy All*, kterým zvolíme všechny atributy.

a) **b)**

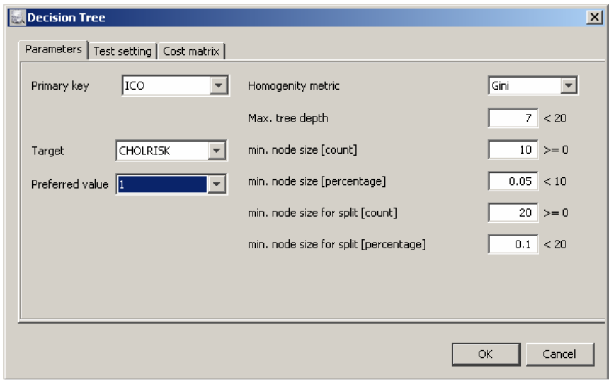


Obr. 6.3: Komponenta **a)** *Transformations*, **b)** *Reduce/Partition*

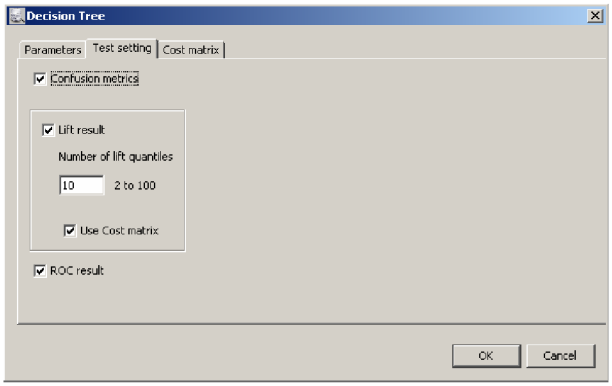
Komponentu, kterou je nutně potřeba připojit v případě klasifikačních a prediktivních úloh, je *Reduce/Partition*. Tato komponenta nebyla součástí původního systému a byla přidána jako součást úprav jádra panem Šebkem. Umožňuje redukovat množství dat, které jsou předávány do dolovacího modulu. Hlavním smyslem však bylo rozdělit data na množinu trénovací a množinu testovací. Jedná se o velice jednoduchou komponentu, kde je po zvolení *Enable data partitioning* možno nastavit poměr velikosti trénovací ku testovací množině. Pokud by tato komponenta nebyla vložena před klasifikačním nebo regresním dolovacím modulem, pak by tomuto modulu nebyla předána žádná testovací data. Modul by považoval všechna vstupní data jako trénovací.

Následující připojenou komponentou je dolovací modul, tedy v našem případě *Decision Tree*. Po otevření se zobrazí okno se třemi záložkami pro zadávání parametrů. V záložce *Parameters* (Obr. 6.4 a)) je nutno nastavit pomocí výběru z roletkových menu primární klíč tabulky, atribut, který chceme klasifikovat, a jeho preferovanou hodnotu. V pravé části je nastavení parametrů modelu rozhodovacího stromu (dělící kriterium, maximální hloubka, …). Pro náš příklad nastavíme jako primární klíč hodnotu *ICO*, cílový atribut na *CHOLRISK* a preferovanou hodnotu na *1*. Ostatní parametry ponecháme na výchozích hodnotách. Záložka *Test setting* obsahuje nastavení testovacích metrik (Obr. 6.4 b)). Vybereme všechny možnosti a počet quantilů v Liftu ponecháme na hodnotě *10*. Poslední záložka (*Cost matrix*) obsahuje cenovou matici příslušného cílového atributu a možnosti jejího nastavení (Obr. 6.4 c)). Nastavení ponecháme na výchozích hodnotách. Tato záložka není v počátečním nastavení přístupná a je potřeba ji povolit v nastavování testovacích parametrů v záložce *Test setting*.

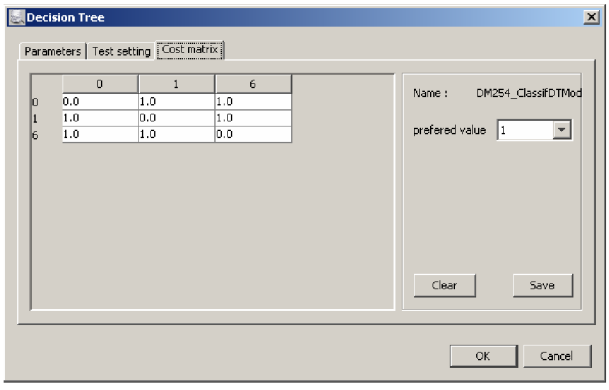
a)



b)



c)



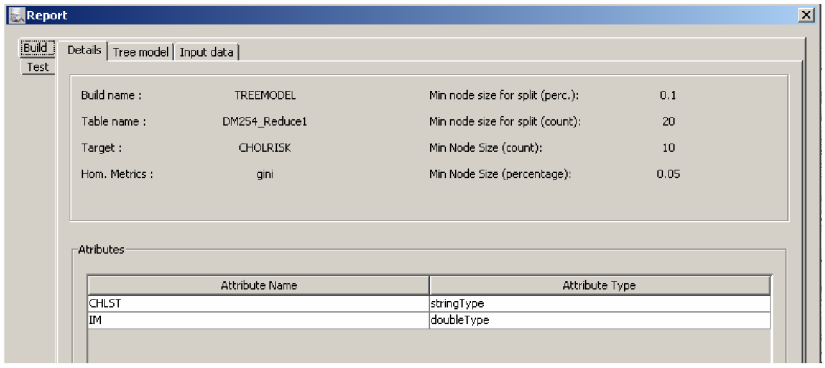
Obr. 6.4: Panely komponenty Decision Tree

Poslední je komponenta *Report*, která slouží pro zobrazování výsledků dolování. Zobrazuje výsledné znalosti té komponenty, která je k ní připojena hranou. V našem případě zobrazí dvě záložky (*Build* a *Test*).

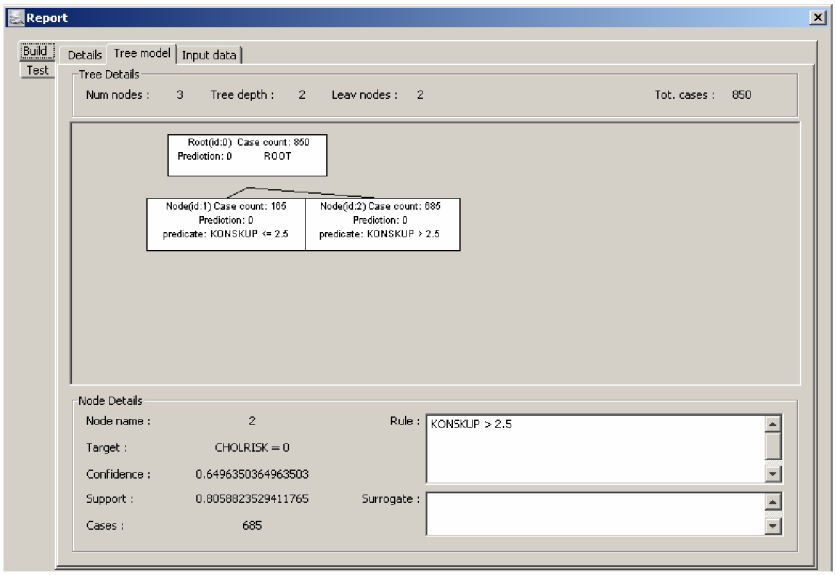
Build reprezentuje vytvořený a naučený model rozhodovacího stromu. Je to vlastní výsledek modulu klasifikace. Je tvořena třemi záložkami:

- *Details* – zobrazuje v horní části jméno trénovací tabulky, cílový atribut, parametry dolovací úlohy a další hodnoty, které definují proces učení. Ve spodní části je zobrazena tabulka s atributy, které byly použity při tvorbě rozhodovacího stromu (Obr. 6.5 a)).
- *Tree model* – ukazuje model rozhodovacího stromu. V horní části jsou zobrazeny parametry stromu. Pokud myší stiskneme levé tlačítko na některém uzlu stromu, zobrazí se ve spodní části panelu dostupné informace o tomto uzlu (Obr. 6.5 b)).
- *Input Data* – tabulka obsahující všechny vstupní data (tj. řádky a sloupce), která byla využita při procesu učení modelu.

a)



b)



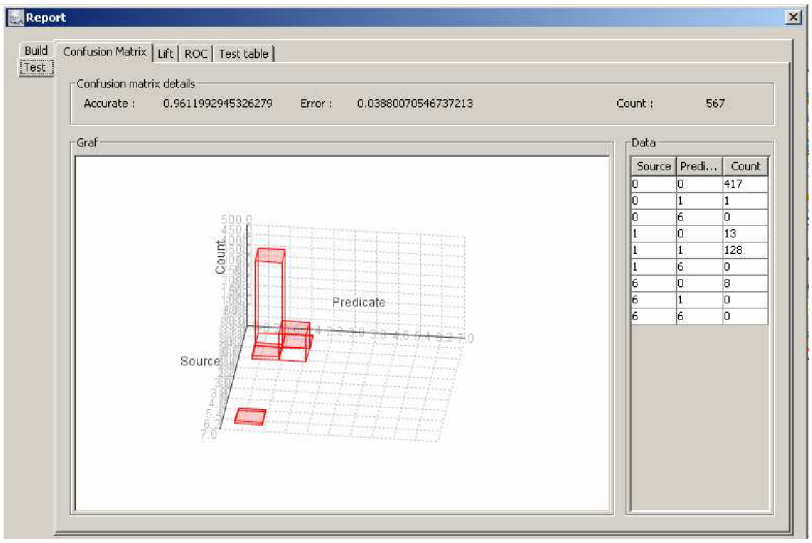
Obr. 6.5: Části *Build* komponenty *Report* pro *Decision Tree*

Test zobrazuje testovací metriky klasifikačního modelu. Tato část komponenty *Report* obsahuje čtveřici záložek, kde první tři reprezentují jednotlivé klasifikační metriky, které jsou popsány v kapitole 4.1.1.1.

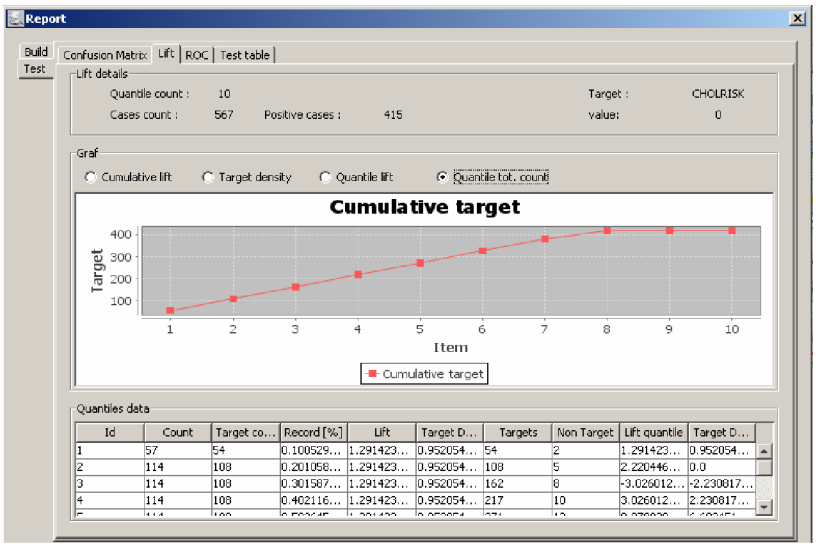
- *Confusion matrix* – zobrazuje detaily tabulky přesností. V levé části je zobrazena v podobě 3D grafu a v pravé části tatáž informace v podobě tabulky. Grafem lze otáčet okolo všech os prostoru (Obr. 6.6 a)).
- *Lift* – v horní části panelu jsou celkové detaily metriky. Prostřední část umožňuje zobrazit několik grafů, které dávají do souvislostí některé významné hodnoty. Spodní část zaujímá tabulka se všemi hodnotami, které byly získány při vytváření metriky (Obr. 6.6 b)).
- *ROC* – centrální část zde zaujímá graf ROC křivky, vpravo jsou umístěny detaily křivky a ve spodní části je zobrazena tabulka s hodnotami (Obr. 6.6 c)).

- *Test data* - tabulka obsahující všechny vstupní data, která byla využita při procesu testování modelu.

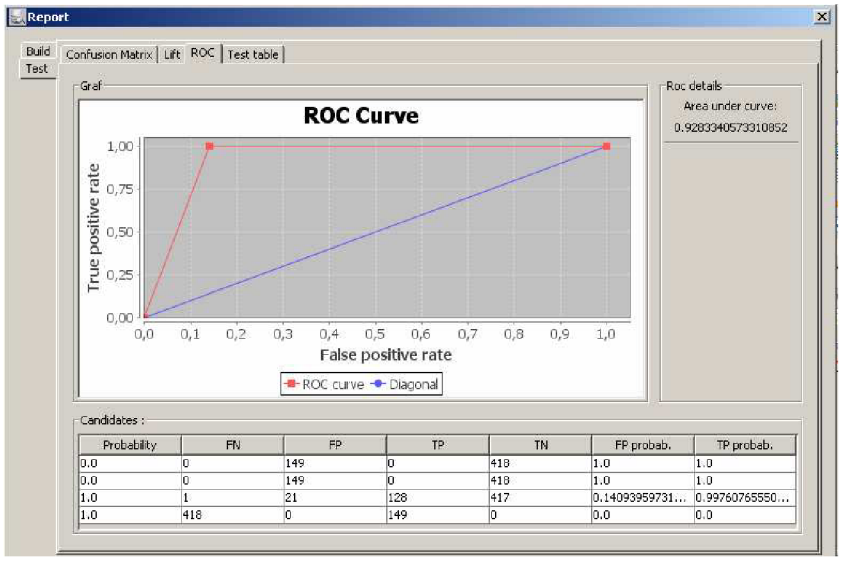
a)



b)



c)



Obr. 6.6: Část *Test* komponenty *Report pro* Decision Tree

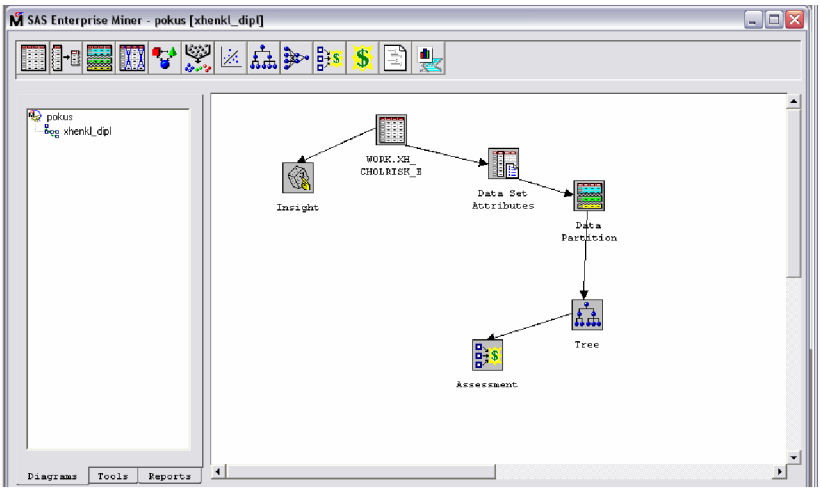
7 Porovnání systému

V současné době se na FIT VUT pro výukové účely používá systém SAS Enterprise Miner (SAS EM) od společnosti SAS. Tento systém není fakultě poskytován bezplatně, a proto vznikla myšlenka tento systém nahradit jiným, vytvořený v ráci diplomových a bakalářských prací, který by SAS EM pro potřeby výuky mohl nahradit. Pro tyto účely je vyvíjen systém Data Miner (DM). Ten zatím zdaleka neposkytuje možnosti produktu společnosti SAS, ale již lze porovnat některé vlastnosti nebo moduly.

Tato kapitola se zabývá právě tímto srovnáním, které by ukázalo do jaké míry se daří vyvíjený systém přibližovat profesionálnímu softwaru. Bude zde porovnán dolovací systém jako celek a implementovaný modul klasifikace pomocí rozhodovacího stromu především, co se týče možností nastavení, jednoduchosti a intuitivnosti ovládání, vizualizace výsledků a přesnosti. Bude provedeno testování na datech projektu *STULONG*. Z důvodu, že tato data byla využita v předchozí kapitole, kde jsou i ilustrovány obrázky z DM, tak zde již nejsou znovu tyto obrázky zobrazeny.

SAS Enterprise Miner

Specifikace dolovací úlohy je velice podobná našemu systému, pomocí myši pouhým přetáhnutím komponent z palety do pracovní plochy. Propojení hranami, které reprezentují tok dat, je také velice jednoduché, použitím myši směrem od zdroje k cíli. Ukázka grafického rozhraní je na obrázku 7.1. Tím ale skoro veškerá podobnost končí. Jednotlivé komponenty jsou mnohem složitější a méně přehledné. To je s největší pravděpodobností způsobeno velkými možnostmi tohoto systému, které jistě sahají za možnosti naší aplikace Data Miner, to ale činí systém pro uživatele na první pohled značně nepřehledný. Potvrzování voleb v komponentách se děje pomocí zavření okna, kde se vás poté aplikace zeptá, zdali chcete změny uložit či nikoliv. Takřka žádná tlačítka ve stylu „OK“, pro potvrzení nebo „Cancel“, pro zrušení. Mnoho nastavování je přes kontextová menu, která jsou někdy velice obsáhlá a nepřehledná.

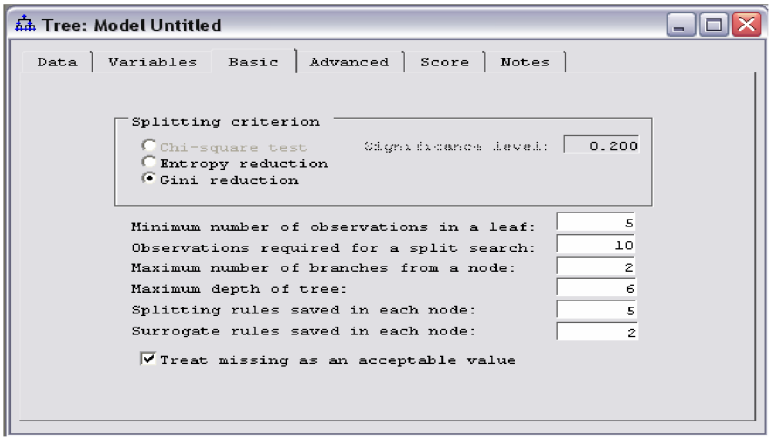


Obr. 7.1: GUI SAS Enterprise Miner

Většina těchto vyjmenovaných nedostatků by byla jistě odstraněna po přečtení manuálu systému, to ale není podle mého názoru účelem aplikace, která je pro studenty prvním seznámením s problematikou získávání znalostí z databází.

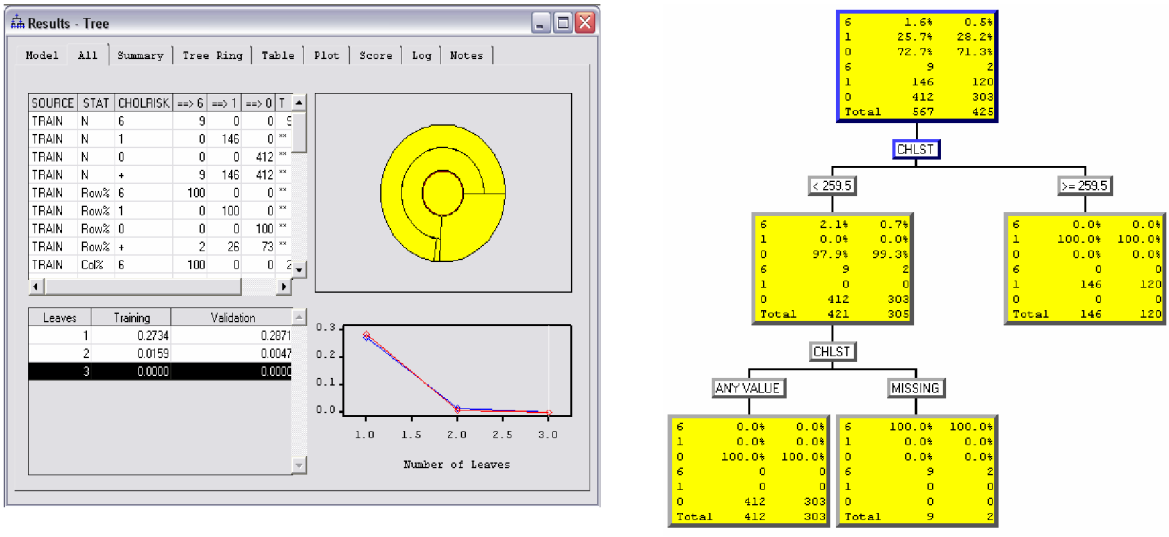
7.1 Porovnání

Porovnání dolovacích modulů bylo provedeno na stejných datech z projektu *STULONG*. Na obrázku 7.2 je ukázka nastavení parametrů. Obě tyto nabídky mají možnost nastavení dvou dělících kritérií – Gini a Entropy, maximální hloubky stromu a minimální množství prvků v uzlu. Ovšem nastavování primárního klíče relace a cílového atributu jsou v SAS EM součásti jiné (předcházející) komponenty, pravděpodobně z důvodu očekávání více dolovacích modulů zapojených do jedné úlohy. Nastavování testovacích metrik neprobíhá vůbec, protože jsou všechny vytvářeny automaticky.



Obr. 7.2: Nastavení parametrů v SAS EM

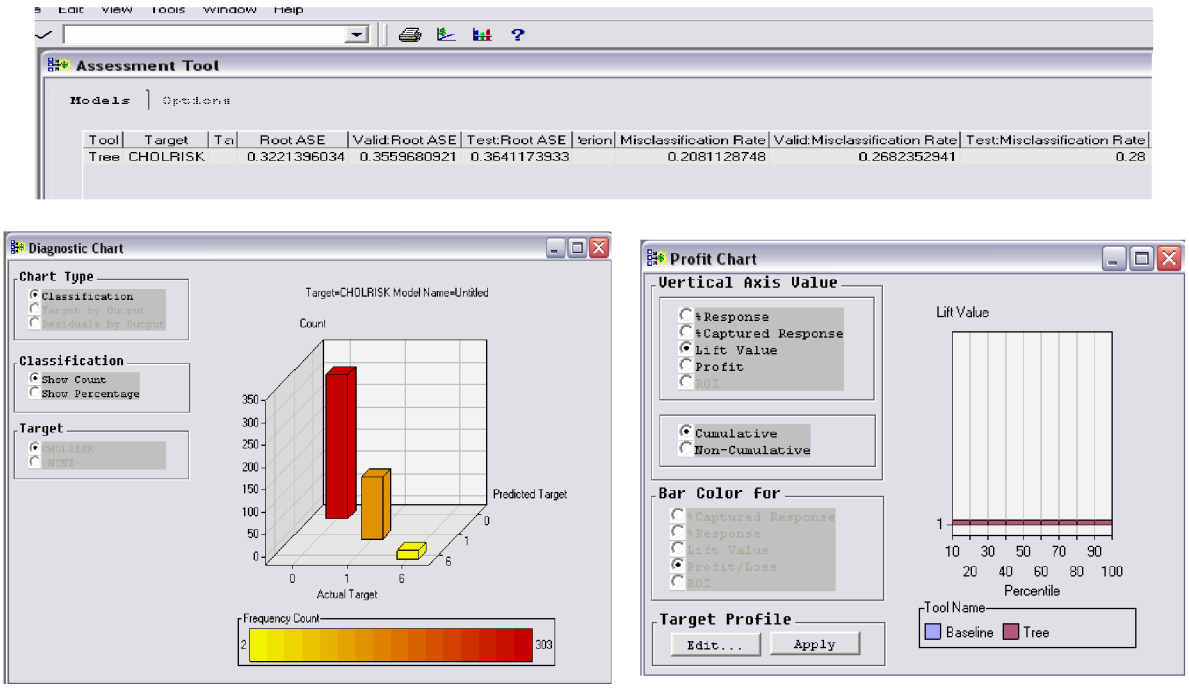
Spuštění probíhá u obou aplikací stejně, přes kontextové menu nad komponentou, kde se zvolí položka *run*. Zobrazení výsledků je u SAS EM prováděno na dotaz ihned po provedení úlohy nebo přes kontextové menu. Okno výsledků obsahuje řadu informací, několik grafů a řadu záložek (Obr. 7.3). Zajímavou informací je graf v pravé dolní části, který zobrazuje přírůstek přesnosti s rostoucím počtem listů. Rozhodovací strom lze zobrazit až přes dvouúrovňové kontextové menu, přes možnost View -> Tree.



Obr. 7.3: Zobrazení výsledku dolování a rozhodovací strom

Rozhodovací strom je u DM součástí komponenty Report, společně s testovacími metrikami. Pro zobrazení testovacích metrik je v SAS EM potřeba napojit na dolovací modul komponentu

Assessment, která zobrazuje výsledné přesnosti, respektive chybovosti modelu. Pro zobrazení metrik je potřeba otevřít panel komponenty a poté se v horní liště celé aplikace objeví dvě ikonky s obrázky grafů, kde první slouží pro Lift metriky a druhá pro tabulku přesnosti. Pro zobrazení tabulky přesnosti slouží 3D graf. Pro vizualizaci Lift metrik slouží 2D grafy, které nejsou počítány nad kvantily, ale procentily. Tato komponenta slouží spíše pro porovnávání několika dolovacích modulů. Taková komponenta systému DM chybí.



Obr. 7.4: Vizualizace metrik v aplikaci SAS EM

7.2 Zhodnocení

Aplikace SAS EM je velice rozsáhlá a co se týče možností nastavení, je jistě neporovnatelná s DM. Má mnohem více modulů pro dolování, více komponent pro úpravu dat před dolováním i více komponent pro zpracování výsledků. Její největší nevýhodou, oproti DM, je její složité ovládání. Velké množství nastavování přes kontextová menu dělají aplikaci poměrně složitou na pamatování si, kde co nastavit a hlavně jak.

Co se týče dolování pomocí modulu klasifikace rozhodovacího stromu jsou, podle mého názoru, tyto aplikace srovnatelné. Protože toto dělení neprobíhá pokaždé stejně a rovnoměrně, mohou být výsledky po každém dělení trošku jiné. Zvláště pak pokud je některá třída cílového atributu málo častá, může se stát, že se v trénovacích datech vůbec nebude vyskytovat a tím je zásadně ovlivněn výsledný model. Menší nevýhodou DM je závislost na podpoře modulu OMD a informacích, které jsou poslány z Oraclu zpět do aplikace. Především by pak model stromu mohl obsahovat více informací o jednotlivých uzlech.

Hlavní výhodou SAS EM oproti DM je, že je vytvořena i pro porovnávání několika dolovacích metod, což reprezentuje komponenta *Assessment*. Komponenta podobného zaměření aplikaci DM chybí. Je to z důvodu, že se u původní aplikace nepředpokládalo spuštění více dolovacích metod pro jedna data.

8 Návrh úprav a změn systému

Komponenta Report a Multiple Report

Aplikace neumožňuje vícenásobné vložení komponenty *Report*. Tzn., že v případě kdybychom měli vloženo více dolovacích modulů v jedné úloze, tak nemůžeme zobrazit výsledky obou komponent a porovnat jejich výsledky. Musíme zrušit hranu od jednoho modulu a *Report* přepojit k druhému.

Multiple Report je komponenta, která by umožňovala zobrazení výsledků několika dolovacích modulů najednou nebo alespoň jejích podobných částí nebo metrik. Současná aplikace toto neumožňuje.

Příprava dat

Měla by být vytvořena komponenta, která by umožňovala automatizované hledání atributů, mezi kterými je silná korelace. Tento výběr je v současné aplikaci ponechána na uživateli a ten nemusí mít vždy schopnost takovéto atributy odhalit.

Vytvořit v jádře reakce na stisk kláves

Po označení komponenty nebo hrany je možné je smazat pouze otevřením kontextového menu a zvolením položky *remove*. Lepší, pohodlnější a intuitivnější by bylo pro tyto případy zvolit reakci na stisk klávesy *Delete*.

Zobrazení výjimek

Jádro aplikace ve stávajícím řešení neobsahuje mechanizmus pro zobrazení chyb při spuštění běhu komponenty. Pokud se při spuštění komponenty vyskytne chyba, která je zachycena výjimkou, nelze uživateli vypsat zprávu o této chybě.

Další moduly

Rozšíření aplikace o další moduly.

9 Závěr

Cílem práce bylo implementovat dolovací modul pro dolování z dat metodou klasifikace pomocí rozhodovacího stromu. Tento modul poté začlenit do aplikace Data Miner, která se vyvíjí na FIT VUT. K tomuto účelu bylo potřeba nastudovat práci Ing. Krásného [3], který zásadním způsobem přestavěl celé jádro systému, a práci Ing. Madera [8], který jako první implementoval modul pro dolování z dat do aplikace. Aplikace Data Miner pracuje s databázovým systémem firmy Oracle, který nabízí podporu pro dolování v podobě modulu Oracle Data Mining. Bylo tedy nezbytnou součástí nastudovat práci s ODM a problematiku dolování z dat pomocí klasifikační metody rozhodovacího stromu.

V rámci projektu byl implementován nový modul pro dolování. Modul využívá podporu Oracle Data Minig a je tedy na této podpoře závislý. ODM podporuje pro dolování pomocí rozhodovacího stromu využít metody Gini a Entropy. Obě tyto metody jsou uživateli poskytnuty pro využití. Výhodou algoritmu rozhodovacího stromu je jeho relativně vysoká rychlost, dobrá přesnost a dobrá stabilita. Jeho výstupem jsou rozhodovací pravidla, která jsou dobře čitelná i pro člověka bez speciálních znalostí. Aplikace vizualizuje naučený model dat jako binární strom, kde listové uzly reprezentují výsledná pravidla.

Pro potřeby projektu byl také nastudován jazyk DMSL, který slouží k ukládání definice dolovací úlohy i pro uskladnění vydolovaných znalostí. Tento jazyk byl doplněn o nové definice elementů DataMiningTask, který uchovává informace pro specifikaci dolovací úlohy, a element Knowledge, který slouží pro uložení výsledných znalostí.

Nedílnou součástí klasifikačních metod jsou testovací metriky, které uživateli vizualizují kvalit vytvořeného modelu. Na základě studia ukázkových příkladů bylo zjištěno, že výstup těchto testovacích metrik, je pro všechny metody klasifikace podporované ODM stejný. Proto byla rámci projektu snaha vytvořit pro tyto metody společný zobrazovací aparát. Tím by měl být implementovaný balík tříd Metrics. Jedná se především o vizualizaci pomocí 2D a 3D grafů.

V době tvorby této práce byla prováděna rozsáhlá úprava jádra v podobě práce pana Šebka, který přepracoval některé komponenty dolovacího procesu a vytvořil novou komponentu potřebnou pro moduly klasifikace. Aby byl modul použitelný v současné aplikaci, bylo nutné jej začlenit do nově upraveného jádra.

Aplikace Data Miner by mohla v budoucnu nahradit systém SAS Enterprise Miner od společnosti SAS, který je komerční a na FIT VUT se používá pro výukové účely v oblasti získávání znalostí z databází. Proto se tato práce krátce zabývá i porovnáním těchto dvou systémů, především pak srovnáním nového modulu s podobným modulem v komerční aplikaci.

Literatura

[1] Zendulka, J., Bartík, V., Lukáš, R., Rudolfová, I.: Získávání znalostí z databází. Studijní opora, 2006, Fakulta informačních technologií VUT v Brně.

[2] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Second Edition. Elsevier Inc., 2006.

[3] Krásný M. : Systém pro dolování dat v prostředí oracle, diplomová práce, Brno, 2008, Vysoké učení technické v Brně, Fakulta informačních technologií.

[4] Building Classification Models: ID3 and C4.5, 2.1.2009, [online], URL: http://www.cis.temple.edu/~ingargio/cis587/readings/id3-c45.html

[5] Kotásek P.: DMSL: The Data Mining Specification Language, Disertační práce, 2003. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=7348

[6] NetBeans, Wikipedia, [online], URL: http://www.netbeans.org/features/index.html

[7] Margaret T. aj.: Oracle Data Mining Concepts, 10.g Release 2 (10.2), 2005 URL: http://download.oracle.com/docs/pdf/B14339_01.pdf

[8] Mader P. : Dolovací moduly systému pro dolování z dat v prostředí Oracle, diplomová práce, Brno, 2009, Vysoké učení technické v Brně, Fakulta informačních technologií.

[9] Gálet M. : Grafická nadstavba pro systém získávání znalostí, diplomová práce, Brno, 2007, Vysoké učení technické v Brně, Fakulta informačních technologií.

[10] Oracle Data Mining, Wikipedia, [online], URL:http://en.wikipedia.org/wiki/Oracle_Data_Mining

[11] Oracle ® Data Mining Application Developer's Guide, 10g Release 2 (10.2), Oracle, 2005. URL: http://download.oracle.com/docs/html/B14340_01/toc.htm

[12] Oracle ® Data Mining Concepts, 11g Release 1 (11.1), Oracle, 2008, URL: http://download.oracle.com/docs/pdf/B14339_01.pdf

[13] JMathPlot, JmathTools, [online], 2009, URL: http://jmathtools.berlios.de/doku.php?id=start

[14] JSR-73 Java Documentation, Oracle, [online], URL: http://www.oracle.com/technology/products/bi/odm/jsr-73/index.html

[15] EuroMISE, Projekt STULONG, [online], 2003, URL: http://euromise.vse.cz/stulong/index.php

[16] Data Mining, DNA, 2007, [online], URL: http://www.dna-consult.com/client/dm.asp

[17] Oracle Database 10g Release 2, Oracle, 2005, URL: http://www.oracle.com/technology/software/index.html

[18] JFreeChart, JFree, [oline], 2009, URL: http://www.jfree.org/jfreechart/download.html

Seznam příloh

Příloha A: DDL specifikace úprav DMSL

Element dataMiningTask

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

<!ELEMENT DecisionTree (UseMatrix, InputDataType, Parametres, CostMatrix,
ConfMatrix, Lift, ROC)>
<!ATTLIST DecisionTree  algorithm (Gini|Entropy) #REQUIRED>

<!ELEMENT UseMatrix>
<!ATTLIST UseMatrix     testMatrix CDATA #REQUIRED>
                        trainMatrix CDATA #REQUIRED>

<!ELEMENT InputDataType>
<!ATTLIST InputDataType       tableKey CDATA #REQUIRED>
                              prefVal CDATA #REQUIRED>
                              target CDATA #REQUIRED>

<!ELEMENT Parameters (MaxDepth, MinNodeSizeP, MinNodeSizeC, MinSplitRecP,
MinSplitRecC)>

<!ELEMENT MaxDepth (#PCDATA)>

<!ELEMENT MinNodeSizeP (#PCDATA)>

<!ELEMENT MinNodeSizeC (#PCDATA)>

<!ELEMENT MinSplitRecP (#PCDATA)>

<!ELEMENT MinSplitRecC (#PCDATA)>

<!ELEMENT CostMatrix (Rows*)>
<!ATTLIST CostMatrix          name CDATA #REQUIRED>
                              type (INT|STRING|DATE|REAL) #IMPLIED>

<!ELEMENT Rows (Value+)>
<!ATTLIST Rows          value CDATA #REQUIRED>

<!ELEMENT Value (#PCDATA)>
<!ATTLIST Value         source CDATA #REQUIRED>
                        target CDATA #REQUIRED>

<!ELEMENT ConfMatrix>
<!ATTLIST ConfMatrix          use (true|false) #REQUIRED>

<!ELEMENT ROC>
<!ATTLIST ROC           use (true|false) #REQUIRED>

<!ELEMENT Lift (Count?)>
<!ATTLIST Lift          use (true|false) #REQUIRED>

<!ELEMENT Count (#PCDATA)>
```

Element Knowledge

```
<!ELEMENT Knowledge ANY >
<!ATTLIST Knowledge           name              CDATA #REQUIRED
                              Language          CDATA #REQUIRED
                              languageVersion   CDATA #IMPLIED
                              type              CDATA #IMPLIED >
```

```
<!ELEMENT Build (Parameters, Attributes, TreeModel)>
<!ATTLIST Build         buildTable CDATA #REQUIRED>
                        name CDATA #REQUIRED>
                        date CDATA #IMPLIED>
                        target CDATA #REQUIRED>

<!ELEMENT Attributes (Attribute+)>

<!ELEMENT Attribute>
<!ATTLIST Attribute           name CDATA #REQUIRED>
                              type CDATA #REQUIRED>

<!ELEMENT TreeModel (Nodes+)>
<!ATTLIST TreeModel     leavNode NUMBERS #REQUIRED>
                        numNodes NUMBERS #REQUIRED>
                        treeDepth CDATA #REQUIRED>

<!ELEMENT Nodes (Root, Node+)>

<!ELEMENT Root >
<!ATTLIST Root    caseCount INT-NUMBER #REQUIRED>
                  id CDATA #REQUIRED>
                  prediction CDATA #REQUIRED>

<!ELEMENT Node (Predicate, Surrogate*, Rule?)>
<!ATTLIST Node    caseCount NUMBERS #REQUIRED>
                  id CDATA #REQUIRED>
                  level NUMBERS #REQUIRED>
                  chilCount NUMBER #REQUIRED>
                  parent CDATA #REQUIRED>
                  prediction CDATA #REQUIRED>

<!ELEMENT Predicate>
<!ATTLIST Predicate     flag(lessOrEqual|grtrOrEqual|greater|less|equal)
                        #REQUIRED>
                        name CDATA #REQUIRED>
                        value CDATA #REQUIRED>

<!ELEMENT Surrogate>
<!ATTLIST Surrogate     flag(lessOrEqual|grtrOrEqual|greater|less|equal)
                        #REQUIRED>
                        id NUMBERS #REQUIRED>
                        name CDATA #REQUIRED>
                        value CDATA #REQUIRED>

<!ELEMENT Rule (Consequent)>
<!ATTLIST Rule          confidence NUMBERS #REQUIRED>
                        support NUMBERS #REQUIRED>

<!ELEMENT Consequent>
<!ATTLIST Consequent    name CDATA #REQUIRED>
                        value CDATA #REQUIRED>

<!ELEMENT Test (ConfusionMatrix, Lift, Roc)>
<!ATTLIST Test          testTable CDATA #REQUIRED>
                        name CDATA #REQUIRED>

<!ELEMENT ConfusionMatrix (Item+)>
<!ATTLIST ConfusionMatrix     accuracy NUMBERS #REQUIRED>
                              error NUMBERS #REQUIRED>

<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item          actual CDATA #REQUIRED>
                        predicate CDATA #REQUIRED>
```

```
<!ELEMENT Lift (Quantile+)>
<!ATTLIST Lift          cases NUMBERS #REQUIRED>
                        numOfQuantiles NUMBERS #REQUIRED>
                        positiveCases NUMBERS #REQUIRED>
                        positiveTargetValue CDATA #REQUIRED>
                        targetName CDATA #REQUIRED>

<!ELEMENT Quantile>
<!ATTLIST Quantile      cumLift NUMBERS #REQUIRED>
                        cumNonTargets NUMBERS #REQUIRED>
                        cumRecordP NUMBERS #REQUIRED>
                        cumTargetDensity NUMBERS #REQUIRED>
                        cumTargets NUMBERS #REQUIRED>
                        id CDATA #REQUIRED>
                        quantLift NUMBERS #REQUIRED>
                        quantTargetCount NUMBERS #REQUIRED>
                        quantTotalCount NUMBERS #REQUIRED>
                        targetDensity NUMBERS #REQUIRED>

<!ELEMENT Roc (Cadndidate+)>
<!ATTLIST Roc           areaUCurve NUMBERS #REQUIRED>
                        numTresholCand NUMBERS #REQUIRED>

<!ELEMENT Cadndidate >
<!ATTLIST Cadndidate    falseNeg NUMBERS #REQUIRED>
                        falsePos NUMBERS #REQUIRED>
                        falsePosFrac NUMBERS #REQUIRED>
                        id CDATA #REQUIRED>
                        probability NUMBERS #REQUIRED>
                        trueNeg NUMBERS #REQUIRED>
                        truePos NUMBERS #REQUIRED>
                        truePosFrac NUMBERS #REQUIRED>
```

Příloha B: Obsah přiloženého CD

1. workspace – zdrojové kódy celé aplikace i s modulem rozhodovacího stromu
2. dataMiner – spustitelná distribuce aplikace DataMiner
3. documentation – textová část diplomové práce a dokumentace JavaDoc modulu
4. sample – ukázkové příklady
5. readme.txt - popis obsahu CD a ukázkových příkladů
6. dmsl.xml – ukázka DMSL dokumentu