VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

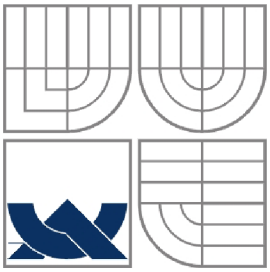
VYTVOŘENÍ NOVÝCH PREDIKČNÍCH MODULŮ V SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

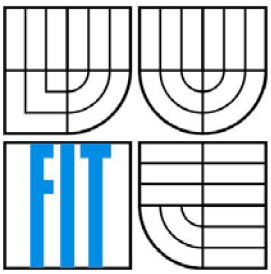
AUTOR PRÁCE
AUTHOR

Bc. DAVID HAVLÍČEK

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

VYTVOŘENÍ NOVÝCH PREDIKČNÍCH MODULŮ V SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

CREATION OF NEW PREDICTION UNITS IN DATA MINING SYSTEM ON NETBEANS PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID HAVLÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ROMAN LUKÁŠ, PhD.

BRNO 2009

Abstrakt

Předmětem této diplomové práce je vytvoření nového predikčního modulu pro již existující systém pro získávání znalostí z databází. První část práce se věnuje obecné problematice získávání znalostí, predikci a predikčním metodám. Druhá část se věnuje systému vyvíjenému na FIT, pro který se modul implementuje, použitým technologiím, návrhu a implementaci samotného dolovacího modulu pro uvedený systém. Řešení je implementováno v jazyce Java a postaveno na platformě NetBeans.

Abstract

The issue of this master's thesis is a creation of new prediction unit for existing system of knowledge discovery in database. The first part of project deal with general problems of knowledge discovery in database and predictive analysis. The second part of the project deal with system developed on FIT, for which is module implemented, used technologies, concept and implementation of mining module for this system. The solution is implemented in Java language and is a built on the NetBeans platform.

Klíčová slova

získávání znalostí, dolování dat, DMSL, Oracle Data Mining, Java, Java API, SVM, Support Vector Machines, Platforma NetBeans, modul, predikce, regrese, lineární regrese, nelineární regrese

Keywords

Knowledge discovery, data mining, DMSL, Oracle Data Mining, Java, Java API, SVM, Support Vector Machines, Netbeans Platform, program unit (module), predictive analysis, regression, linear regression, nonlinear regression

Citace

Havlíček David: Vytvoření nových predikčních modulů v systému pro dolování z dat na platformě NetBeans. diplomová práce, Brno, FIT VUT v Brně, 2009

Vytvoření nových predikčních modulů v systému pro dolování z dat na platformě NetBeans

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Romana Lukáše, PhD.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
David Havlíček
24. 5. 2009

Poděkování

Rád bych poděkoval panu Ing. Romanovi Lukášovi, PhD. za jeho pomoc a podporu při řešení této diplomové práce.

Obsah

1 Úvod

Žijeme v době, ve které patří informace k nejžádanějšímu a také k nejcennějšímu artiklu. V době, která produkuje obrovské množství dat. Mnohdy si to ani neuvědomujeme, ale s trochou nadsázky lze tvrdit, že každý náš krok produkuje nějaká data. Každý náš telefonát, zpráva, pohyb na internetu, nákup v obchodě, návštěva lékaře, koupě dovolené, registrace do klubu a další produkují nějaké informace - data. Existují oblasti, především obchodní sféra a organizace v ní, pro které jsou tyto data neocenitelnou položkou. Tyto informace jim slouží pro automatizaci jejich operativního zpracování. Tím tyto organizace získávají určitý potenciál, který pak dovedou v určité podobě zúročit, stejně tak jako každý inteligentní jedinec, který na základě svých zkušeností a informací je schopen se nějak rozhodovat a reagovat. Otázkou zůstává, jakým způsobem lze automatizovaně využít tato obrovská data.

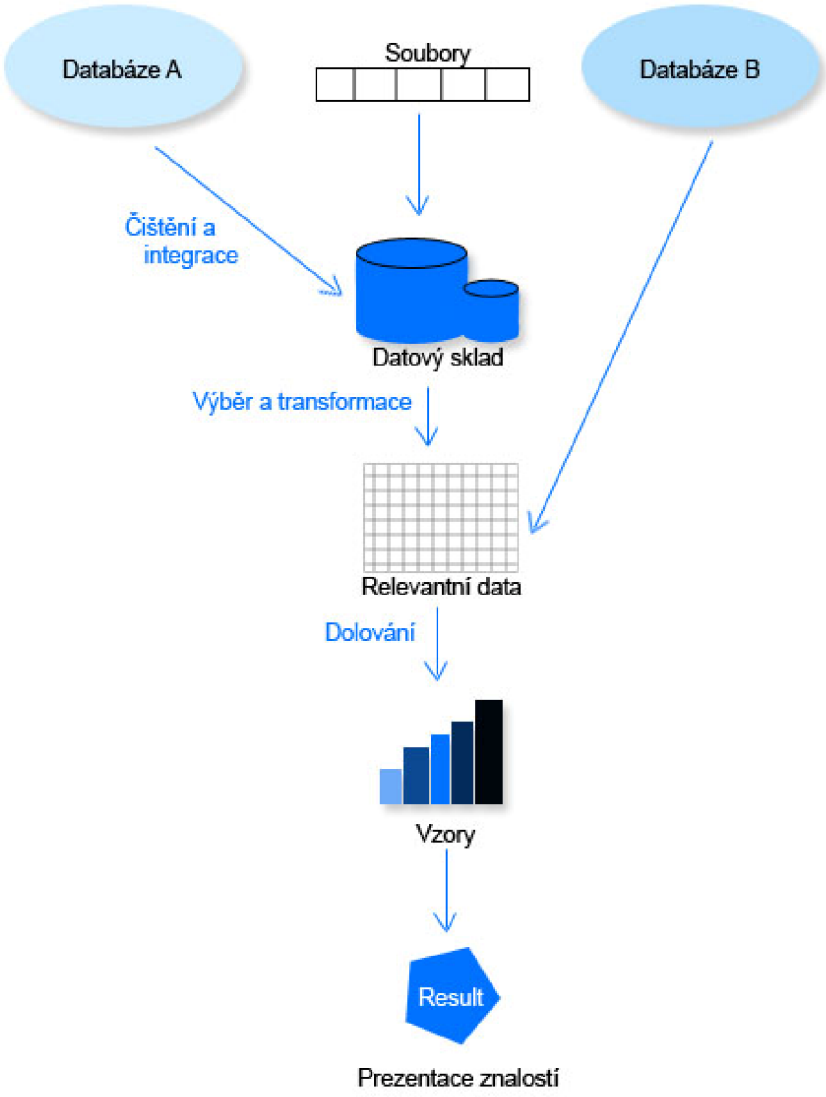
Jednou z cest, jak z údajů získat to důležité, je technologie vytváření datových skladů (*data warehousing*) a aplikace různých dotazovacích a analytických nástrojů OLAP (*Online Analytical Processing*), schopných prezentovat různé agregované údaje v co nejnázornější podobě. Zatímco běžné OLAP nástroje jsou určeny pro interakční datovou analýzu, postupně se dostávala stále více do popředí snaha o poskytnutí technik, metod a nástrojů, které by analýzu do určité míry automatizovaly. Výsledkem byl vznik nového směru v oblasti IT, který se nazývá *data mining* (dolování z dat). Jde o automatizovanou extrakci modelů dat představujících znalost, která je obsažena v datech uložených v rozsáhlých databázích, datových skladech, na webu či v jiných rozsáhlých úložištích dat nebo proudech dat. Je zřejmé, že proces dolování dat a hlavně jeho výsledky jsou využitelné v mnoha oblastech a oborech, podpora řízení, správa, rozhodování, analýzy a další. V obchodní sféře je tato schopnost „odezírání“ z dat přímo úměrná kvalitě rozhodnutí, které se vytváří na jejím základě. Tak je v porovnání s jinými subjekty tržního prostředí vytvářen silný prvek konkurenční výhody (nebo v opačném případě ztráty). V bankovnictví je tato schopnost využitelná při rozhodování o důvěryhodnosti zákazníka, ve zdravotnictví a výzkumu zase při objevování souvislostí na základě již známých informací. Bezesporu se dá tvrdit, že dolovat se dá v jakémkoliv oboru z jakýchkoliv dat, otázkou je jak kvalitní a použitelný získáme výsledek.

Cílem této práce je návrh a implementace predikčního modulu pro již existující systém získávání znalostí z databází vyvíjeným na FIT, který je postaven na jazyku DMSL. Modul by měl být realizován na platformě NetBeans. Opírá se o základy práce pana Ing. Krásného (*Systém pro dolování z dat v prostředí Oracle*) [2]. Tento systém prozatím poskytuje možnost výběru vstupních dat a řadu funkcí pro jejich předzpracování. V čem je systém „chudý“, jsou dolovací moduly. Doposud byl implementován jediný dolovací modul a to v rámci diplomové práce pana Ing. Madera [8] (*modul pro získávání asociačních pravidel*). Jak již bylo řečeno, cílem této práce je tedy vytvoření dalšího dolovacího modulu a tedy i rozšíření počtu možných způsobů dolování. Pro implementaci predikčního modulu byla vybrána metoda regrese.

Text je organizován do devíti kapitol. První kapitola v sobě zahrnuje úvod a popis práce. Druhá kapitola obecnou charakteristiku získávání znalostí z dat, třetí kapitola se věnuje predikci a predikčním metodám, čtvrtá kapitola popisuje použité technologie. Pátá kapitola se věnuje systému, pro který se tento modul vyvíjí. Šestá kapitola se věnuje návrhu modulu. Sedmá kapitola popisuje implementaci modulu. Osmá zobrazuje GUI a příklady užití. Poslední devátá kapitola shrnuje dosažené výsledky a diskutuje další možný vývoj projektu.

2 Získávání znalostí z dat

Získávání znalostí z databází [3] je extrakce (neboli „dolování“) zajímavých (netriviálních, skrytých, dříve neznámých a potenciálně užitečných) informací z velkých objemů dat. Netriviálnost znamená, že nejde o informaci, kterou lze získat např. jen pouhým SQL dotazem, nepatří sem ani deduktivní databázové a expertní systémy. *Skrytost* říká, že musí jít o informace, které nejsou na první pohled patrné, musí se teprve nalézt. *Potenciální užitečnost* získaných znalostí (informací) znamená, že má význam pro další rozhodování. Může jít například o podklad pro rozhodnutí o půjčce klientovi banky nebo rozhodnutí o uspořádání zboží v supermarketu na základě znalosti druhů zboží, které zákazníci často kupují. Jiným příkladem může být upozornění na podezřelé bankovní operace či chování některých osob ve sledovaném prostoru. V oblasti vědy může výsledek dolování dat být podkladem pro nějakou novou hypotézu nebo může sloužit k potvrzení či vyvrácení nějaké hypotézy dříve stanovené.

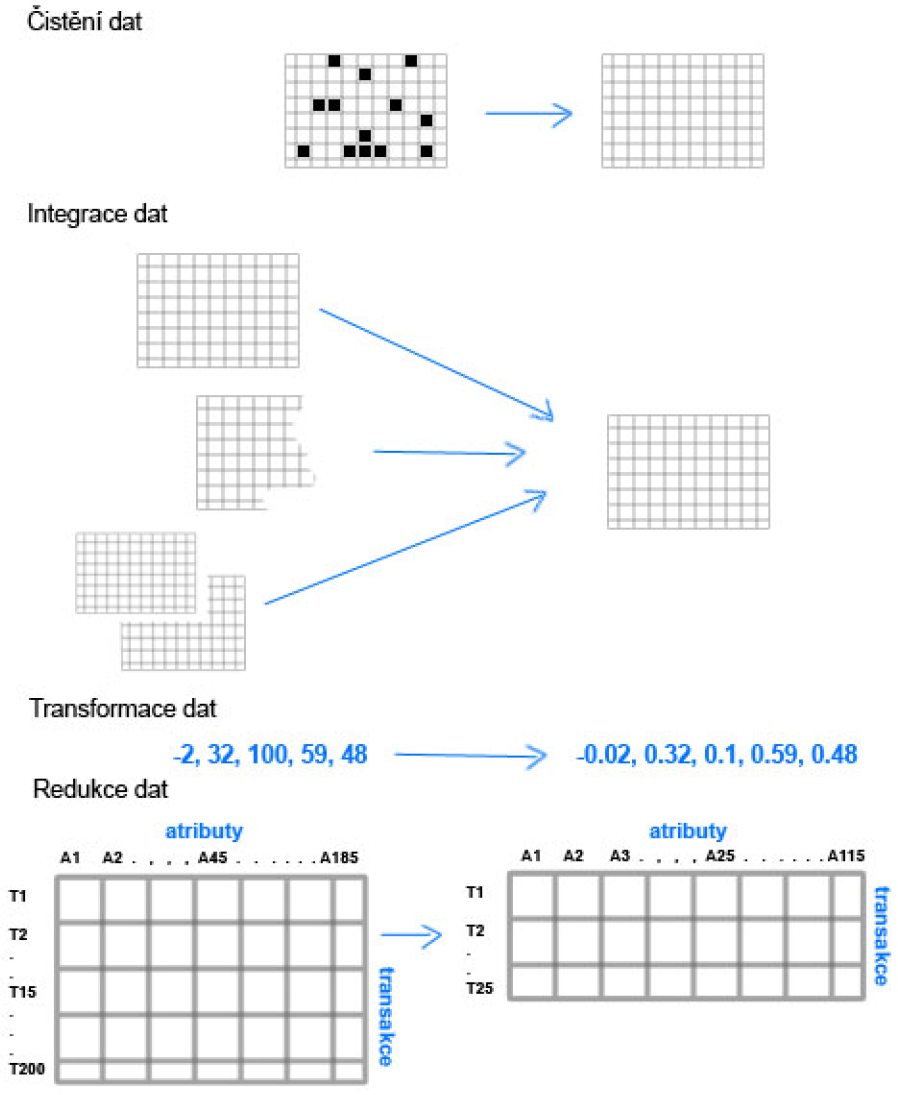


Obrázek 2.1 Proces dolování dat

Proces získávání znalostí z dat se skládá z několika kroků, které se z pravidla v určitých iteracích opakují a kterým se budu následně věnovat. Tento proces demonstruje obrázek 2.1. V této kapitole jsem vycházel z prací [1] a [3]. Nejprve je třeba připravit a vyčistit data, poté najít znalosti a z nich vybrat jen ty důležité a zajímavé.

2.1 Příprava a předzpracování dat

Většina dat, které využíváme pro dolování dat, nejsou zpravidla v takovém tvaru, abychom na nich mohli bezprostředně použít dolovací algoritmus k dolování. Obsahují obvykle více informací než je pro naše zpracování potřeba, zašuměná data, nekonzistentní data a mnohdy mohou některé hodnoty chybět. Právě špatná kvalita vstupních dat může vést k nepřesným či dokonce nesprávným a zavádějícím výsledkům z dolování. Důležitým krokem je tedy *čištění a integrace dat*.



Obrázek 2.2 Příprava a předzpracování dat

2.1.1 Čištění dat

Cílem je odstranit chybějící hodnoty, najít, identifikovat a odstranit odlehlé hodnoty, řešit nekonzistence dat a řešit redundance způsobené integrací dat. Důvodem těchto problémů může být lidský faktor, porucha na přístroji pro sběr dat, chyba komunikačního kanálu atd. Nekvalitní vstupní data mohou ve velké míře ovlivnit i samotný dolovací algoritmus, který poté může dávat nespolehlivé výsledky. Příkladem může být například zjištění platu běžného občana. Problém je v tom, že pokud spočítáme pouze aritmetický průměr platů všech občanů, může se stát, že výsledek ve značné míře ovlivní menšina občanů s nadprůměrnými daty. Tyto nadprůměrné platy jsou příkladem odlehlých hodnot, proto je potřeba tyto odlehlé hodnoty odstranit.

2.1.1.1 Chybějící hodnota

Častým problémem je chybějící hodnota atributu, který však může reprezentovat do značné míry důležité informace pro proces dolování. Většinou se jedná o neklíčové atributy, které dovolují uchovávat hodnoty *NULL*. Odstranění těchto chybějících hodnot může proběhnout následujícími způsoby:

1. *Ignorování n-tice* – je nejjednodušším způsobem jak provést odstranění chybějících hodnot, nicméně tak můžeme přijít i zajímavé informace. Tento způsob je vhodný pouze v případě, pokud v prvku relace chybí některé další atributy (nelze odvodit hodnotu chybějícího atributu) nebo v případě čištění dat pro klasifikaci.

2. *Manuální nahrazení* – tento způsob je mnohem vhodnější, protože se očekává uživatelova dobrá znalost problematiky a je schopen data vhodně doplnit. Nicméně je tento způsob značně nereálný při velkých objemech dat.

3. *Automatická náhrada* – tento způsob nabízí několik variant nahrazení:

    a. *Globální konstantou* – jde o obdobu NULL v databázích. Používá se hodnota mimo rozsah platných hodnot daného atributu (např. 0 nebo ∞ pro numerický atribut). Pokud by výskyt této odlehlé hodnoty byl nízký, algoritmus pro dolování ji může ignorovat, ale v případě častého výskytu může tento způsob negativně ovlivnit výsledek dolování.

    b. *Průměrnou hodnotou atributu* – jedná se o průměr hodnot atributu z ostatních hodnot relace

    c. *Průměrem hodnot n-tic patřících do téže třídy* – je použita průměrná hodnota atributu z relací, které patří do stejné třídy. Pokud například chybí hodnota atributu *příjem* v řádku klienta, který má návěští třídy (hodnota atributu *schopnost_splácet*) rovno *bezproblémový*, pak by se na místo chybějící hodnoty doplnila hodnota průměrného příjmu všech klientů patřících do kategorie *bezproblémový*.

    d. *Nejpravděpodobnější hodnotou* – tato hodnota se odhadne podle ostatních nenulových atributů. Řeší se vlastně úloha klasifikace a predikce s hledaným atributem jako cílem. Lze využít *regrese*, *Bayesovské klasifikace* nebo *rozhodovacího stromu.*

Všechny metody automatické náhrady určitým způsobem ovlivňují data, jako nejpříznivější se jeví poslední způsob nahrazení nejpravděpodobnější hodnotou, neboť nejvíce využívá informace obsažené v datech.

2.1.1.2 Šum v datech

Šumem v datech rozumíme náhodné chyby nebo odchylky hodnot atributů. Příčin je několik, např. porucha na zařízení sběru dat, lidský faktor, chyby programu nebo hardwaru, nekonzistence v datech jako jsou různé formáty zadávaných dat a podobně. Způsoby k vyhlazení šumů jsou následující:

1. *Plnění (binning)* - tento způsob vyhlazuje setříděná numerická data pomocí lokálního vyhlazení. Vstupy se rozdělí do tzv. košů tak, že každý z nich obsahuje přibližně stejný počet hodnot. Poté se každá z těchto hodnot nahradí průměrnou hodnotou koše, mediánem nebo nejbližší krajní hodnotě koše.

2. *Regrese* – tento způsob vyhlazuje data pomocí regresní křivky

3. *Shlukování* – tento způsob je sám o sobě dolovací úlohou, může však také posloužit k odstranění odlehlých hodnot. Ty hodnoty, které po aplikaci shlukování nepatří do žádné skupiny shluku, můžeme prohlásit za odlehlé.

2.1.2 Integrace dat

Data, která zpracováváme, obvykle pochází z více zdrojů a je potřeba je integrovat do jednotného úložiště. Přitom může dojít k nejednotnosti atributů. Například identifikátor v jednom zdroji se může nazývat *ItemID* a v druhém zdroji třeba *IdPolozky*. Mezi hlavní problémy integrace patří:

1. *Konflikt schématu* – znamená integraci metadat do jedněch metadat popisujících výsledný zdroj dat. Příkladem může být zadání adresy. V jednom zdroji může být adresa zadávána pomocí tří sloupců, jako jsou *ulice*, *město*, *PSČ* a ve druhém zdroji pouze jedním sloupcem *adresa*. Proto je potřeba při integraci takové konflikty schématu řešit.

2. *Konflikt hodnot* – konfliktem hodnot rozumíme zpravidla různé formáty dat, různé stupnice a jednotky, různé konvence a podobně. Řešením je transformace do jednotné formy.

3. *Konflikty identifikace* – tytéž objekty reálného světa mohou být identifikovány odlišným způsobem nebo naopak odlišné objekty stejně. Například pokud slučujeme informace o zaměstnancích z různých poboček, ve kterých měli v rámci své pobočky unikátní identifikátor, pak dochází při sloučení k poruše této unikátnosti.

4. *Redundance* – u redundance nejde pouze o výskyt stejných atributů v různých zdrojích dat, ale i o odvozené atributy, jejichž hodnotu lze odvodit z hodnot jiných atributů. Například z data narození jsme schopni odvodit věk, stejně tak z rodného čísla.

2.1.3 Transformace dat

Cílem transformace je dostat data do podoby vhodné pro dolování. Bývá nástrojem pro integraci dat. Transformace může obsahovat následující operace:

1. *Vyhlazení* – jedná se o odstranění šumu z dat, viz kapitola 2.1.1.2
2. *Konstrukce atributů* – jde o odvození nových atributů ze stávajících atributů, cílem je zkvalitnit dolování.
3. *Agregace* – jinak také sumarizace dat. Rozdíl mezi agregací a konstrukcí atributů je ten, že při agregaci sumarizujeme data v nějakém rozměru (čas, pobočka), kdežto konstrukce atributů pouze sloučí více atributů do jednoho.
4. *Generalizace* – jde o nahrazení hodnoty atributu jejich obecnější hodnotou jako u hierarchie konceptů. Například atribut ulice může být na vyšší úrovni nahrazen názvem města či kraje.
5. *Normalizace* – transformace dat do vhodnějšího tvaru – intervalu, typicky <-1,1>, <0,1>.

2.1.4 Redukce dat

Dolování se provádí nad obrovským množstvím dat, cílem redukce je tyto data do jisté míry zredukovat bez porušení integrity a zachovat charakter původního souboru. Existuje několik druhů redukcí:

1. *Agregace datové kostky* – jedná se o redukci dat agregováním údajů, typické pro datové sklady
2. *Výběr podmnožiny atributů* – cílem je vybrat pouze ty atributy, které jsou potřeba pro dolování
3. *Redukce dimenzionality* – data se zakódují tak, že dojde k redukci, ale je možné s daty nadále provádět operace
4. *Redukce počtu hodnot* – data jsou nahrazena nějakým redukovaným modelem
5. *Diskretizace a generace konceptuální hierarchie* – jedná se o metodu mapující souvislé hodnoty na diskrétní intervaly.

2.2 Typy dolovacích úloh

Dolovací úlohy můžeme rozdělit do dvou základních kategorií a to na *popisné (deskriptivní)* a *prediktivní*. Popisné charakterizují vlastnosti dat a prediktivní na základě analýzy použitých dat provádí dedukci pro předpověď budoucích dat.

2.2.1 Charakterizace a diskriminace dat

2.2.1.1 Charakterizace dat

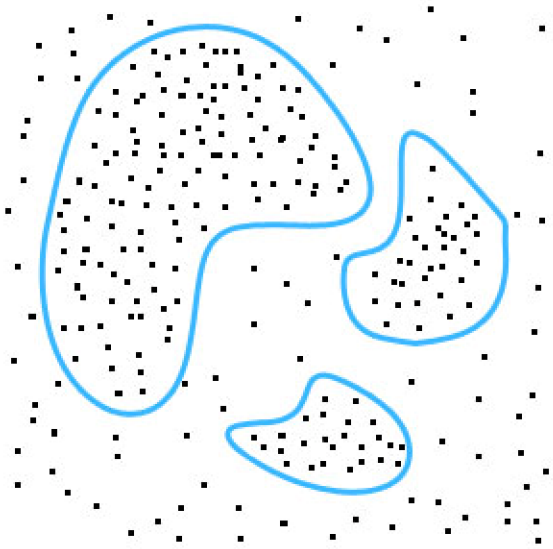
Charakterizace značí sumarizaci vlastností analyzované třídy. Tyto data lze obvykle dostat z databáze nějakým jednoduchým dotazem. Příkladem může být dotaz: *„Co je charakteristické pro zákazníky, kteří utratili v našem obchodě více jak 5000 Kč? “*

2.2.1.2 Diskriminace dat

Diskriminace na rozdíl od charakterizace hledá atribut nebo atributy, které se nejvíce liší od dané třídy. Příkladem může být dotaz: *“Čím se liší zaměstnanci, kteří za minulý rok využili méně dovolené od těch, kteří využili všechnu?“.*

2.2.2 Shluková analýza

Shluková analýza je dalším typem dolovacích úloh, který shlukováním analyzuje datové objekty bez znalosti přiřazení do tříd. Cílem je tedy nalézt třídy objektů, které mají co nejvíce společného a zároveň se od ostatních co nejvíce liší. Tyto třídy objektů pak vytváří shluky.



Obrázek 2.3 Shluková analýza

2.2.3 Analýza odlehlých hodnot

Tato analýza vyhledává extrémní, neobvyklé, něčím výjimečné prvky. Důležité je tedy, aby při přípravě dat nebyly použity metody, které by odlehlé hodnoty vyhladily nebo je nějakým způsobem poškodila normalizace. Tato metoda se nejvíce používá při odhalování bankovních podvodů nebo při analýze datových toků.

2.2.4 Evoluční analýza

Jedná se o analýzu dat v čase, která hledá modely trendů a jejich vývoj. Zkoumá se vývoj a rychlost změn v čase anebo pravidelnost dat v jistých časových intervalech. Tato metoda se využívá při analýze průběhu hodnot akcií a rozhodování o investicích

2.2.5 Asociační analýza

Asociační analýza je metoda, ve které hledáme tzv. *asociační pravidla*. Jedná se o vztahy mezi daty, které se dají nejlépe vysvětlit na nákupním košíku. Máme-li seznam nákupů (transakcí), které zákazníci uskutečnili, pak můžeme hledat vztahy mezi jednotlivými výrobky (položky v transakcích). Hledáme množiny takových výrobků, které zákazníci často kupují dohromady v jednom nákupu (silné množiny).

Například můžeme získat asociační pravidlo, které tvrdí že *„Zákazník který si koupí rohlíky si také s 60% pravděpodobností koupí uzeninu. Co do četnosti výskytu ze všech uskutečněných nákupů je toto pravidlo platné v 12%“*

2.2.5.1 Frekventovaná množina prvků

Frekventovaná množina prvků je taková množina vzorů, která se vyskytuje v datech velmi často. Vede k odhalení zajímavých korelací nebo použití asociační analýzy a vyhledávání asociačních pravidel ve tvaru $A \Rightarrow B$, kde A a B jsou tvrzení týkající se hodnot atributů. Asociační pravidlo nám říká, jaká je pravděpodobnost výskytu prvků v transakci.

Silná asociační pravidla

jsou taková, která mají vyšší spolehlivost než je námi stanovená minimální spolehlivost. Opakem jsou pravidla slabá.

Asociační pravidla se hledají ve dvou krocích:

1. *Nalezení frekventovaných množin* – nejpoužívanějším algoritmem pro získání těchto množin je *algoritmus Apriori*, nebo některá z jeho variant. Tento krok je jádrem celého dolování.

2. *Vygenerování silných asociačních pravidel* – z frekventovaných množin vygenerujeme všechna asociační pravidla a ponecháme jen silná.

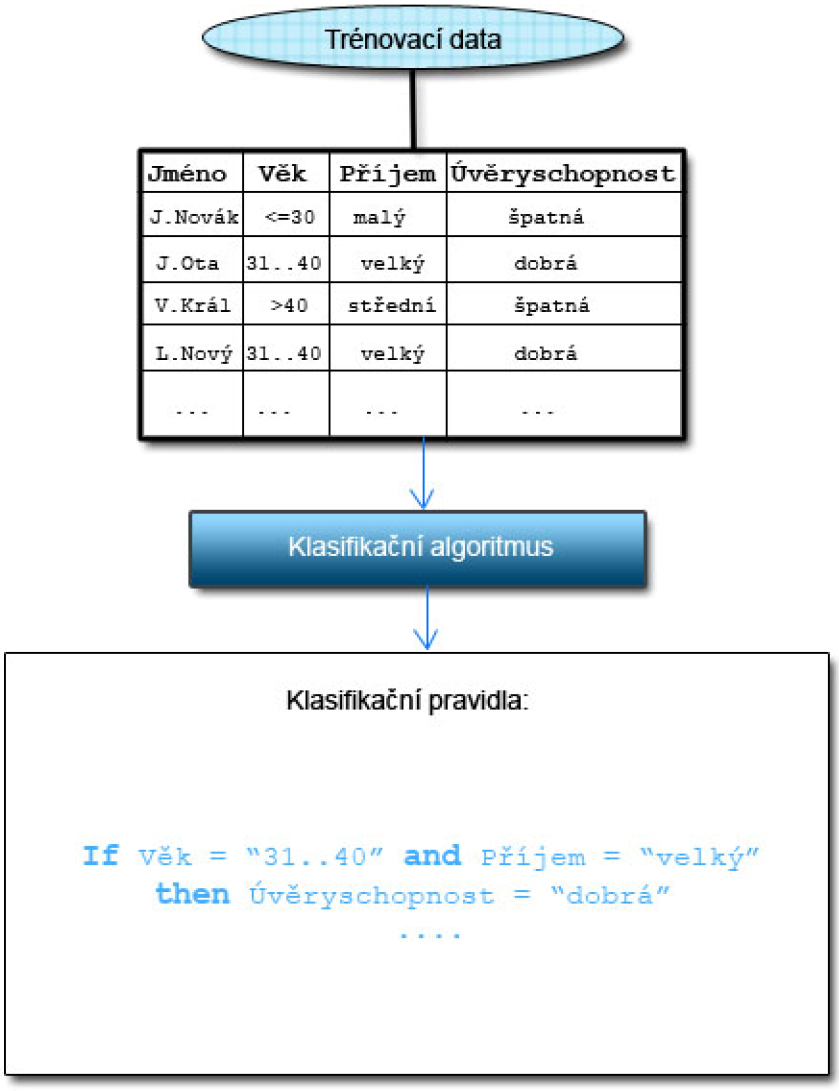
2.2.6 Klasifikace a predikce

Obě tyto metody patří do prediktivních dolovacích úloh, tedy úloh, které na základě stávajících dat předpovídají něco nového.

2.2.6.1 Klasifikace

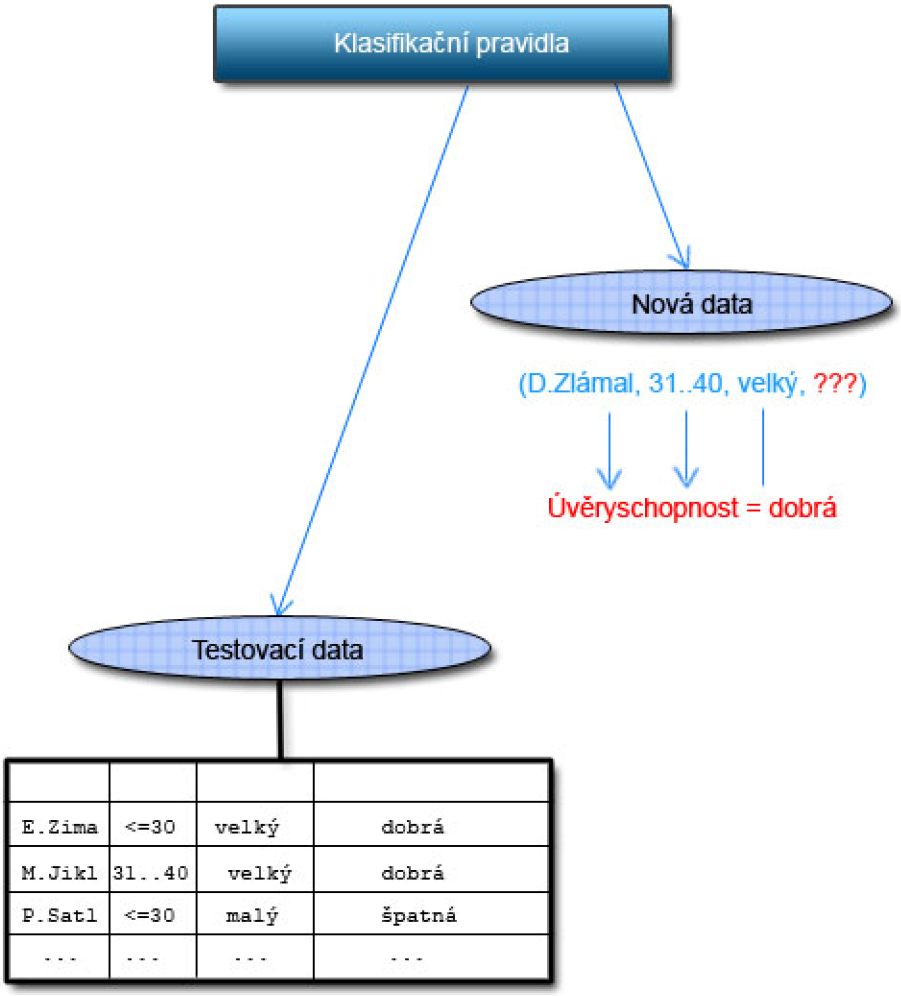
Cílem klasifikace je nalézt takový model, který popisuje a současně rozlišuje třídy dat a ten poté použít pro predikci. Příkladem klasifikace může být třeba potřeba rozdělit klienty banky do dvou skupin na základě známých dat, tak aby jednu skupinu tvořili klienti s minimálním rizikem pro udělení půjčky a ve druhé skupině klienti s rizikem vysokým. Proces klasifikace je rozdělen do dvou kroků:

1. *Trénování (učení)* – na základě analýzy trénovací množiny je vytvořen klasifikační model, u těchto dat, která tvoří trénovací množinu, musíme předem znát, do jaké třídy jsou zařazeny. Tyto vzorky dat jsou vstupem pro klasifikátor, ten má za úkol zjistit tzv. klasifikační pravidla, pomocí kterých se daný objekt klasifikuje do konkrétní dané třídy. Tuto fázi popisuje obrázek 2.4



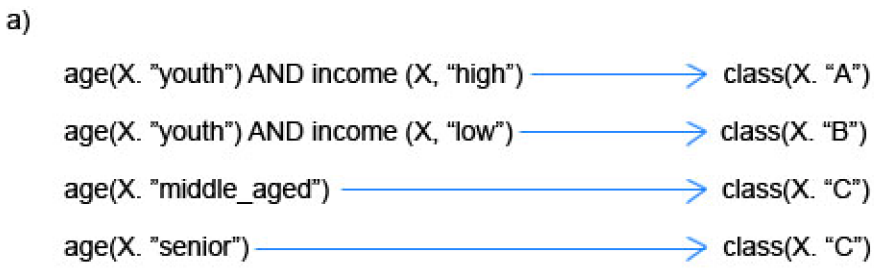
Obrázek 2.4 Klasifikace – fáze trénování

2. *Testování* – testováním ověřujeme kvalitu modelu pomocí testovací množiny. Testovací množinu opět tvoří data, u kterých je předem známa klasifikační třída. **Tyto vzorky dat ale musí být nezávislé na předchozích datech**, neměly by tedy být vybrány z trénovací množiny. Testováním získáváme četnost výsledků, pro které daný model klasifikuje správně. Podle tohoto výsledku se pak odvíjí další kroky. Pokud jsou výsledky špatné, je potřeba model upravit nebo vytvořit znovu, v opačném případě daný model lze prohlásit za připravený. Tuto fázi popisuje obrázek 2.5

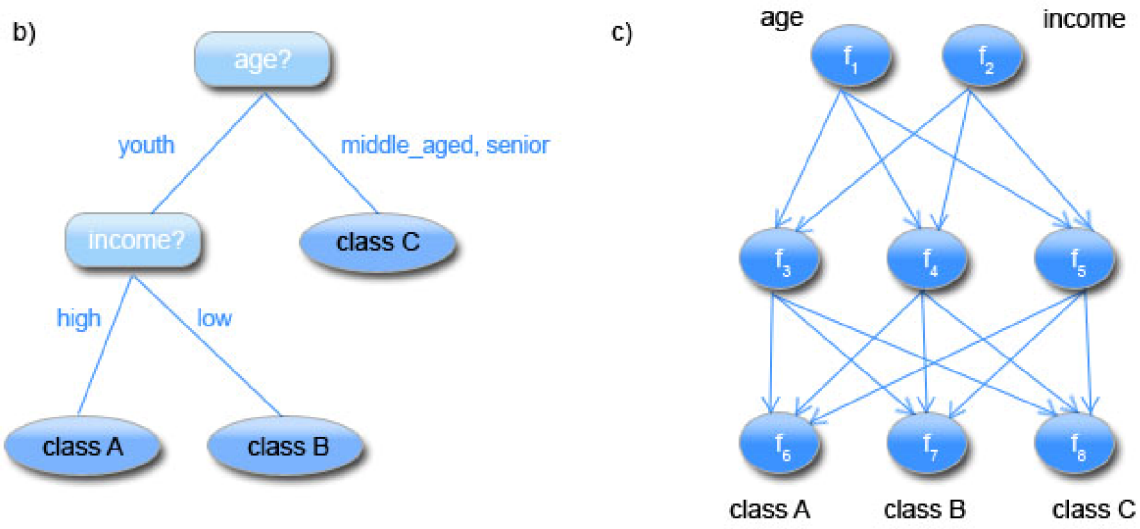


Obrázek 2.5 Klasifikace – fáze testování

Model klasifikace může mít (viz obrázek 2.6 a 2.7) různou podobu, například podobu klasifikačních *IF-THEN pravidel*, *rozhodovacího stromů*, *matematické formule*, *neuronové sítě* a mnoho dalších.



Obrázek 2.6 Formy reprezentace klasifikačního modelu: a) klasifikační pravidla



Obrázek 2.7 Formy reprezentace klasifikačního modelu: b) rozhodovací strom, c) neuronová síť

Klasifikace se používá k predikci diskrétních hodnot. Naopak pro hodnoty spojité využíváme Predikci.

2.2.6.2 Predikce

Predikce je proces, který umožní přiřazovat datům hodnoty, které mají obecně spojitý charakter. Predikci je věnována následující samostatná kapitola č. 3.

2.2.6.3 Příprava dat pro klasifikaci a predikci

Abychom zefektivnili predikci i klasifikaci, je vhodné provést různé úpravy vstupních dat. Mezi ty základní patří následující tři, o nichž jsme se již zmínili v předcházejících kapitolách:

1. *Čištění dat* – jedná se o odstranění šumu v datech a chybějících hodnot.

2. *Významnostní analýza* – jde o odstranění nepotřebných atributů v datech pro danou klasifikaci

3. *Transformace dat* – převod dat na jiný formát, typickým prvkem transformace je normalizace

2.2.6.4 Porovnávání klasifikačních metod

Predikční a klasifikační metody se porovnávají především podle následujících pěti kritérií:

1. *Robustnost* – popisuje schopnost vytvořit správný model, pokud daná data obsahují šum a chybějící hodnoty
2. *Rychlost* – popisuje složitost pro vygenerování a používání klasifikačních pravidel

3. *Přesnost* – popisuje v jaké míře – v kolika procentech daný model klasifikuje nová data, tedy data, která nebyla obsažena v trénovací množině

4. *Stabilita* – popisuje schopnost vytvořit správný model pro obrovské množství dat

5. *Interpretovatelnost* – popisuje složitost daného modelu pro pochopení

3 Predikce

O predikci již bylo něco málo řečeno v předcházející kapitole, tato kapitola se jí věnuje více dopodrobna. Predikcí myslíme předpověď určité hodnoty (obecně *spojitého charakteru*) pro daný objekt na základě jeho vlastností. Příkladem predikce je například určení výše platu daného pracovníka na základě znalostí jeho dalších vlastností. Predikce se tedy liší od klasifikace v tom, že předpovídá hodnoty spojitého charakteru, klasifikace diskrétního charakteru. Obě tyto metody mají však společnou chronologii procesu.

Proces predikce je tedy stejně jako proces klasifikace rozdělen do dvou kroků:

1. *Trénování (učení)* – na základě analýzy trénovací množiny dat je vytvořen predikční model (závislost prediktorů, v závislosti na známé hodnotě, která by měla být predikována). Jednoduše řečeno, na vzorku dat z trénovací množiny se proces „naučí“ jak predikovat – predikovaná hodnota je totiž známa. **Důležitou podmínkou je nezávislost vzorků dat pro trénovací množinu a pro testovací množinu**. Neměly by tedy obě tyto množiny obsahovat stejná data.

2. *Testování* – testováním ověřujeme kvalitu modelu pomocí testovací množiny. Testovací množinu opět tvoří data, u kterých je předem známa hodnota která by měla být predikována. Testováním tedy získáváme predikované hodnoty na základě vytvořeného predikčního modelu a porovnáváme je s hodnotami, které byly známy v testovací množině vstupních dat a to ty, které měly být predikovány. Podle tohoto výsledku se pak odvíjí další kroky. Pokud jsou výsledky špatné, tedy rozdíl mezi predikovanými daty a známými daty je příliš velký, je potřeba model upravit nebo vytvořit nový, v opačném případě daný model lze prohlásit za připravený.

Někdy se jako další krok procesu uvádí *Aplikační fáze* (Aplikace) – ta na základě vytvořeného a otestovaného predikčního modelu a množině vstupních dat pro aplikaci je schopna predikovat nová data.

Pro metodu predikce se využívá *regrese – regresní analýza*. Regresní analýza se snaží nalézt hodnoty parametrů funkce, takové, která nejlépe kopíruje výsledné body na množině dat. Následující rovnice popisuje regresní funkci, ta zobrazuje proces odhadující hodnotu spojité veličiny jako funkce F s jedním nebo více prediktory (x_1, x_2, …, x_n), množinou parametrů (θ_1, θ_2,…, θ_n) a odchylkou (e). (Viz rovnice 3.1):

$$y = F(x, \theta) + e \tag{3.1}$$

Prediktory si můžeme představit jako nezávislé proměnné a cílovou hodnotu jako závislou proměnnou. Odchylku e chápeme jako rozdíl mezi známou cílovou hodnotou a predikovanou hodnotou závislé proměnné. Regresní parametry označujeme jako regresní koeficienty. Fáze učení regresního modelu se snaží nalézt parametry (koeficienty) s co nejmenší odchylkou e. Regresní funkce můžeme z matematického pohledu rozdělit do 3-4 typů. (lineární jednoduchá, lineární vícenásobná, nelineární jednoduchá, nelineární vícenásobná)

3.1 Lineární jednoduchá regrese

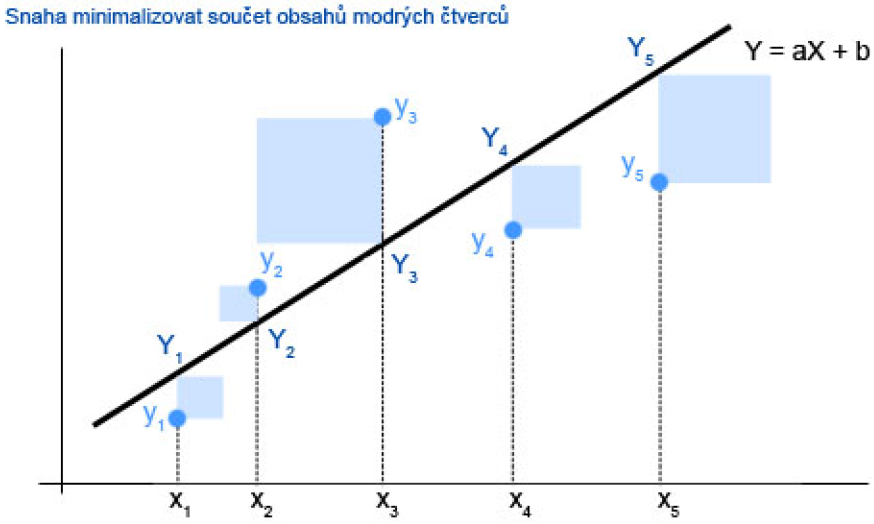
Lineární regrese [4] představuje aproximaci daných hodnot polynomem prvního řádu (tedy přímkou) metodou nejmenších čtverců. Jinak řečeno, jedná se o proložení několika bodů v grafu takovou přímkou, aby součet druhých mocnin odchylek jednotlivých bodů od přímky byl minimální.

Pro lineární jednoduchou [3] regresi jsou očekávána data ve tvaru $(x_1, y_1),\ldots,(x_s, y_s)$, kde x_i reprezentuje vstupní atribut a y_i výstupní atribut, jehož hodnota se bude pro nová data předpovídat, pro všechna $i = 1,\ldots, s$. Tato data jsou aproximována přímkou o rovnici (rovnice 3.2):

$$Y = aX + b \tag{3.2}$$

Zaveďme následující označení:

- $x_1, x_2, \ldots, x_s$ – značí vstupní atributy daných dat
- $y_1, y_2, \ldots, y_s$ – značí výstupní atributy skutečných dat
- $Y_1, Y_2, \ldots, Y_s$ – jsou hodnoty, které byly získány dosazením vstupního parametru do dané rovnice přímky (tedy ty hodnoty, které budou pro dané vstupní atributy předpovězeny)



Obrázek 3.1 Metoda nejmenších čtverců

Metoda nejmenších čtverců je založena na takovém principu, že součet druhých mocnin rozdílu skutečné hodnoty y_i a aproximované hodnoty Y_i je minimální. Tato myšlenka je ilustrována na obrázku 3.1

Pro metodu nejmenších čtverců lze koeficienty a, b spočítat pomocí vzorů (viz rovnice 3.3):

$$a = \frac{\sum_{i=1}^{s}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{s}(x_i-\bar{x})^2} \qquad b = \bar{y} - a\bar{x} \tag{3.3}$$

Kde $\bar{x}$ symbolizuje aritmetický průměr hodnot $x_1, x_2, \ldots, x_s$ a $\bar{y}$ symbolizuje aritmetický průměr hodnot $y_1, y_2, \ldots, y_s$.

3.1.1 Příklad výpočtu jednoduché lineární regrese

Mějme [3] tabulku obsahující data zaměstnanců firmy, kde x udává, kolik let má zaměstnanec praxi a y udává jeho plat v tisících Kč.

x_i	3	8	9	13	3	6	11	21	1	16
y_i	30	57	64	72	36	43	59	90	20	83

Tabulka č. 3.1 Příklad jednoduché lineární regrese – data zaměstnanců firmy

Z uvedených dat určíme $\bar{x}$ = 9,1 a $\bar{y}$ = 55,4 a z uvedených vzorců jsme schopni vypočítat koeficienty *a* a *b (*viz rovnice 3.4*)*:

$$a = \frac{(3-9{,}1)(30-55{,}4)+(8-9{,}1)(57-55{,}4)+\cdots+(16-9{,}1)(83-55{,}4)}{(3-9{,}1)^2+(8-9{,}1)^2+\cdots+(16-9{,}1)^2} = 3{,}5$$

$$b = 55{,}4-(3{,}5)(9{,}1) = 23{,}6 \tag{3.4}$$

Rovnice přímky má tedy tvar: Y = 3,5X + 23,6
Pro 5letou praxi pracovníka (X = 5) prediktor například předpoví plat: Y=(3,5)(5) + 23,6 = 41 100 Kč

3.2 Lineární vícenásobná regrese

Lineární vícenásobná regrese je regrese, která na rozdíl od jednoduché lineární regrese je schopna predikovat nikoliv jeden atribut, ale obecně *v* atributů. Data očekávejme [3] ve tvaru:

$$(x_{11}, x_{12}, \dots, x_{1v}, y_1), (x_{21}, x_{22}, \dots, x_{2v}, y_2), \dots, (x_{s1}, x_{s2}, \dots, x_{sv}, y_s)$$

Aproximaci provedeme pomocí rovnice (rovnice 3.5):

$$Y = a_0 + a_1X_1 + a_2X_2 + \cdots + a_vX_v \tag{3.5}$$

Je tedy potřeba najít hodnoty koeficientů $a_0, a_1, a_2, \dots, a_v$. Sestavme ze získaných dat matice *X* a *Y*, výsledek bude matice A:

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1v} \\ 1 & x_{21} & \cdots & x_{2v} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{s1} & \cdots & x_{sv} \end{pmatrix} \qquad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{pmatrix} \qquad A = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_v \end{pmatrix}$$

Vzorec, podle kterého určíme výslednou matici *A* (a tím jednotlivé koeficienty) je (rovnice 3.6):

$$A = (X^TX)^{-1}X^TY \tag{3.6}$$

kde horní index *T* značí matici transponovanou a horní index –1 matici inverzní. Příklad na vícenásobnou regresi je uveden v následující kapitole, která zadání úkolu transformuje na právě tento druh regrese.

3.3 Nelineární regrese

Nelineární regresi řešíme většinou tak, že ji transformujeme na lineární regresi. Předpokládejme, že chceme například provést regresi pomocí kubické funkce, tedy funkce, kterou budeme hledat ve tvaru (viz rovnice 3.7):

$$Y = a_0 + a_1 X + a_2 X^2 + a_3 X^3 \tag{3.7}$$

Potom můžeme nadefinovat následujícím způsobem nové proměnné X_1, X_2 a X_3 (viz rovnice 3.8):

$$X_1 = X,\ X_2 = X^2,\ X_3 = X^3 \tag{3.8}$$

Dosadíme-li tyto nové proměnné do výše uvedeného vzorce, dostaneme novou funkci ve tvaru (viz rovnice 3.9):

$$Y = a_0 + a_1 X_1 + a_2 X_2 + a_3 X_3 \tag{3.9}$$

Toto ovšem není nic jiného, než rovnice pro lineární vícenásobnou regresi, kterou již řešit umíme. Podobným způsobem lze převést i nějaké další nelineární regrese na lineární.

3.3.1 Příklad nelineární regrese transformované na lineární vícenásobnou

Uvažujme následující tabulku dat, u které předpokládáme, že hodnota *y* je závislá na hodnotě *x*, a to kvadraticky:

x_i	**0**	**1**	**-1**
y_i	2	2	0

Tabulka č. 3.2 Příklad nelineární regrese transformované na lineární – data

Regresní funkci budeme hledat tedy ve tvaru (viz. rovnice 3.10):

$$Y = a_0 + a_1 X + a_2 X^2 \tag{3.10}$$

Zavedeme nové proměnné (atributy) (viz. rovnice 3.11):

$$X_1 = X,\ X_2 = X^2 \tag{3.11}$$

Dosadíme-li tyto nové proměnné do výše uvedeného vzorce, dostaneme již lineární regresní funkci ve tvaru (viz. rovnice 3.12):

$$Y = a_0 + a_1 X_1 + a_2 X_2 \tag{3.12}$$

Tabulku dat tedy můžeme rozšířit o tyto nové atributy, které jsou spočítány z hodnoty atributu $\boldsymbol{x_i}$.

x_i	$x_{1i} = x_i$	$x_{2i} = x_i^2$	y_i
0	0	0	2
1	1	1	2
-1	-1	1	0

Tabulka č. 2.3 Příklad nelineární regrese transformované na lineární – rozšíření předešlé tabulky

Hodnoty z tabulky můžeme přepsat do matic, které jsou tedy ve tvaru:

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{pmatrix} Y = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}$$

Provedeme výpočet koeficientů a_0, a_1, a_2 (viz. rovnice 3.13):

$$X^T X = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 2 \end{pmatrix}$$

$$(X^T X)^{-1} = \frac{1}{4} \begin{pmatrix} 4 & 0 & -4 \\ 0 & 2 & 0 \\ -4 & 0 & 6 \end{pmatrix} \tag{3.13}$$

$$A = (X^T X)^{-1} X^T Y = \frac{1}{4} \begin{pmatrix} 4 & 0 & -4 \\ 0 & 2 & 0 \\ -4 & 0 & 6 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ -1 \end{pmatrix}$$

Platí tedy, že $a_0 = 2, a_1 = 1, a_2 = -1$.
Hledaná regresní funkce je ve tvaru: $Y = 2 + X - X^2$

4 Použité technologie

4.1 Oracle Data Mining

Součástí relačního databázového systému firmy Oracle [8] je i podpora pro dolování z dat, která jsou v databázi Oracle uložena. Podpora se nazývá *Oracle Data Mining* a je dostupná od databázové verze 9iR2. ODM nabízí funkce pro předzpracování dat, které předcházejí vlastnímu dolování. Těmito funkcemi jsou například diskretizace, normalizace nebo plnění (binning). Vstupními daty dolovacích funkcí je relační tabulka nebo pohledy uložené v databázi. Hlavní výhodou řešení firmy Oracle je umístění veškerých dolovacích funkcí do jádra databáze. Toto řešení eliminuje jakýkoliv přesun dat a šetří mnoho času. Uživatel pouze definuje model dolovací úlohy, uloží ho na server a požádá o spuštění dolovacího procesu. ODM dále nabízí podporu pro dolování kolekcí algoritmů a datových analýz pro klasifikaci, predikci, regresi, shlukování, asociační analýzu, měření významnosti atributů, detekci odlehlých hodnot a další specializované analýzy. Spolupráce s dolovacími funkcemi je uživateli umožněna buď přes jazyk PL/SQL nebo Java API (*Java Data Mining*). 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*. K dispozici jsou také vzorové příklady a programové kódy, které by měly uživateli ukázat, jak nejefektivněji dolovacího systému ODM využít.

4.1.1 Přehled funkcí ODM

4.1.1.1 Předzpracování dat:

1. *Transformace dat*: vzorkování dat, plnění (binning), diskretizaci a další transformace

4.1.1.2 Deskriptivní úlohy

1. *Asociační analýza:* pro dolování asociačních pravidel ODM používá algoritmus *Apriori*
2. *Shlukování:* dostupné jsou algoritmy *k-means*, *Orthogonal Partitioning Clustering* O-Cluster
3. *Textové a prostorové dolování:* dolování v textu, prostorová/GIS data
4. *Specializované analýzy:* porovnání sekvencí DNA a proteinových řetězců. K dispozici je BLAST algoritmus

4.1.1.3 Prediktivní úlohy

1. *Klasifikace:* pro klasifikační úlohy jsou k dispozici *rozhodovací stromy*, *bayesovská klasifikace*, *bayesovské sítě* a *SVM* (*Support vector machines*)
2. *Regrese: SVM* (*Support vector machines*)

3. *Měření významnosti atributů:* Tato úloha slouží k ulehčení práce klasifikačních modelů při práci nad datovými tabulkami, které obsahují mnoho atributů. Měření významnosti má za úkol vybrat z množiny atributů podmnožinu takových, které jsou nejvíce relevantní pro určení výsledku klasifikace. Použit je algoritmus *Minimum description length* (MDL)

4. *Detekce odlehlých hodnot: One-class SVM* (*Support vector machines*)

Všechny prediktivní úlohy poskytují několik možností zobrazení výsledků. První je tabulka přesnosti, která udává, v kolika případech byla klasifikační metoda úspěšná. Další možností je zkoumání několika druhů tzv. "*liftů*" (kumulativní, nekumulativní, kvantilový a další). Tyto charakteristiky jsou použitelné pouze při klasifikaci do binárních tříd. Dále je také možné porovnávat úspěšnosti jednotlivých klasifikačních metod mezi sebou pomocí tzv. "*Receiver Operating Characteristics (ROC křivek)*".

4.2 Platforma NetBeans

NetBeans [12] je open source projekt podporovaný zejména firmou Sun Microsystems. V rámci projektu existují dva hlavní produkty: vývojové prostředí NetBeans (*NetBeans IDE*) a vývojová platforma NetBeans (*NetBeans Platform*).

Vývojové prostředí NetBeans IDE je nástroj, pomocí kterého programátoři mohou psát, překládat, ladit a šířit programy. 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 (*C++, PHP, Ruby*). V Javě podporuje všechny 3 hlavní platformy - *J2SE, J2EE* a *J2ME*. Zjednodušuje práci s frameworky jako je *Struts* nebo *RoR* a umožňuje mimo jiné vývoj SOAP aplikací a webových služeb i webových aplikací. Obsahuje také *RAD* nástroje pro tvorbu designu aplikace. Kromě toho také existuje velké množství modulů, které toto vývojové prostředí rozšiřují.

Kromě vývojového prostředí je také dostupná vývojová platforma NetBeans Platform, což je modulární a rozšiřitelný 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.

5 Systém pro dolování z dat vyvíjený na FIT

V posledních několika letech je na VUT Fakultě Informačních Technologií v rámci ročníkových a diplomových prací vyvíjen systém pro získávání znalostí z databází. Systém je již kompletní a umožňuje tak veškeré kroky procesu dolování – výběr a předzpracování dat, umožňuje připojení modulů, které zajišťují samotné dolování a prezentaci výsledků. V následujících letech a pracích půjde spíše o vytváření nových dolovacích modulů a možné vylepšování stávajícího systému.

5.1 Charakteristika systému

Klientská aplikace je implementovaná v prostředí jazyka *Java* a postavena na jazyku *DMSL*. Klient se připojuje k serveru *Oracle*. Výhodná je volba prostředí Java, díky němuž je systém přenositelný a využívá veškeré výhody tohoto řešení. Jazyk DMSL je využíván systémem k definici všech dat a operací nad nimi, ke komunikaci s jádrem a moduly. Práce mezi vrstvami je rozdělena následovně: klientská aplikace zajišťuje definici dat, příprava a transformace těchto dat jsou rozděleny mezi obě vrstvy (klient, server) a samotné dolování pak probíhá na serveru. Systém je postaven na platformě *NetBeans Platform* a je plně modulární, což byl hlavní požadavek při implementaci.

5.2 Historie projektu

Začátku projektu předcházelo několik návrhů a prací, nicméně základ tohoto systému pracujícím nad databází *Oracle*, byl vytvořen (v roce 2006) diplomovou [5] prací Ing. Doležala. Cílem jeho práce bylo vytvořit jádro systému, které bylo základem pro formulářovou aplikaci, vykonávalo předzpracování dat, řídilo samotný proces dolování a mělo jednoduché grafické uživatelské rozhraní. Systém byl navrhnut a naprogramován tak, aby bylo možné do něj přidávat dolovací metody pomocí nezávislých modulů. Dolovací modul lze tedy vytvořit samostatně a jediné, co je vyžadováno je to, aby modul implementoval rozhraní pro dolovací modul.

Na jeho práci o rok později navázal svou diplomovou prací [6] Ing. Gálet (2007). Ten měl za úkol vylepšit stávající systém o přijatelnější grafickou nadstavbu, příjemné uživatelské rozhraní a některé nedostatky jádra. Ve své práci poukázal na některé další nedostatky systému a navrhnul možnosti pro jejich odstranění a následné vylepšení. Díky platformě, na které byl systém vyvíjen, bylo možné využít různé nové techniky a na ty Ing. Gálet upozornil. Ty byly předlohou pro diplomovou práci [2] Ing. Krásného (2008). Provedl změny v grafické nadstavbě a zásadně přestavěl jádro, aby bylo možné více využít potenciál grafického uživatelského prostředí, protože umožňoval definovat dolovací proces pomocí orientovaných grafů. Byly vytvořeny nové datové struktury, pro něž byly dopsány nové algoritmy jádra. Dále byl upraven jazyk *DMSL*, zejména formát ukládání dat, aby lépe vyhovoval novým požadavkům aplikace a také bylo změněno rozhraní pro připojování dolovacích modulů. Poslední práce věnovaná tomuto systému je práce Ing. Madera [8], ve které vytvořil první funkční dolovací modul pro získávání asociačních pravidel.

Více podrobných informací o tomto projektu naleznete v pracích [2], [5], [6] a [8].

5.3 Struktura systému

5.3.1 Jazyk DMSL

Jazyk DMSL (*Data Mining Specification Language*) [7] je založený na bázi *XML*, tedy stejně jako v *XML* jeho strukturu tvoří elementy. V projektu postihuje celý proces dolování dat, od definice vstupních dat, přes jejich transformace, definice dolovacích úloh až po popis získaných znalostí. Je určen k ukládání *metadat*, nikoliv samotných dat. Data se nacházejí v databázových tabulkách na serveru, *DMSL* pouze popisuje kde.

5.3.1.1 Struktura jazyka

Struktura [6] jazyka se skládá ze šesti částí, viz obrázek 5.1:

1. *Data Model* – (Model dat) – je to element, který slouží k definici vstupních dat. V rámci něj je definován element *DataMatrix*, každý sloupec tabulky je reprezentován elementem *DataField*.

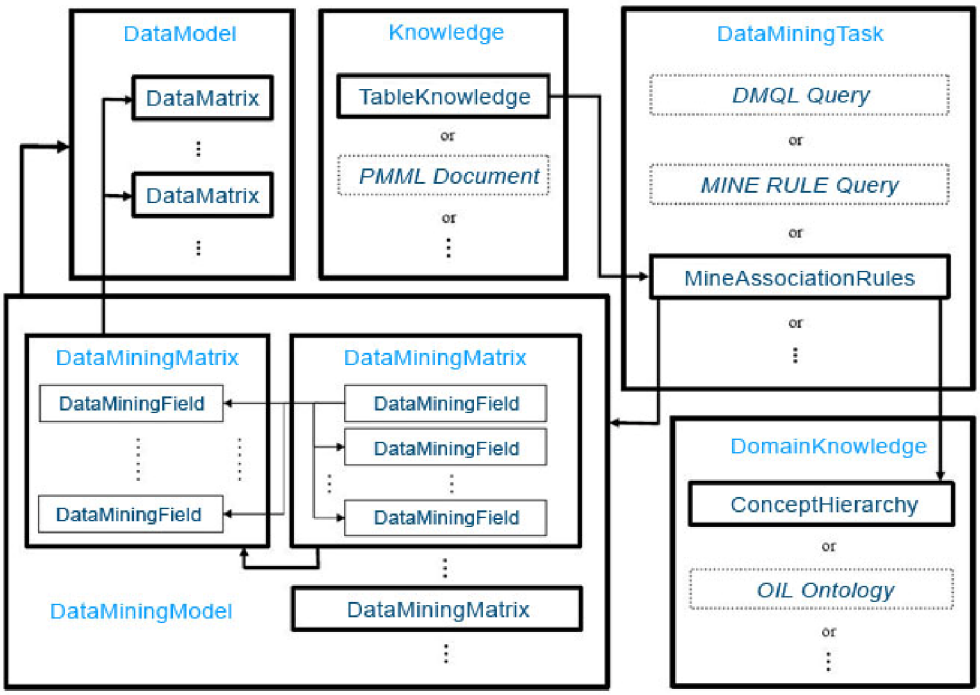
2. *Data Mining Model* – (Dolovací model) – tento element popisuje, která data budou zahrnuta pro dolování, aplikace transformací na sloupce, každý DataMiningModel se musí odkazovat na jeden DataModel.

3. *Domain Knowledge* – (Znalosti o doméně) – element obsahující znalosti o datech, jako jsou vtahy, integritní omezení, obory hodnot apod.

4. *Data Mining Task* – (Dolovací úloha) – element specifikující dolovací funkce, které se budou provádět nad daty, má volnou syntaxi. Každá dolovací funkce má jiné požadavky, podle kterých se bude odvíjet podoba syntaxe

5. *Knowledge* – (Znalosti) – element obsahující vydolované znalosti, poslední element, který má volnou syntaxi. Podle typu získaných znalostí je definována syntaxe, která bude vhodná k uložení daného typu znalostí

6. *Function Pool* – (Funkce) – element definující funkce. Ty mohou být použity při čištění dat a transformacích. Každý *FunctionPool* má svůj název, podle kterého může být jednoznačně identifikován, na nějž se mohou odkazovat jednotlivé elementy. Funkce mohou být interní nebo externí. Interní funkce mají hlavičku i tělo definované v rámci *DMSL* dokumentu. Externí funkce mají v dokumentu definovánu pouze hlavičku, kompletní definice je v externím zdroji.



Obrázek 5.1 Závislost DMSL elementů – převzato z [6]

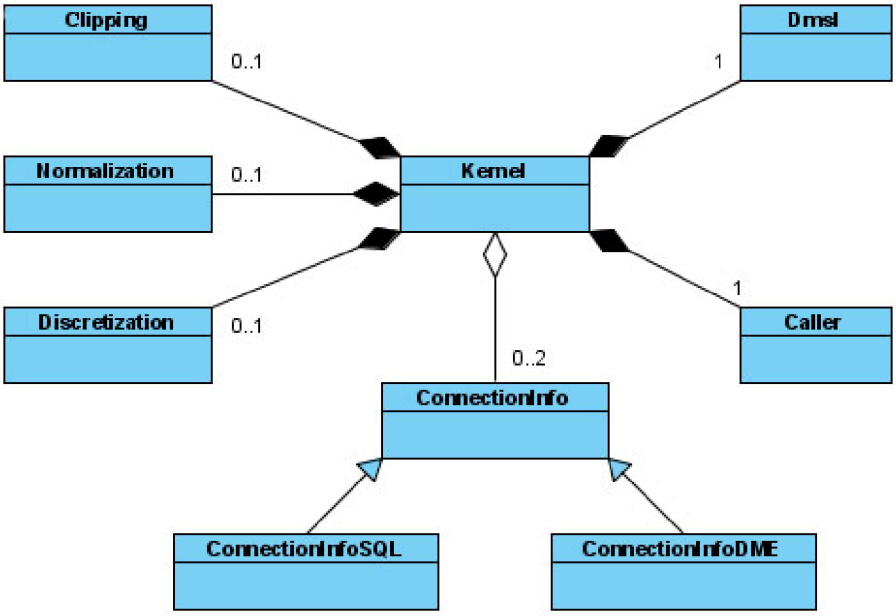
5.3.2 Systém

Jak již bylo zmíněno v předcházejících kapitolách, je tento systém rozdělen do dvou základních částí. První z nich je *Jádro* a druhou částí je *Grafické uživatelské rozhraní*. Další možnou částí jsou samostatné dolovací moduly, ale ty tu popisovat nebudeme. Chápejme tedy pojem systém jako jádro a GUI. Tento systém umožňuje organizovat práci do projektu, předzpracování dat, transformace, zvolit dolovací úlohu, přidávat dolovací moduly, načítat a ukládat do *DMSL*.

Co se týká umístění, byl zaveden systém balíčků s kořenem *cz.vutbr.fit.dataminer*. Jádro používá balíček *cz.vutbr.fit.dataminer.core*. Pro grafické uživatelské rozhraní slouží balíčky umístěné v *cz.vutbr.fit.dataminer.**. Jádro je zkompilováno a zabaleno do *java* archívu (*JAR*). Knihovny, na kterých je jádro závislé, jsou taktéž distribuovány jako *JAR* archívy. Pro datové struktury *DMSL* byl zaveden balíček *cz.vutbr.fit.dataminer.core.system.dmsl.*

5.3.2.1 Jádro

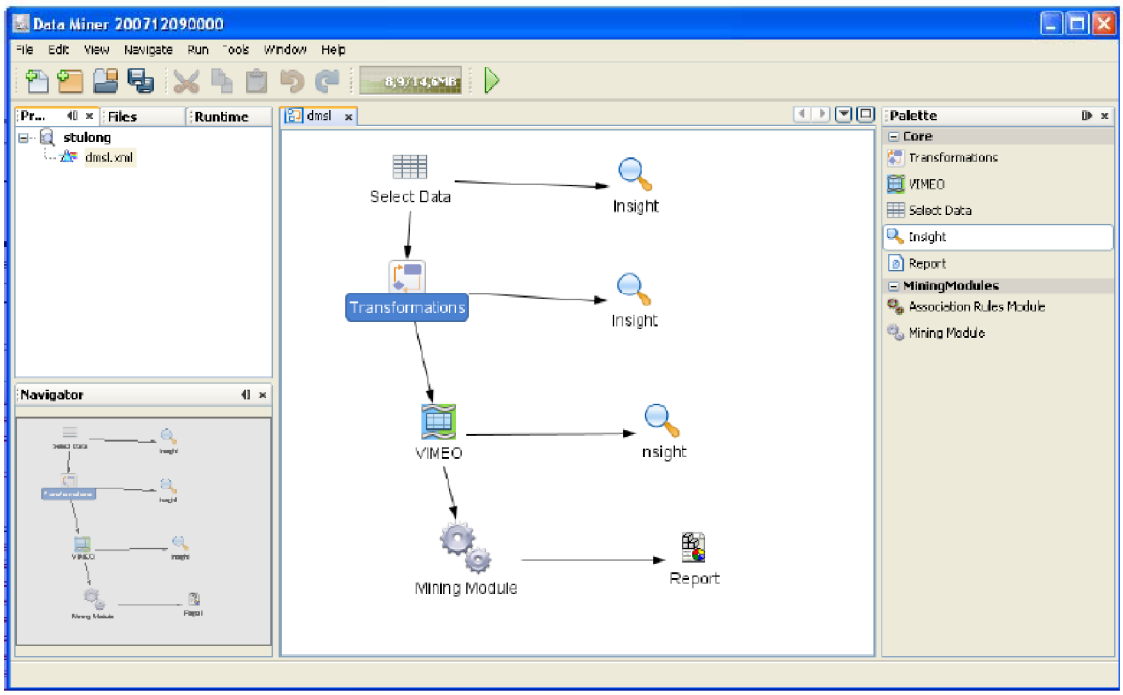
Strukturu jádra popisuje obrázek 5.2. Jádro má svoji centrální třídu *Kernel.java.* Další třídy jsou vázané na tuto třídu. Třída *Dmsl.java* obsahuje datové jednotky, je určena pro správu DMSL dokumentů, slouží tedy jako centrální úložiště dat. Třída má za úkol poskytovat připojení k serveru – jeden typ pro *SQL dotazy*, jeden pro *DME transformace*. Vytvoření připojení nezajišťuje *Kernel*, proto je ve vztahu uvedena agregace. Dále *Kernel* poskytuje *Caller* pro volání zkompilovaných interních funkcí a přístup k datové struktuře *Dmsl.java*. Instance *Clipping*, *Normalization* a *Discretization* se vytvářejí při volání metody, obsluhující spuštění běhu komponenty *Transformations* a zajišťují inicializaci parametrů pro server a jejich odeslání.



Obrázek 5.2 Diagram tříd Kernel – struktura jádra – převzato z [2]

5.3.2.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je nadstavbou nad jádrem. Na obrázku 5.3 je znázorněna grafická předloha tak jak ji ve své práci vytvořil Ing.Gálet [6]. Tvoří část pro zobrazení otevřených projektů a jejich obsahu. Po poklepání na příslušný *DMSL* soubor se otevře editor s paletou elementů - komponent, které lze vkládat do editoru a modifikovat tak samotný dokument. Komponenty umístěné na plátno je nutné propojit šipkami (`Ctrl+click a "tah myší"`). Každá komponenta má své kontextové menu, lze ji otevřít a pracovat s jejím dialogovým oknem, lze ji spustit z kontextového menu položkou Run. Každá komponenta může mít pouze jednu vstupní hranu, a pokud nemá žádnou, pak není komponenta připojena a nejde ji tedy ani spustit ani editovat.



Obrázek 5.3 Grafická nadstavba Ing. Gáleta

5.3.2.3 Komponenty dolovacího procesu

Komponenta Select Data

Tato komponenta umožňuje výběr vstupních dat. Možné je zvolit vstupní data z CSV souboru nebo z databáze. V levé části dialogového okna je zobrazen strom, který zobrazuje databázové schéma a tabulky v něm uložené. Z těchto tabulek může uživatel libovolně vybírat sloupce, ve kterých jsou data, která mají být použita v dolovací úloze. Dále je zde umístěna funkce, která slouží ke vkládání podmínek pro spojování tabulek.

Komponenta Transformations

Tato komponenta umožňuje aplikace ODM transformací, clipping, normalizaci a diskretizaci na vstupních datech. Dále umožňuje nezahrnout do výstupu vstupní sloupce a pomocí funkcí odvodit sloupce nové.

Komponenta 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 tu již zmíněná komponenta Transformations. Komponenta Vimeo zajišťuje přístup k editacím funkcí.

Komponenta Reduce

Tato komponenta umožňuje redukovat počet vstupních dat a také umožňuje rozdělit vstupní data do 2 tabulek pro trénovací fázi a pro testovací fázi, které jsou součástí procesu dolování u klasifikačních a predikčních modulů.

Komponenta Mining Module

Tato komponenta reprezentuje uživatelem vybraný dolovací modul. Funkčnost komponenty závisí na požadavcích vybrané dolovací úlohy.

Komponenta Insight

Tato komponenta může být připojena k ostatním komponentám a zobrazuje obsah vytvořených tabulek, které tyto komponenty při spuštění vytvořily.

Komponenta Report

Tato komponenta je poslední komponentou jak ve výčtu tak i co se týká užití v systému. Zajišťuje zobrazení výsledků. Výsledné zobrazení se liší v závislosti na zvoleném dolovacím modulu.

6 Návrh predikčního modulu

V této kapitole se budu podrobně zabývat návrhem predikčního modulu. Pro svoji implementaci jsem si zvolil druh predikce - *regresi*. Oracle Data Mining nabízí pro regresi algoritmus *SVM – Support Vector Machines*.

6.1 Algoritmus SVM

Support Vector Machines je algoritmus založený na statistickém učení. Tento algoritmus je v rámci ODM využíván k lineární i nelineární regresi, ke klasifikaci a k detekci odlehlých hodnot. Jedná se o silný algoritmus, s pevnou teoretickou základnou, která se opírá o teorii *„Vapnik-Chervonenkis“*. SVM má silné regularizační (sjednocující) vlastnosti. Regularizace rozhoduje o zobecnění modelu pro nová data. Modely SVM fungují na podobných principech jako neuronové sítě a radiální bázové funkce, obě jsou to velmi známé techniky pro dolování. Nicméně žádná z těchto technik není podložená takovými fakty pro regularizaci, jako algoritmus SVM. Při použití algoritmu SVM je rozdíl výsledné kvality zobecnění a jednoduššího učení nad daty značný oproti zmiňovaným tradičním metodám.

SVM je schopna analyzovat komplexní modely dat, obtížnější problematiky jako je klasifikace textu a obrázků, rozpoznávání ručně psaného textu a podobně. Algoritmus SVM pracuje efektivně na množině dat, které mají velký počet atributů, a to i přesto, že například nemá velkou možnost provést fázi učení na velkém vzorku dat. Tento algoritmus nemá žádnou horní hranici pro počet atributů, jediným omezením je zatížení hardwaru. Naproti tomu, tradiční neuronové sítě za těchto okolností nepracují efektivně.

6.1.1 Výhody algoritmu SVM v ODM

Oracle Data Mining má svoji vlastní implementaci metody *SVM*, která využívá spoustu výhod tohoto algoritmu. Poskytuje rozšiřitelnost a použitelnost, které jsou potřebné k vytvoření kvalitního dolovacího systému.

6.1.1.1 Použitelnost

Použitelnost je jednou z nejstěžejnějších vlastností tohoto algoritmu, jelikož na *SVM* bývá často pohlíženo jako na nástroj pro experty. Algoritmus obvykle potřebuje předzpracovaná a optimalizovaná data. Implementace tohoto algoritmu v rámci *ODM* tyto nároky minimalizuje. Uživatel nemusí být expert, aby vytvořil kvalitní SVM model. Předzpracování dat není ve většině případů nutné.

6.1.1.2 Rozšiřitelnost

Pokud se zabýváme velkou množinou dat, je často vyžadováno vzorkování, avšak díky implementaci *SVM* v *ODM* to nutné není, jelikož algoritmus sám využívá rozdělení vzorků dat na několik vrstev k redukci počtu trénovacích dat na potřebný počet. Implementovaný algoritmus je optimalizovaný, vytváří model inkrementálně na malých vzorcích dat, z celkového souboru dat. Model je ve fázi učení tak dlouho, dokud se neblíží právě zpracovávané množině dat, poté se model přizpůsobí novým datům. Proces pokračuje iterativně, dokud nejsou splněny podmínky konvergence.

6.1.1.3 Další výhody algoritmu

Kernel-Based Learning

SVM je „*Kernel-based*" algoritmus. *Kernel* je funkce, která transformuje vstupní data do vícerozměrného prostoru, kde se daná problematika řeší. Funkce Kernel může být *lineární* nebo *nelineární*, v rámci ODM se jedná o *lineární* a *gaussovský* (nelineární) *kernel*. *Funkce lineárního kernelu* redukuje lineární rovnost na originálních atributech z trénovacích dat. *Lineární kernel* pracuje efektivně, pokud má hodně atributů v trénovacích datech. *Gaussovský kernel* transformuje každou položku z trénovacích dat na bod do *n-dimenzionálního prostoru*, kde n je počet položek. Algoritmus usiluje o oddělení bodů do podmnožiny se stejnou (homogenní) cílovou hodnotou.

Active Learning

Aktivní učení je optimalizační metoda pro kontrolu zvětšování modelu a redukování času při jeho vytváření. Bez aktivního učení, SVM model vzrůstá podle toho, jak se zvětšuje množina vstupních dat, ta efektivně omezuje SVM model v rámci malé a střední množiny trénovacích dat (méně jak 100 000 položek). Aktivní učení poskytuje způsob jak překonat toto omezení. Díky aktivnímu učení je možné vytvořit efektivně model na velké množině trénovacích dat. Aktivní učení se využívá jak pro lineární, tak i pro gaussovský kernel. Obzvláště pro gaussovský kernel je výhodné, jelikož nelineární model může rychleji narůstat a mít značné nároky na paměť a další systémové zdroje.

Vylepšování SVM modelu

SVM obsahuje mechanismus, který umí automaticky vybrat nejvhodnější nastavení vstupních parametrů v závislosti na vstupních datech.

6.1.2 Předzpracování dat pro SVM

Algoritmus SVM pracuje nad atributy *numerical,* tedy nad číselnými atributy. Automaticky rozděluje *categorical* data do množiny binárních atributů, každý podle hodnoty kategorie. Například sloupec *pohlaví* ze vstupní tabulky s hodnotami *muž* nebo *žena* by byl transformován do dvou číselných atributů *muž* a *žena*. Nové atributy budou pak nabývat hodnot *1(true)* nebo *0(false)*.

Pokud jsou některé řádky prázdné ve sloupcích s jednoduchým datovým typem (ne typ *nested*), pak je SVM interpretuje jako náhodné hodnoty. Algoritmus automaticky nahradí chybějící *categorical* hodnoty na *mode* (hodnota s největší frekvencí výskytu) a chybějící *numerical* hodnoty na *mean* (aritmetický průměr). Pokud se jedná o sloupce s datovým typem *nested*, pak chybějící data SVM algoritmus interpretuje jako *sparse data* (řídká data). Algoritmus automaticky nahradí *sparse numerical data* za nuly a *categorical sparse data* za nulové vektory.

6.1.2.1 Normalizace

SVM vyžaduje provést normalizaci nad číselnými daty. Normalizace umisťuje hodnoty číselných atributů ve stejném poměru a zamezuje tak atributům s velkým poměrem ovlivňovat řešení. Díky normalizaci je tedy možné předcházet ovlivnění řešení a navíc numerické atributy se dostávají do stejného poměru (0,1) jako rozdělené *categorical* data.

6.1.2.2 SVM a automatické předzpracování

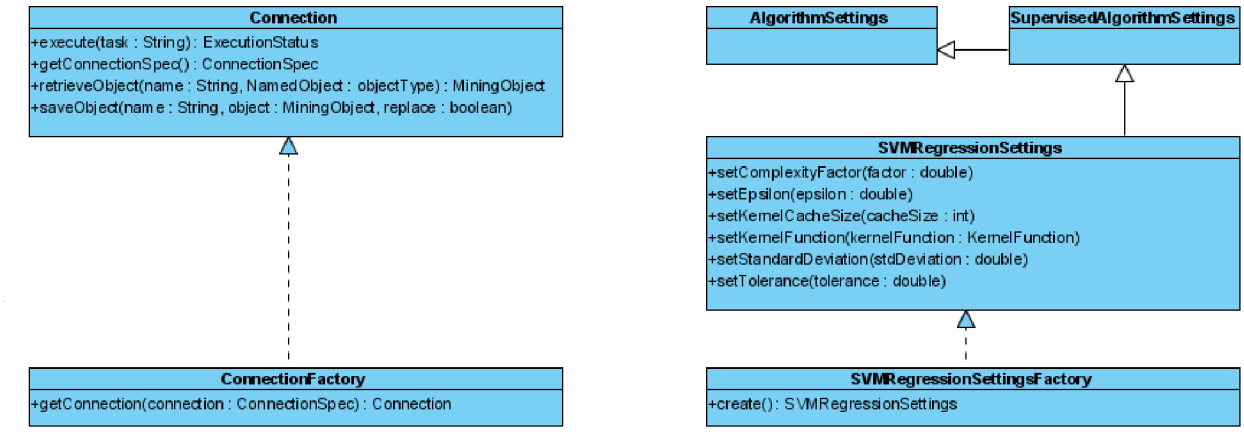
SVM algoritmus umí automaticky ošetřit chybějící hodnoty a transformovat *categorical* data, co ale neumí je normalizace. Tu je nutné provést ručně.

6.1.3 SVM regrese

Algoritmus SVM regrese se snaží nalézt spojitou funkci, takovou aby co největší počet datových bodů ležel ve čtvercovém prostoru o vzdálenosti epsilon. Body, které leží v tomto prostoru, pak nejsou brány jako chybné. Hodnota epsilon je regularizační nastavení pro SVM regresi, ta balancuje na hranici chyby robustnosti modelu dosáhnout nejlepší zobecnění nových dat.

6.1.4 Rozhraní ODM a JavaAPI

Na obrázku 6.1a a 6.1b jsou zobrazeny třídy tvořící rozhraní pro použití regrese pomocí ODM. O vytváření instancí tříd se stará metoda *create()* příslušné *Factory*. Objekt třídy *Connection* poskytuje připojení k „data mining enginu“ a přes tento objekt probíhá ukládání objektů na server, spuštění úlohy a zisk vytvořeného dolovacího modelu. První krok definice úlohy je vytvoření objektu *PhysicalDataSet*, který popisuje strukturu dat určených k dolování, odkazuje prostřednictvím URI na vlastní data. S tímto objektem souvisí objekt *PhysicalAttribute* odpovídající sloupci databázové tabulky. Pro sestavení dolovací úlohy je potřeba nastavit parametry - objekt třídy *BuildSettings* slouží k zachycení vstupních parametrů modelu, k nastavení dolovacího algoritmu a cílového atributu. O nastavení konkrétních parametrů metody se stará objekt *AlgorithmSettings* a s ním úzce související objekt *SVMRegressionSettings*, zde se nastaví potřebné hodnoty jako je funkce kernelu, epsilon, tolerance a další (viz kapitola 7.2.2). Objekt *Task* slouží pro specifikaci činností, které bude v určité fázi DME provádět. Pro vytvoření modelu slouží objekt *BuildTask*, pro testování slouží objekt *TestTask*, pro aplikaci modelu slouží objekt *ApplyTask* a pro výpočet deskriptivních statistik slouží *TestMetricsTask*. Po spuštění úlohy *BuildTask* se na serveru vytvoří model. Objekt *Model* je výsledkem zpracování trénovacích dat. Objekt *ModelDetail* je úzce spojen s objektem *Model* a jedná se o podrobnější informace v rámci konkrétního dolovacího algoritmu, v tomto případě jde o objekt *SVMRegressionModelDetail*. Objekt *TestMetrics* a s ním související objekt *RegressionTestMetrics* zobrazuje vydolované metriky v rámci testování na množině testovacích dat.



Obrázek 6.1a Třídy rozhraní

Obrázek 6.1b Třídy rozhraní

6.2 Vytvoření modulu a integrace do aplikace

Tato kapitola popisuje postup vytvoření a integrace nového modulu, tak jak ji popsal ve své práci Ing. Krásný [2] a Ing. Mader [8], aplikovaná na tento modul.

6.2.1 Vytvoření modulu NetBeans

Celá aplikace DataMiner je vyvíjena v prostředí NetBeans, důležité tedy při instalaci tohoto prostředí je zvolit verzi NetBeans Development, ta jediná totiž umožňuje vývoj NetBeans modulů. Pro nový modul zvolíme projekt *Module*, z kategorie *NetBeans Modules*. Nyní máme dvě možnosti, pokud máme k dispozici všechny zdrojové kódy aplikace, můžeme modul vyvíjet jako součást tohoto celku. V tomto případě zvolíme možnost *Add to module suite*. Pokud máme k dispozici pouze přeloženou aplikaci, musíme nový modul vyvíjet jako samostatný projekt. V tomto případě zvolíme možnost *Standalone module*. Aby měl nový modul přístup k třídám a knihovnám, které jsou potřebné pro jeho vývoj, je nutné nastavit platformu, která odpovídá aktuální verzi aplikace DataMiner. Vybereme možnost přidání nové platformy a zde zvolíme adresář aplikace DataMiner. Automaticky se přidá nová platforma, pro kterou se bude nový modul vyvíjet. Zvolíme název modulu a jako předponu *Code Name Base* použijeme *cz.vutbr.fit.dataminer*. Dokončíme vytváření modulu. Na konec je nutné ještě nastavit ve složce projektu v prostředí NetBeans je položka *Libraries*, zvolíme *Add Module Dependency* a v ní vybereme *Dialogs API*, *dm-api*, *dm-core-wrapper*, *UI Utilities API* a *Utilities API* jsou závislosti potřebné pro vývoj nového modulu. Takto vytvořený a nastavený modul je připraven k implementaci nezbytných tříd a funkcí a následné integraci do aplikace.

6.2.2 Implementace abstraktní třídy MiningPiece

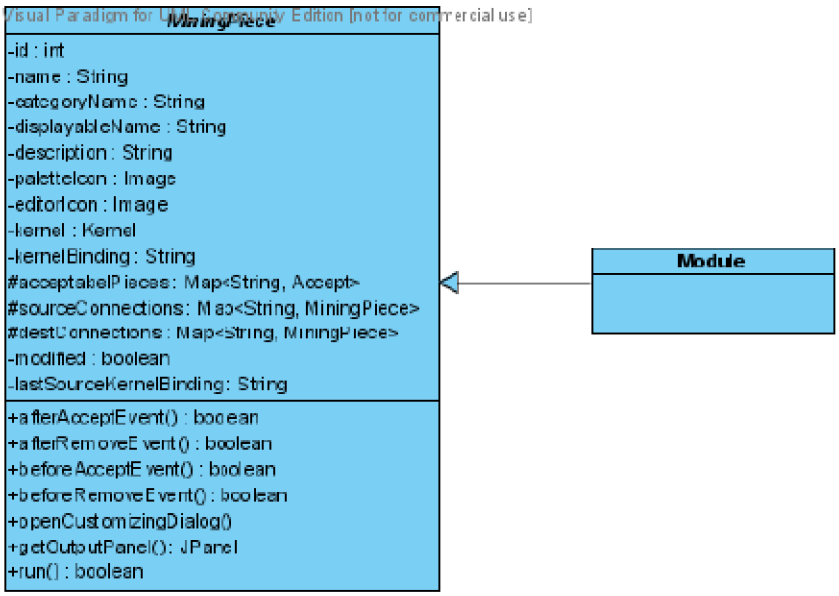
Aby [8] dolovací modul byl přístupný v aplikaci, je potřeba implementovat *MiningPiece* abstraktní třídu, která se nachází v API vrstvě. Každá třída, která dědí od třídy *MiningPiece*, představuje prvek, který lze vložit do procesu dolování dat. Implementovaná třída je pak základní třídou celého modulu V konstruktoru třídy je nutné nastavit parametry zobrazení komponenty v aplikaci – jméno, popis a cesty k ikonám komponenty.

```
setDisplayableName(NbBundle.getMessage(PredictReModule.class, "NAME_PredictReModule"));
setDescription(NbBundle.getMessage(PredictReModule.class, "HINT_PredictReModule"));
setPaletteIcon("cz/vutbr/fit/dataminer/predictremodule/resources/Module16.png");
setEditorIcon("cz/vutbr/fit/dataminer/predictremodule/resources/Module48.png");
```

Poslední dva řádky určují cestu v adresáři modulu k obrázkům, které jsou použity jako ikony modulu v paletě komponent a editoru. První dva říkají, že jméno a popis modulu jsou uloženy v lokalizačním bundle, pod vlastnostmi *NAME_ModuleName* a *HINT_ModuleName*. Aby aplikace nalezla tyto popisy, je nutné do souboru *Bundle.properties*, který se nachází v adresáři modulu, přidat následující řádky.

```
NAME_PredictReModule=Regression
HINT_PredictReModule=Mining module for prediction - regression
```

Třída *MiningPiece* (viz obrázek 6.2) implementuje metody, které jsou popsány níže a které je nutné v našem novém modulu také implementovat.



Obrázek 6.2 Abstraktní třída MiningPiece - převzato [8]

Je nutné implementovat metodu *openCustomizingDialog()*, která zajišťuje otevření dialogu pro nastavení parametrů komponenty. Implementace by neměla vytvářet vlastní okna, ale měla by použít prostředky platformy *NetBeans* pro dialogy, které se starají o zachování jednotného vzhledu a korektní chování. Tento panel by měl obsahovat prvky pro zadání parametrů dolovacího modulu. Metody *beforeAcceptEvent(), afterAcceptEvent(), afterRemoveEvent()* jsou volány jako reakce na provedenou akci v editoru dolovací úlohy. Pokud je požadována reakce modulu na akci, je implementována zde. Např. metoda *afterAcceptEvent()* do DMSL vloží základ elementů *DataMiningTask* a *Knowledge*. Metoda *getOutputPanel()* je volána při otevření standardní komponenty *Report*. Měla by zajišťovat vytvoření panelu, ve kterém se budou zobrazovat výsledky dolovacího procesu. Metoda *run()* spouští akci komponenty. V případě dolovacího modulu by v této komponentě měl proběhnout samotný proces aplikace algoritmu získání znalostí na vstupní data.

6.2.3 Integrace do aplikace

Dále [6] je potřeba ke konečné integraci modulu do aplikace, do které se instaluje, jej zaregistrovat do souboru *layer.xml*. Tento soubor představuje virtuální systém souborů, který se z různých modulů integruje do jednoho celku. To znamená, že záznamy, které tento modul vytvoří, budou viditelné i z ostatních modulů. Ostatní moduly o dolovacím modulu nic neví, dokonce ani nemají přístup k jeho třídám, i přesto, že běží v rámci jednoho virtuálního stroje *Javy*. O vytváření instancí se stará *System FileSystem*, který poskytuje instance ostatním modulům.

Aby se dolovací modul objevil v *MiningPieceRegistry* (a tím i v paletě komponent), je potřeba vytvořit pár záznamů v *layer.xml*:

```
<filesystem>
 <folder name="MiningPieceRegistry">
  <folder name="MiningModules">
   <file name="cz-vutbr-fit-dataminer-predictremodule-PredictReModule.instance"/>
  </folder>
 </folder>
```

Tento záznam vytvoří instanci třídy modulu. A zaregistruje modul v *MiningPieceRegistry* do kategorie Mining Modules. Komponenta bude zobrazena v paletě Mining Modules. Další skupina záznamů říká, které komponenty mohou modulu předcházet, a které jej mohou následovat. V případě tohoto modulu regrese *MiningPieceConfig* říká, že tento modul může být připojen pouze na komponentu *Reduce*, a to proto, jelikož modul potřebuje dvě vstupní tabulky, jednu tabulku pro učení, druhou tabulku pro testování. To komponenta *Reduce* nabízí. Tento predikční modul může následovat jediná komponenta a to komponenta pro zobrazení výsledků *Report*.

```
<folder name="MiningPieceConfig">
     <folder name="cz-vutbr-fit-dataminer-predictremodule-PredictReModule">
      <folder name="accept">
            <file name="cz-vutbr-fit-dataminer-editor-palette-items-Reduce"/>
      </folder>
     </folder>
     <folder name="cz-vutbr-fit-dataminer-editor-palette-items-Report">
      <folder name="accept">
           <file name="cz-vutbr-fit-dataminer-predictremodule-PredictReModule"/>
      </folder>
     </folder>
</folder>
</filesystem>
```

7 Implementace modulu Regrese

Tato kapitola popisuje vlastní implementaci predikčního modulu regrese. V rámci implementace tohoto modulu bylo nutné navrhnout *DMSL* elementy *DataMiningTask* a *Knowledge*, implementovat samotnou dolovací úlohu, navrhnout a implementovat Report a implementovat možnost aplikační fáze predikce.

7.1 Návrh DMSL

V rámci předešlých prací byly vytvořeny elementy *DataMiningTask*, specifikující dolovací úlohu a element *Knowledge*, obsahující vydolované znalosti. Oba tyto elementy mají volnou syntaxi, jelikož každá dolovací funkce má jiné požadavky a jiné výsledky. V následujících podkapitolách je uveden návrh syntaxe elementů pro predikční modul regrese, algoritmu *Support Vector Machines*.

7.1.1 DataMiningTask

Element *DataMiningTask* specifikuje dolovací úlohu, v původním návrhu je jeho syntaxe elementu volná. Každý typ úlohy může mít svoje specifické požadavky na parametry, které je zapotřebí ukládat. Ty se mohou lišit i ve stejném typu úlohy podle vybraného algoritmu. Jediný povinný atribut je *language*. Ten udává, v jakém jazyce je element *DataMiningTask* popsán. Seznam povinných atributů rozšířil ve své práci Ing. Krásný. Zavedl povinný atribut *name*, který slouží jako propojení s dolovacím modulem.

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

Element *Regression* pro predikční modul je navržen v rámci elementu *DataMiningTask*. Uchovává v sobě informace o vstupních datech a parametrech, které jsou potřeba k definici regrese algoritmem Support Vector Machines.

```
<!ELEMENT Regression (InputDataType, Parametres)>
<!ATTLIST Regression algorithm CDATA #REQUIRED>
```

V elementu *InputDataType* jsou uloženy informace určující typ vstupních dat. Atribut *tableKey* uchovává identifikátor tabulky, atribut *target* uchovává název sloupce – cílový atribut, pro který bude vytvořena predikce.

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

V elementu *Parametres* jsou uloženy parametry pro dolování. Elementy *KernelFunction*, *Tolerance*, *ActiveLearning* jsou povinné, zbylé elementy *Epsilon*, *ComplexityFactor*, *StandartDeviation*, *CacheSize* nutné nejsou.

```
<!ELEMENT Parametres (KernelFunction, Tolerance, ActiveLearning, Epsilon?,
ComplexityFactor?, StandartDeviation?, CacheSize?)>

<!ELEMENT KernelFunction (Gaussian|Linear|System determined)>

<!ELEMENT Tolerance (%REAL-NUMBER;)>

<!ELEMENT ActiveLearning (true|false)>

<!ELEMENT Epsilon (%REAL-NUMBER;)>

<!ELEMENT ComplexityFactor (%REAL-NUMBER;)>

<!ELEMENT StandartDeviation (%REAL-NUMBER;)>

<!ELEMENT CacheSize (%INT-NUMBER;)>
```

7.1.2 Knowledge

Element *Knowledge* specifikuje vydolované znalosti, v původním návrhu je jeho syntaxe elementu volná. Každá metoda má jiné vydolované znalosti, které je zapotřebí ukládat. Ty se mohou lišit i ve stejném typu úlohy podle vybraného algoritmu nebo podle vybrané dolovací funkce. Jediný povinný atribut je *language*. Ten udává, v jakém jazyce je element *Knowledge* popsán. Seznam povinných atributů rozšířil ve své práci Ing. Krásný, zavedl povinný atribut *name*, který slouží jako propojení s dolovacím modulem, který dané znalosti získal.

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

Stejně tak, jak je proces predikce rozdělen na 3 fáze, je rozdělen i element *Knowledge*. Ten pomocí elementů *BuildTask*, *TestTask*, *ApplyTask* uchovává informace o jednotlivých fázích procesu a získaných znalostí v rámci každé fáze.

7.1.2.1 BuildTask

Element *BuildTask* obsahuje informace získané během fáze učení. Uchovává název úlohy *name*, jméno modelu *model*, název vstupní tabulky pro učení *input* a informace o vytvoření a době trvání úlohy *start*, *end, status*.

```
<!ELEMENT BuildTask Model>
<!ATTLIST BuildTask name CDATA #REQUIRED
                    model CDATA #REQUIRED
                    input CDATA #REQUIRED
                    start %DATE-TIME; #REQUIRED
                    end %DATE-TIME; #REQUIRED
                    status CDATA; #REQUIRED >
```

Element *BuildTask* dále obsahuje element *Model*, ten popisuje informace o vytvořeném modelu, jeho název *name* a informaci zda je model lineární či nikoliv *isLinear*. Konkrétní informace o modelu popisuje až element *ModelSignatureAttribute*, který uchovává název sloupce, který je do modelu zahrnut *name*, jeho datový typ *dataType* a dolovací typ *miningType*. Pokud je model lineární, je nutné uchovávat získané atributy *value* a koeficienty *coefficient*, což jsou hodnoty parametrů regresní funkce.

```
<!ELEMENT Model (ModelSignatureAttribute)>
<!ATTLIST Model name CDATA #REQUIRED
                bias %REAL-NUMBER #IMPLIED
                isLinear (true|false) #REQUIRED >

<!ELEMENT ModelSignatureAttribute (AttributeProperties*) >
<!ATTLIST ModelSignatureAttribute name CDATA #REQUIRED
                                  dataType CDATA #REQUIRED
                                  miningType CDATA #REQUIRED >

<!ELEMENT AttributeProperties EMPTY>
<!ATTLIST AttributeProperties value CDATA #REQUIRED
                              coefficient %REAL-NUMBER; #REQUIRED >
```

7.1.2.2 TestTask

Podobnou syntaxi jako element *BuildTask* má element *TestTask*, ten uchovává informace získané během fáze testování. Uchovává tedy název úlohy *name*, jméno modelu *model*, název vstupní tabulky pro testování *input*, navíc obsahuje název výstupní tabulky, kde jsou uloženy otestovaná data i s predikovanými pro porovnání *output* a informace o vytvoření a době trvání úlohy *start*, *end, status*.

```
<!ELEMENT TestTask (Metrics)>
<!ATTLIST TestTask name CDATA #REQUIRED
                   model CDATA #REQUIRED
                   input CDATA #REQUIRED
                   output CDATA #REQUIRED
                   start %DATE-TIME; #REQUIRED
                   end %DATE-TIME; #REQUIRED
                   status CDATA; #REQUIRED >
```

Element *TestTask* obsahuje element *Metrics*, ve kterém jsou uloženy získané metriky v průběhu testování. *MAE* – Mean Absolute Error, *MAV* – Mean Actual Value, *MPV* – Mean Predicted Value, *RMSE* – Root Mean Squared Error více viz 7.2.4.2.

```
<!ELEMENT Metrics EMPTY>
<!ATTLIST Metrics MAE %REAL-NUMBER; #REQUIRED
                  MAV %REAL-NUMBER; #REQUIRED
                  MPV %REAL-NUMBER; #REQUIRED
                   RMSE %REAL-NUMBER; #REQUIRED >
```

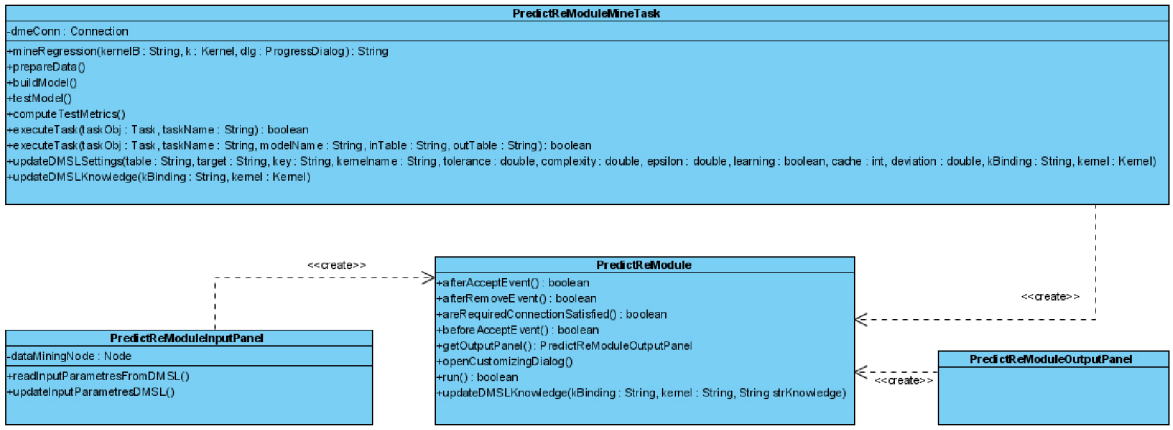
7.1.2.3 ApplyTask

Poslední fáze procesu je aplikační fáze, ta je nepovinná. Informace získané v rámci této fáze jsou uchovány v elementu *ApplyTask*. Je zde uložen název úlohy *name*, jméno modelu *model*, název vstupní tabulky pro aplikaci *input*, název výstupní tabulky, kde jsou uloženy predikovaná data *output* a informace o vytvoření a době trvání úlohy *start*, *end, status*.

```
<!ELEMENT ApplyTask EMPTY>
<!ATTLIST ApplyTask name CDATA #REQUIRED
                    model CDATA #REQUIRED
                    input CDATA #REQUIRED
                    output CDATA #REQUIRED
                    start %DATE-TIME; #REQUIRED
                    end %DATE-TIME; #REQUIRED
                    status CDATA; #REQUIRED >
```

7.2 Implementace modulu

Na obrázku 7.1 je zobrazena struktura predikčního modulu. Nejdůležitější třídou celého modulu je třída *PredictReModule*, zajišťuje komunikaci s jádrem a je přístupovým bodem ke všem funkcím modulu. Jak už bylo popsáno v kapitole 6.2.1 a 6.2.2 je nutné, aby tato třída implementovala abstraktní třídu *MiningPiece* (každá komponenta grafu tuto třídu musí implementovat). O nastavení jména modulu, cest k ikonám a názvů popisků se stará *lokalizační bundle* uvedený v konstruktoru třídy. V třídě *PredictReModule* jsou základní metody spouštějící uživatelské funkce mimo jiné přístup k dialogu pro zadávání vstupních parametrů (implementuje třída *PredicReModuleInputPanel*), metoda *run()*, volající samotný dolovací proces (implementuje třída *PredictReModuleMineTask*) a také přístup k dialogu pro zobrazení výsledků – report (implementuje třída *PredictReModuleOutputPanel*). Velký důraz je kladen na správnost a konzistentnost ukládání údajů do DMSL. V jakékoliv fázi musí být možné uložit úlohu do DMSL aniž by se nějak narušila struktura a konzistence. DMSL figuruje jako jediný prostředek k uchování informací (metadat o dolování).



Obrázek 7.1 Struktura predikčního modulu regrese

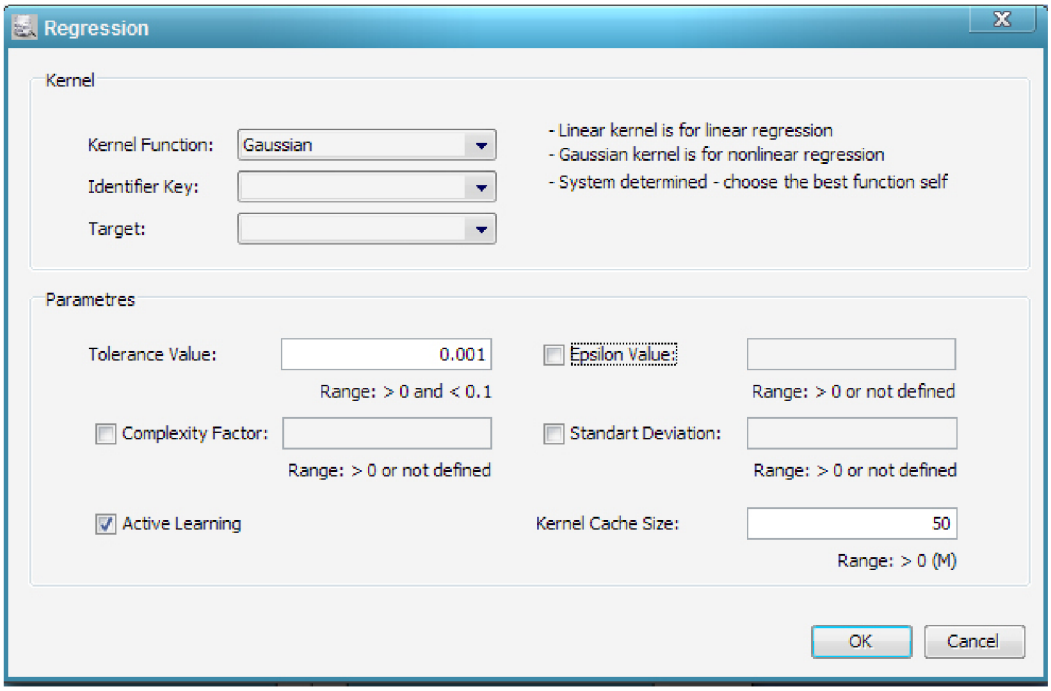
7.2.1 Vložení modulu do grafu

Když vložíme modul do grafu, musíme zajistit, aby se vytvořil záznam v DMSL, o to se stará metoda *afterAcceptEvent()*. V DMSL se vytvoří elementy *DataMiningTask* a *Knowledge* v předdefinované podobě, která je nastavena. Pokud je modul odstraněn, je nutné opět provést změny v DMSL, to má na starost metoda *afterRemoveEvent()*. Následující kód zobrazuje záznamy v DMSL po vložení nového predikčního modulu:

```
<DataMiningTask  language="XML" name="DM122_PredictReModule1" type="Regression">
   <Regression  algorithm="SVM">
      <InputDataType  tableKey="null" target="null"></InputDataType>
      <Parametres >
            <KernelFunction >System determined </KernelFunction>
            <Tolerance >0.001 </Tolerance>
            <ActiveLearning >True </ActiveLearning>
      </Parametres>
   </Regression>
</DataMiningTask>
<Knowledge  language="XML" name="DM122_PredictReModule1" type="Regression">
      <BuildTask name="null" model="null" input="null" start="null" end="null"
      status="null"></BuildTask>
      <TestTask name="null" model="null" input="null" output="null" start="null"
      end="null" status="null"><TestTask>
      <ApplyTask name="null" model="null" input="null" output="null" start="null"
      end="null" status="null"></ApplyTask>
</Knowledge>";
```

7.2.2 Panel parametrů

Panel parametrů je volán metodou *openCustomizingDialog()*, o získání a zobrazení jednotlivých parametrů se stará třída *PredictReModuleInputPanel()*. Pokud uživatel korektně vyplní hodnoty vstupních parametrů, hodnoty se po stisknutí tlačítka Ok ihned uloží do DMSL. Pokud byl otevřen již existující projekt, pak se stisknutím tlačítka Ok vymažou předešlé parametry nastavení, vydolované znalosti a uloží se nové hodnoty vstupních parametrů.



Obrázek 7.2 Dialog vstupních parametrů

Obrázek 7.2 zobrazuje dialogové okno pro zadávání vstupních parametrů, jednotlivé parametry budou popsány následovně:

Kernel Function - jedná se o Kernel funkci pro způsob dolování, možnosti výběru jsou následující:

a) **Linear** – pro lineární regresi (pokud má trénovací množina dat mnoho atributů > 100)
b) **Gaussian** – pro nelineární regresi
c) **System determined** - dolovací modul sám určí v závislosti na vstupních datech, kterou ze dvou předešlých Kernel funkcí zvolí

Identifier Key – tato položka určuje identifikátor (identifikační klíč) vstupní tabulky

Target – tato položka určuje cílový atribut, který chceme predikovat. Jelikož se jedná o predikci, lze zvolit pouze položky, které mají dolovací typ *numerical*.

Tolerance Value – parametr udává měřítko pro dokončení trénování modelu, (hodnota musí být v intervalu I=(0, 0.1)), standardně je nastavena na hodnotu 0,001

Complexity Factor - je regularizační parametr, nastavující rovnováhu složitosti modelu na úkor robustnosti/velikosti modelu za účelem dosáhnout generalizace nových dat. SVM užívá datově řízený

přístup k nálezu správné hodnoty Complexity Factor. Parametr je volitelný, (pokud je zvolen, hodnota musí být kladné reálné číslo)

Active Learning – parametr aktivního učení je volitelný parametr, využívá se u obou funkcí Kernelu, ale stěžejní je pro Gaussovský. Standardně je tento parametr zapnutý.

Epsilon Value – je regularizační parametr regrese, podobný parametru Complexity Factor. Tento parametr specifikuje povolenou odchylku od hranice regresní funkce v datech. Jedná se o volitelný parametr, (pokud je zvolen, hodnota musí být kladné reálné číslo)

Standart Deviation – SVM užívá datově řízený přístup k nálezu správné hodnoty Standart Deviation která je ve stejném poměru jako vzdálenost mezi obvyklými případy. Jedná se o volitelný parametr, lze zvolit pouze, pokud je zvolena Kernel funkce Gaussian, (pokud je zvolen, hodnota musí být kladné reálné číslo)

Kernel Cache Size – velikost paměti alokované Kernelem v průběhu vytváření modelu, standardní nastavení je 50MB, tento parametr je použit pouze, pokud je zvolena Kernel funkce Gaussian, (hodnota musí být celé kladné číslo)

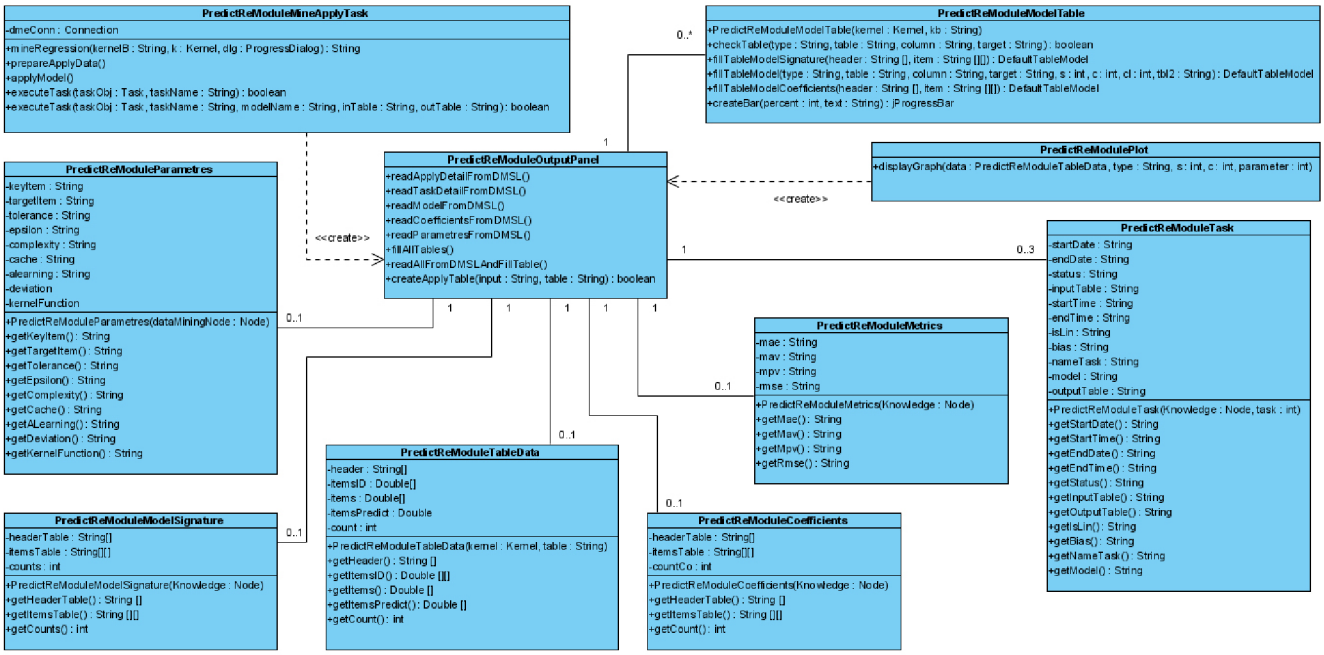
7.2.3 Spuštění procesu dolování

O spuštění procesu dolování se stará metoda *run()*. Třída zajišťující dolovací proces se nazývá *PredictReModuleMineTask*. Té je předán celý element *DataMiningTask*, ze kterého získá dolovací proces vstupní informace. Vstupem pro dolovací úlohu jsou dvě tabulky získané z předešlé komponenty *Reduce*, která pro tento modul rozdělí vstupní tabulku na tabulku pro učení a tabulku pro testování. Nejprve se provede nastavení parametrů, vytvoří se model ve fázi učení, poté se model otestuje v rámci testovací fáze. Po dokončení dolování jsou vydolované znalosti předány hlavní třídě *PredictReModule*, která znalosti uloží do DMSL.

7.2.4 Prezentace výsledků – Report

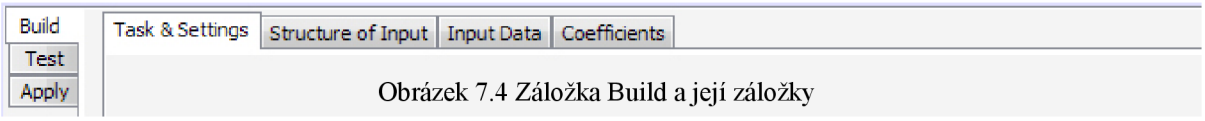
O prezentaci vydolovaných znalostí a informací s nimi spojenými, se stará komponenta *Report.* Pro každý dolovací modul je nutné mít vlastní podobu tohoto panelu, jelikož každá metoda dolování má odlišné výsledné znalosti nebo jejich formu. Proto v tomto případě výstupní panel implementuje třída *PredictReModuleOutputPanel*, která je volána z hlavní třídy *PredictReModule* metodou *getOutputPanel()*. Struktura zobrazování výsledků dolování je ilustrována na obrázku 7.3.

Po otevření panelu se uživateli zobrazí panel se třemi záložkami. Každá záložka popisuje jednotlivé fáze *Build - BuildTask*, *Test - TestTask* a *Apply - ApplyTask*. Podrobněji se každé záložce věnují následující podkapitoly. Informace o daných úlohách implementuje třída *PredictReModuleTask*. Třída *PredictReModuleModelTable* implementuje vstupní model dat pro všechny zobrazované tabulky.



Obrázek 7.3 Implementace zobrazení výsledků dolování

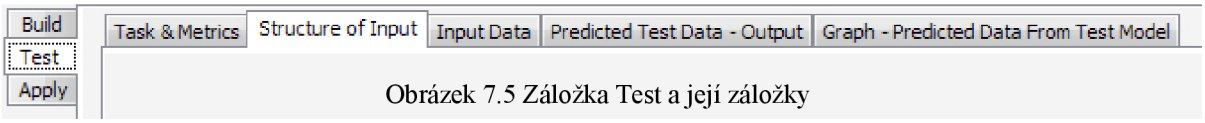
7.2.4.1 Prezentace výsledků – záložka Build – Trénovací úloha



Obrázek 7.4 Záložka Build a její záložky

Záložka *Build* odpovídá trénovací fázi. V této fázi se nastaví hodnoty vstupních parametrů z DMSL a vytvoří se model, tyto informace popisuje záložka *Task & Settings*. Jsou zde uvedeny vstupní parametry, informace o vytvořené úloze trénování, zda proběhla úspěšně, jak dlouho trvala a podobně. Informace o nastavení implementuje třída *PredictReModuleParametres*. Dále je zde tabulka popisující vytvořený model. Ten implementuje třída *PredictReModuleModelSignature*. Další záložka *Structure of Input* zobrazuje tabulku, která popisuje metadata vstupní tabulky pro trénování. Záložka *Input Data* zobrazuje tabulku s daty ze vstupní tabulky pro učení a poslední záložka *Coefficients* zobrazuje koeficienty vstupních atributů, ale jen u lineární regrese. Informace v této záložce obsažené implementuje třída *PredictReModuleCoefficients*.

7.2.4.2 Prezentace výsledků – záložka Test – Testovací úloha



Obrázek 7.5 Záložka Test a její záložky

Záložka *Test* odpovídá testovací fázi. V této fázi se vytvořený model otestuje na vstupní testovací tabulce. V tabulce jsou hodnoty atributu, který chceme predikovat známy, provede se predikce a hodnoty se porovnávají. Záložka *Task & Metrics* zobrazuje informace o proběhlé úloze testování a také vydolované metriky, ty implementuje třída *PredictReModuleMetrics*. Záložka *Structure of Input* zobrazuje tabulku, která popisuje metadata vstupní tabulky pro testování. Záložka *Input Data* zobrazuje tabulku s daty ze vstupní tabulky pro testování. Záložka *Predicted Test Data-Output* zobrazuje tabulku s porovnáním známých hodnot predikovaného atributu a predikovaných hodnot, jejich přesnost a rozdíl. V poslední záložce *Graph-Predicted Data From Test Model* je možné spustit několik grafů popisujících výstupní data. Graf implementuje třída *PredictReModulePlot*. Vstupní data pro grafy implementuje třída *PredictReModuleTableData*.

Testovací metriky

Root Mean Squared Error (RMSE)

Jedná se o statistický údaj, jehož hodnotu popisuje následující rovnice: $RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$
a SQL příkaz:

```
SQRT(AVG((predicted_value - actual_value) * (predicted_value - actual_value)))
```

Mean Absolute Error (MAE)

Jedná se o statistický údaj, jehož hodnotu popisuje následující rovnice: $MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$
a SQL příkaz:

```
AVG(ABS(predicted_value - actual_value))
```

Mean Actual Value (MAV)

Jedná se o údaj udávající průměr všech aktuálních hodnot: $MAV = \frac{1}{n}\sum_{j=1}^{n}\hat{y}_j$
a SQL příkaz:

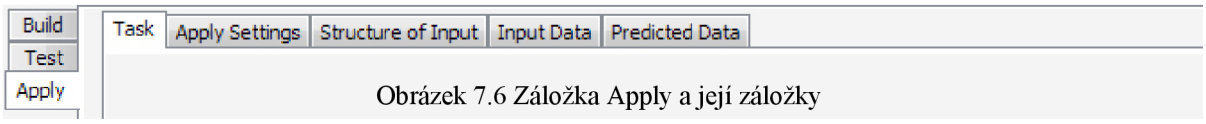
```
AVG(actual_value)
```

Mean Predicted Value (MPV)

Jedná se o údaj udávající průměr všech predikovaných hodnot: $MPV = \frac{1}{n}\sum_{j=1}^{n} y_j$
a SQL příkaz:

```
AVG(predicted_value)
```

7.2.4.3 Prezentace výsledků – záložka Apply – Aplikační úloha



Obrázek 7.6 Záložka Apply a její záložky

Poslední záložka *Apply* odpovídá aplikační fázi. Tato fáze není spuštěna v rámci vytvoření dolovací úlohy, je to volitelná fáze a je možné ji spustit zde z reportu. O implementaci aplikační úlohy se stará třída *PredictReMineApplyTask*. V záložce *Apply Settings* je možné zvolit vstupní tabulku přímo ze serveru nebo vytvořit prázdnou tabulku pro manuální vkládání dat. Po vybrání/vytvoření vstupní tabulky se automaticky zapnou záložky *Structure of Input* a *Input Data*. Ty zobrazují informace podobně jak v předešlých záložkách (*Build*, *Test*). V záložce *Input Data*, pokud byla tabulka vytvořena pro manuální vkládání, je možné přidávat jednotlivá nová data, pro která chceme provést predikci. Pro spuštění aplikační úlohy, je nutné kliknout na tlačítko *Run Apply Task*. Pokud úloha proběhne v pořádku, aktivuje se i poslední záložka *Predicted Data* se získanými výsledky.

8 Ukázkové příklady a Grafické uživatelské prostředí

Tato kapitola demonstruje užití predikčního modulu na dvou typech příkladů. První příklad zobrazuje použití nelineární regrese a příklad druhý zobrazuje lineární regresi. Mimo jiné tak popisuje grafický vzhled celého modulu a jeho částí, převážně prezentaci vydolovaných výsledků. Je zde popsán celý proces od založení nového projektu, až po prezentaci získaných znalostí a provedení aplikační části. Pro oba tyto příklady jsou použita data (SH schéma - SalesHistory) z ukázkové databáze firmy Oracle. Jedná se o data zabývající se historií prodeje nemovitostí a informace o jejich kupcích.

8.1 Nelineární regrese

8.1.1 Vytvoření nového projektu

K vytvoření projektu zvolíme File → New Project a vybereme položku Data mining Project. V následujícím kroku je nutné vyplnit název projektu a jeho umístění. V poslední řadě je nutné vyplnit parametry pro připojení k databázovému serveru.

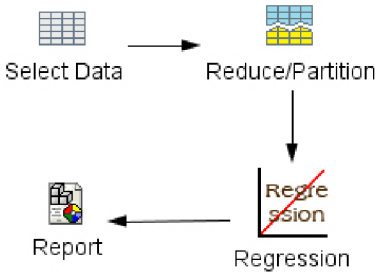
Příklad parametrů pro přístup na server:

```
Login: dmuser5
Heslo: dmuser5
URL: jdbc:oracle:thin:@berta.fit.vutbr.cz:1521:STUD
```

Po zadání parametrů pro připojení je vhodné otestovat připojení, je-li vše v pořádku, můžeme pokračovat dále. Po kliknutí na soubor dmsl.xml v průzkumníku projektu se otevře pracovní plocha a paleta komponent, jednotlivé komponenty lze použít pro definici dolovacího procesu.

8.1.2 Nadefinování komponent v grafu

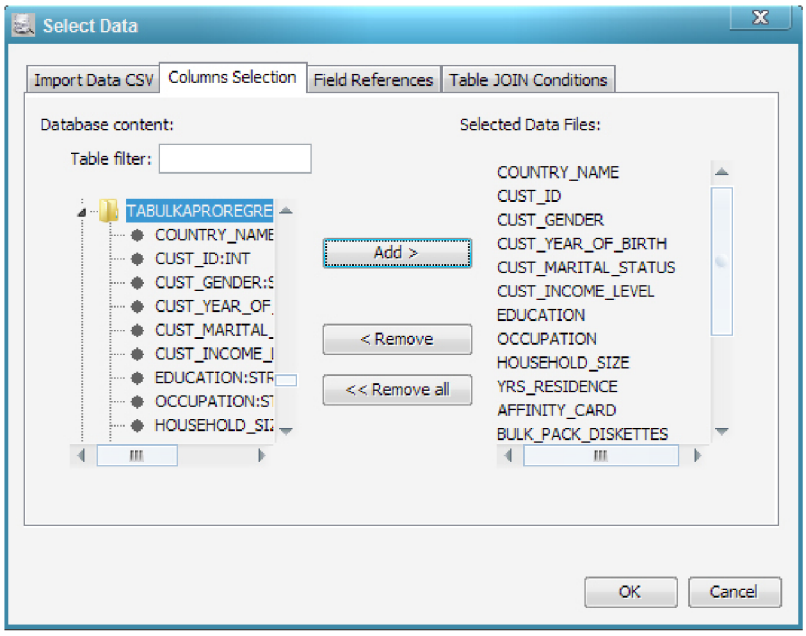
Na obrázku 8.1 jsou zobrazeny komponenty postačující k vytvoření správně fungující dolovací úlohy. Komponenta *SelectData* slouží k výběru vstupních dat, lze vybrat z tabulek dostupných na serveru nebo naimportovat data z csv souboru. Důležité je, aby komponenta *Reduce* předcházela samotný predikční modul a aby byla vůbec použita. Bez ní by byl modul nefunkční, očekává totiž na vstupu dvě tabulky a to tabulku vstupních dat pro fázi učení a tabulku vstupních dat pro fázi testování, to zajišťuje právě komponenta *Reduce*, která tabulku ze *SelectData* rozdělí podle nastavených proporcí právě na zmiňované dvě tabulky. Komponenta *Regression* je samotný dolovací modul a pokud chceme zobrazit vydolované znalosti, pak je nutné za tento modul připojit komponentu *Report*.



Obrázek 8.1 Komponenty v grafu

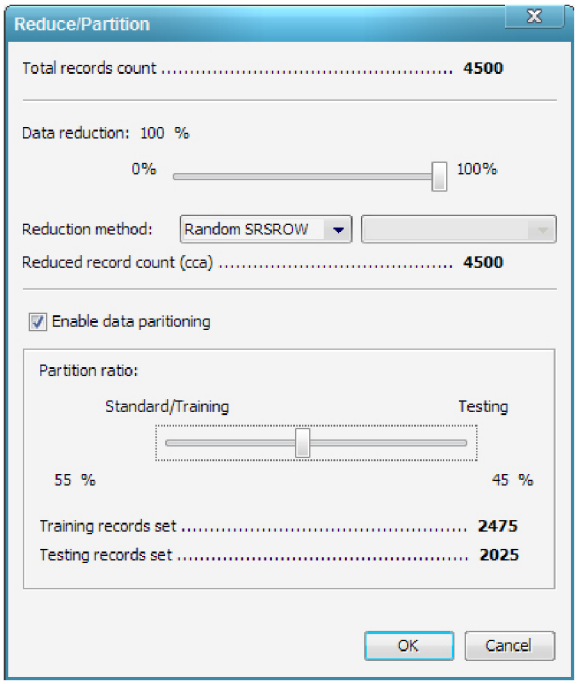
8.1.3 Vstupní data a parametry

V komponentě *SelectData* zvolíme záložku *Columns Selection* a v levé části okna najdeme tabulku TABULKAPROREGRESI (zde jsou uloženy již zmiňovaná data Sales History). Přesuneme všechny položky/sloupce na pravou stranu do výběru sloupců se kterými budeme pracovat. Viz obrázek 8.2. Klikneme na tlačítko Ok. Vrátíme se na graf s komponentami, klikneme pravým tlačítkem na komponentu *SelectData* a zvolíme položku Run, proběhne výběr daných položek z tabulky.



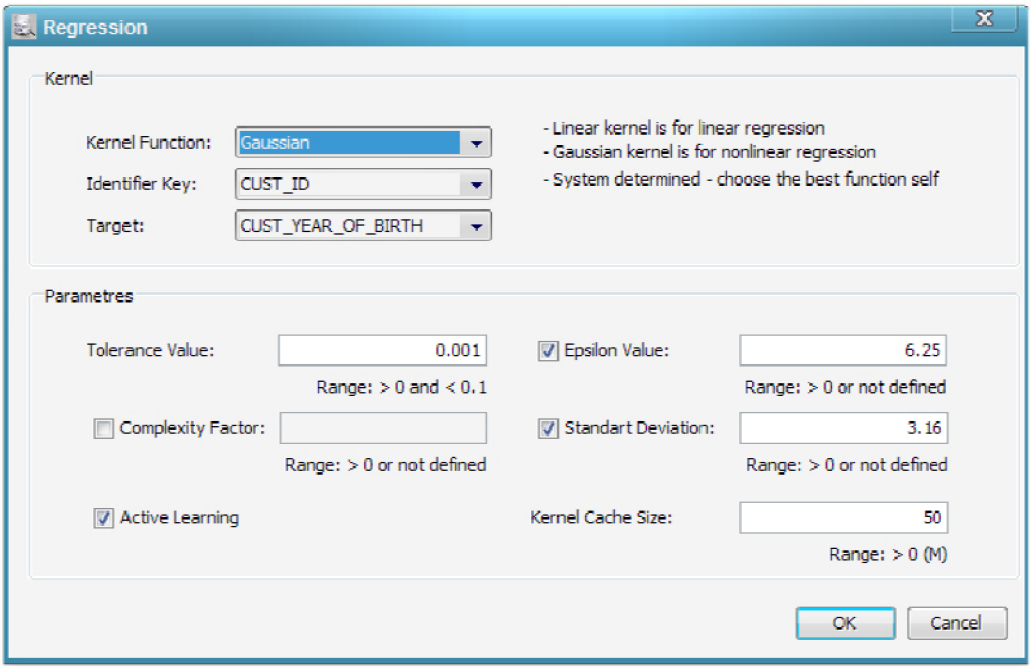
Obrázek 8.2 Komponenta SelectData – výběr vstupních dat

Nyní je potřeba rozdělit získaná data ze zvolené tabulky do dvou tabulek. Otevřeme komponentu *Reduce* obrázek 8.3. Ta umožňuje nejen rozdělení vstupních dat do dvou tabulek podle zvoleného poměru, ale i možnost redukovat počet vstupních dat. Zvolíme pro nás vhodné nastavení a pokračujeme dále.



Obrázek 8.3 Komponenta Reduce – redukce a rozdělení dat

Nyní se již dostáváme k samotnému predikčnímu modulu. Otevřeme komponentu *Regression*, otevře se panel pro zadání vstupních parametrů, obrázek 8.4. Zvolíme *Kernel funkci Gaussian*, jelikož chceme nelineární regresi (pokud uživatel nemá velké zkušenosti s vytvářením dolovacích úloh je více než vhodné zvolit položku *Systém determined*, která způsobí, že systém zvolí ideální nastavení automaticky v závislosti na vstupních datech). Položka *identifier key* je jednoznačný identifikátor v rámci dat, což je primární klíč z tabulky. Položka *Target* označuje sloupec/atribut, který hodláme predikovat (musí to být atribut, který má dolovací typ *numerical*, jelikož regrese pracuje nad spojitými číselnými daty). Ostatní parametry a jejich hodnoty zadejte například následovně (význam parametrů je popsán v kapitole 7.2.2), výhodou je že pokud volitelné parametry nezadáte, systém dokáže jejich hodnoty určit co nejefektivněji v závislosti na vstupních datech a zpětně je doplní do DMSL. Klikneme na tlačítko Ok a pokud je vše zadáno správně můžeme začít dolovat. V grafu klikneme na daný modul pravým tlačítkem a zvolíme Run, tím se spustí dolovací úloha. Dokud probíhá dolování je zobrazen progress dialog informující o činnosti úlohy. Jakmile je úloha dokončena, progress dialog zmizí a my můžeme přejít k prezentaci vydolovaných znalostí.



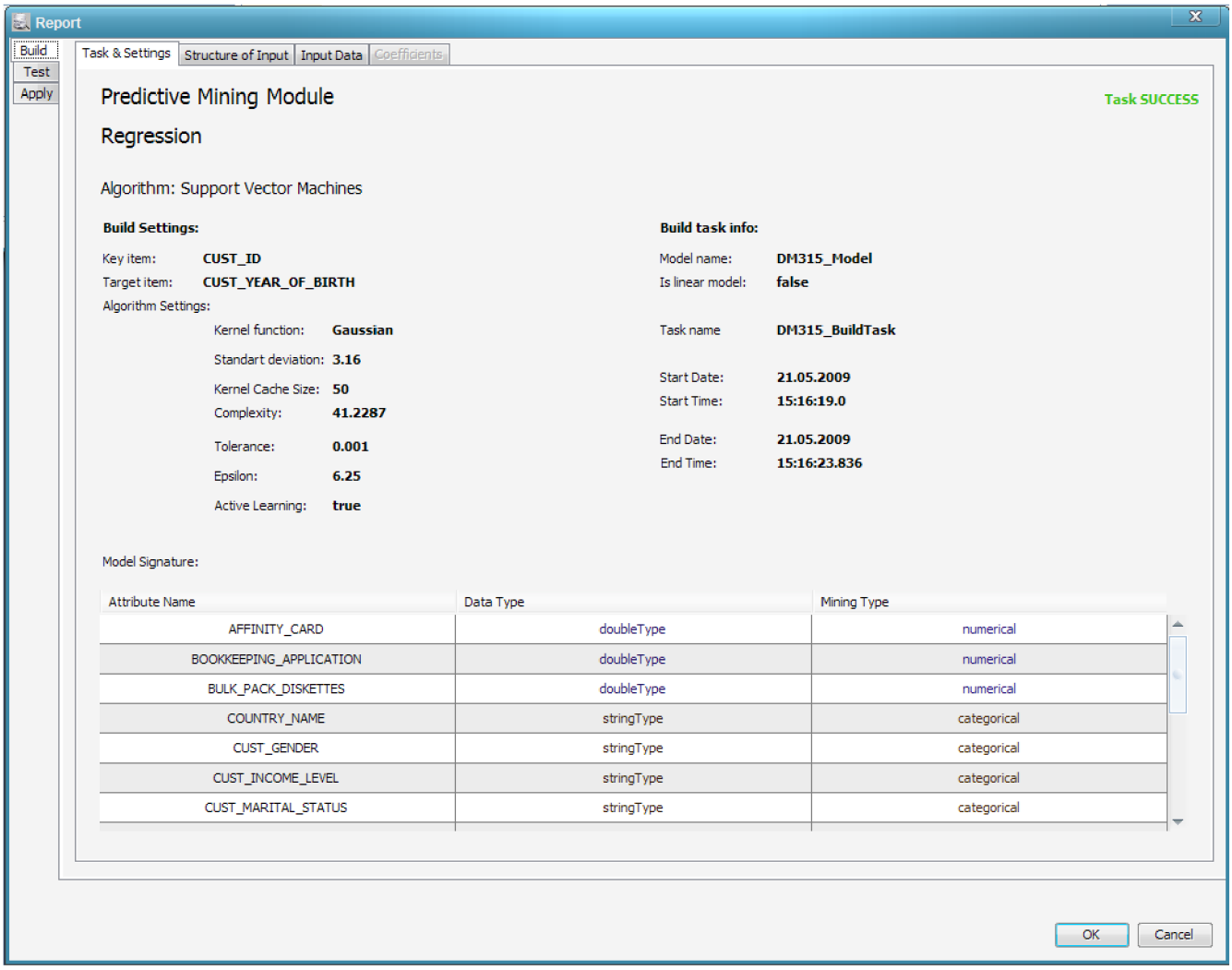
Obrázek 8.4 Komponenta Regression – panel pro zadání vstupních parametrů

8.1.4 Prezentace výsledků

K prezentaci vydolovaných znalostí je možný přístup z komponenty *Report*. Klikneme na tuto komponentu a zobrazí se nám následující panel viz obrázek 8.5.

8.1.4.1 Build fáze

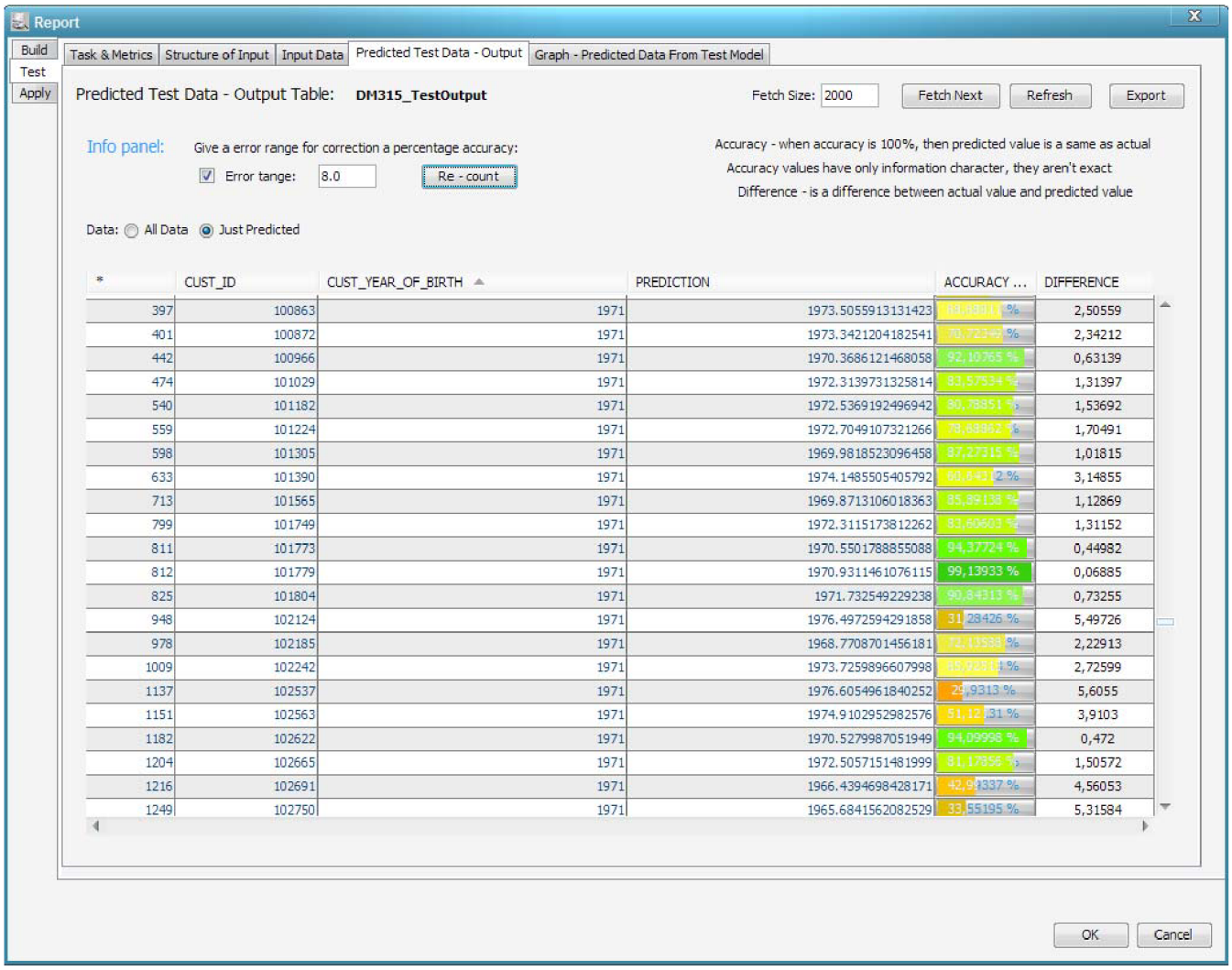
V záložce *Task & Settings* jsou zobrazeny informace o vstupních parametrech dolovací úlohy, informace o trvání trénovací úlohy a v neposlední řadě tabulka s informacemi o vytvořeném modelu. Záložka *Structure of Input* zobrazuje tabulku s metadaty vstupní tabulky, záložka *Input Data* zobrazuje data vstupní tabulky pro učení. Je možné tyto data v této tabulce zobrazovat v určitém objemu a je možné provést i export těchto dat do souboru csv nebo jako tabulku. Poslední záložka *Coefficients* není v dané situaci aktivní, jelikož je dostupná pouze při lineární regresi.



Obrázek 8.5 Komponenta Report – záložka Build – informace o trénovací úloze

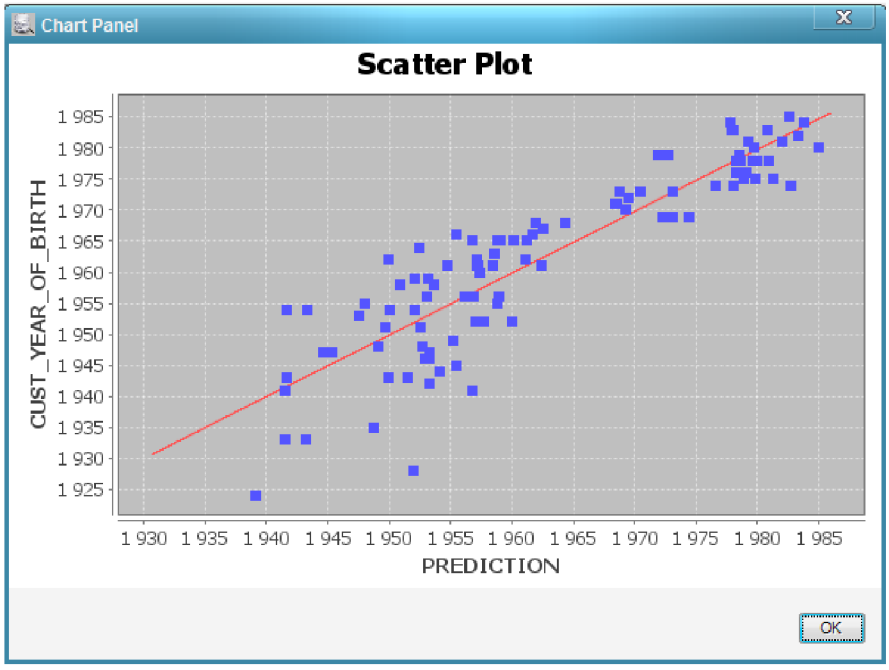
8.1.4.2 Test fáze

V záložce *Task & Metrics* jsou zobrazeny informace o trvání testovací úlohy a také získané metriky z otestovaného modelu. Záložka *Structure of Input* zobrazuje tabulku s metadaty vstupní tabulky pro testování. Záložka *Input Data* zobrazuje data vstupní tabulky pro testování. Další záložka *Predicted Test Data – Output,* viz obrázek 8.6, zobrazuje aktuální data i data predikovaná v rámci testování a jejich porovnání, je zde zobrazen sloupec popisující přesnost predikce a rozdíl obou hodnot. Pokud není zaškrtnuto pole *Error range*, je výpočet procentuelní přesnosti (*Accuracy*) přímo závislý na hodnotě aktuální a predikované. V případě, že pole je zaškrtnuto, bere se v potaz korekční hodnota, pro kterou jsme ochotni akceptovat odchylku predikce. V našem případě jsme ochotni akceptovat maximální možnou výchylku 8let. Poslední záložka *Graph – Predicted Data From Test Model* zobrazuje tabulku pouze predikovaných a aktuálních hodnot sloužící jako zdroj dat pro různé typy grafů, které zde lze vytvořit pro jednoduchou demonstraci. Možností zobrazení je hodně, je zde na výběr z více jak 10 typů grafů. Obrázek 8.7 ilustruje jeden z grafů (*Power Regression*), který popisuje závislost aktuálních a predikovaných hodnot (*Čím víc se blíží hodnoty k přímce, tím přesnější jsou – se sobě rovnají*). U posledních tří záložek je možné data uvedené v tabulkách zobrazovat v určitém objemu a je také možné provést export těchto dat do souboru csv nebo jako tabulku.



*	CUST_ID	CUST_YEAR_OF_BIRTH	PREDICTION	ACCURACY ...	DIFFERENCE
397	100863	1971	1973.5055913131423	[illegible] %	2,50559
401	100872	1971	1973.3421204182541	[illegible] %	2,34212
442	100966	1971	1970.3686121468058	92,10765 %	0,63139
474	101029	1971	1972.3139731325814	83,57534 %	1,31397
540	101182	1971	1972.5369192496942	80,78851 %	1,53692
559	101224	1971	1972.7049107321266	78,68862 %	1,70491
598	101305	1971	1969.9818523096458	87,27315 %	1,01815
633	101390	1971	1974.1485505405792	60,8432 %	3,14855
713	101565	1971	1969.8713106018363	85,99138 %	1,12869
799	101749	1971	1972.3115173812262	83,60603 %	1,31152
811	101773	1971	1970.5501788855088	94,37724 %	0,44982
812	101779	1971	1970.9311461076115	99,13933 %	0,06885
825	101804	1971	1971.732549229238	90,84313 %	0,73255
948	102124	1971	1976.4972594291858	31,28426 %	5,49726
978	102185	1971	1968.7708701456181	72,13538 %	2,22913
1009	102242	1971	1973.7259896607998	88,92814 %	2,72599
1137	102537	1971	1976.6054961840252	2[illegible],9313 %	5,6055
1151	102563	1971	1974.9102952982576	51,12[illegible]31 %	3,9103
1182	102622	1971	1970.5279987051949	94,09998 %	0,472
1204	102665	1971	1972.5057151481999	81,17896 %	1,50572
1216	102691	1971	1966.4394698428171	42,9[illegible]337 %	4,56053
1249	102750	1971	1965.6841562082529	33,55195 %	5,31584

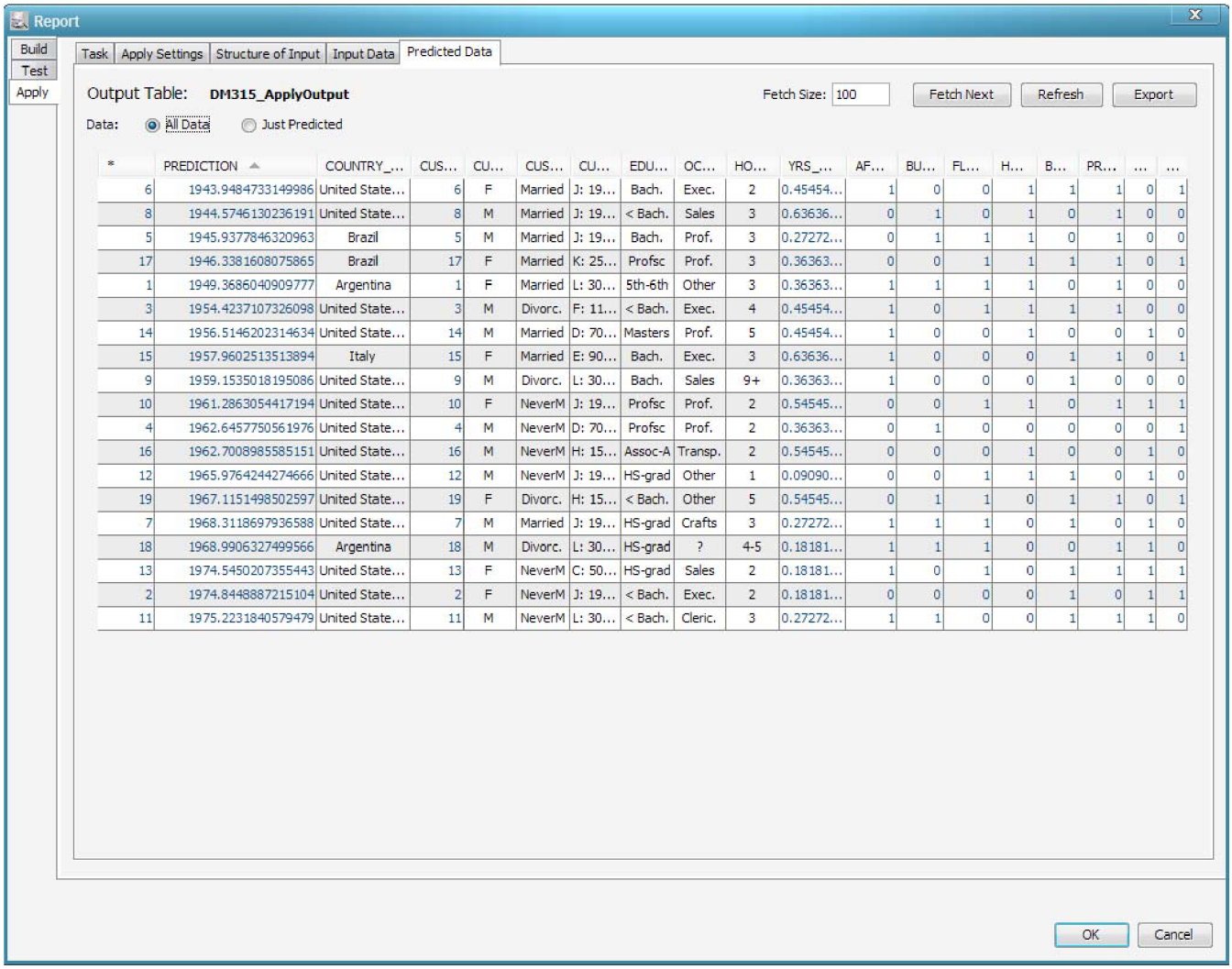
Obrázek 8.6 Komponenta Report – záložka Test – srovnání aktuálních a predikovaných dat



Obrázek 8.7 Komponenta Report – záložka Test – vykreslený graf Power Regression

8.1.5 Aplikační fáze

Aplikační fáze je volitelnou fází. Pro nás je zajímavou pouze tehdy, pokud chceme na základě vytvořeného modelu a vstupních dat s neznámou hodnotou cílového atributu predikovat hodnoty tohoto cílového atributu. Je nutné v záložce *Apply Settings* zvolit vstupní data, ty lze získat dvojím způsobem: z tabulky nebo manuálně. Pokud zvolíme možnost z tabulky, zobrazí se nám stromová struktura tabulek na serveru, vybereme tabulku, která je vhodná pro aplikaci dat (podmínkou je, aby vybraná tabulka obsahovala sloupec s primárními klíči ve stejném názvu a formátu jako sloupec s primárním klíčem použitým při vytvoření modelu), pokud zvolíme možnost vytvoření tabulky a manuální vložení dat, vytvoří se nám tabulka o stejné struktuře, jež byla použita při vytvoření modelu. Vyplníme data, pro která chceme znát výslednou předpověď a spustíme aplikační fázi tlačítkem *Run Apply Task*. Pokud proběhne vše v pořádku, zobrazí se v poslední záložce výsledná data viz obrázek 8.7. V záložce *Task* se zobrazí informace o provedené úloze. Záložka *Structure of Input* popisuje metadata vstupních dat, záložka *Input Data* zobrazuje veškerá data ze vstupní tabulky. U posledních dvou záložek je možné data uvedené v tabulkách zobrazovat v určitém objemu a je také možné provést export těchto dat do souboru csv nebo jako tabulku (neplatí jen v případě, že vstupní tabulka je vytvořena manuálně).



*	PREDICTION	COUNTRY_...	CUS...	CU...	CUS...	CU...	EDU...	OC...	HO...	YRS_...	AF...	BU...	FL...	H...	B...	PR...	...	...
6	1943.9484733149986	United State...	6	F	Married	J: 19...	Bach.	Exec.	2	0.45454...	1	0	0	1	1	1	0	1
8	1944.5746130236191	United State...	8	M	Married	J: 19...	< Bach.	Sales	3	0.63636...	0	1	0	1	0	1	0	0
5	1945.9377846320963	Brazil	5	M	Married	J: 19...	Bach.	Prof.	3	0.27272...	0	1	1	1	0	1	0	0
17	1946.3381608075865	Brazil	17	F	Married	K: 25...	Profsc	Prof.	3	0.36363...	0	0	1	1	1	1	0	1
1	1949.3686040909777	Argentina	1	F	Married	L: 30...	5th-6th	Other	3	0.36363...	1	1	1	1	0	1	0	0
3	1954.4237107326098	United State...	3	M	Divorc.	F: 11...	< Bach.	Exec.	4	0.45454...	1	0	1	1	1	1	0	0
14	1956.5146202314634	United State...	14	M	Married	D: 70...	Masters	Prof.	5	0.45454...	1	0	0	1	0	0	1	0
15	1957.9602513513894	Italy	15	F	Married	E: 90...	Bach.	Exec.	3	0.63636...	1	0	0	0	1	1	0	1
9	1959.1535018195086	United State...	9	M	Divorc.	L: 30...	Bach.	Sales	9+	0.36363...	1	0	0	0	1	0	0	0
10	1961.2863054417194	United State...	10	F	NeverM	J: 19...	Profsc	Prof.	2	0.54545...	0	0	1	1	0	1	1	1
4	1962.6457750561976	United State...	4	M	NeverM	D: 70...	Profsc	Prof.	2	0.36363...	0	1	0	0	0	0	0	1
16	1962.7008985585151	United State...	16	M	NeverM	H: 15...	Assoc-A	Transp.	2	0.54545...	0	0	0	1	0	0	1	0
12	1965.9764244274666	United State...	12	M	NeverM	J: 19...	HS-grad	Other	1	0.09090...	0	0	1	1	1	0	1	0
19	1967.1151498502597	United State...	19	F	Divorc.	H: 15...	< Bach.	Other	5	0.54545...	0	1	1	0	1	1	0	1
7	1968.3118697936588	United State...	7	M	Married	J: 19...	HS-grad	Crafts	3	0.27272...	1	1	1	0	1	0	1	0
18	1968.9906327499566	Argentina	18	M	Divorc.	L: 30...	HS-grad	?	4-5	0.18181...	1	1	1	0	0	1	1	0
13	1974.5450207355443	United State...	13	F	NeverM	C: 50...	HS-grad	Sales	2	0.18181...	1	0	1	0	1	1	1	1
2	1974.8448887215104	United State...	2	F	NeverM	J: 19...	< Bach.	Exec.	2	0.18181...	0	0	0	0	1	0	1	1
11	1975.2231840579479	United State...	11	M	NeverM	L: 30...	< Bach.	Cleric.	3	0.27272...	1	1	0	0	1	1	1	0

Obrázek 8.8 Komponenta Report – záložka Apply – tabulka zobrazující výsledné predikované hodnoty i s atributy (prediktory), ze kterých se vycházelo

8.2 Lineární regrese

8.2.1 Vytvoření nového projektu

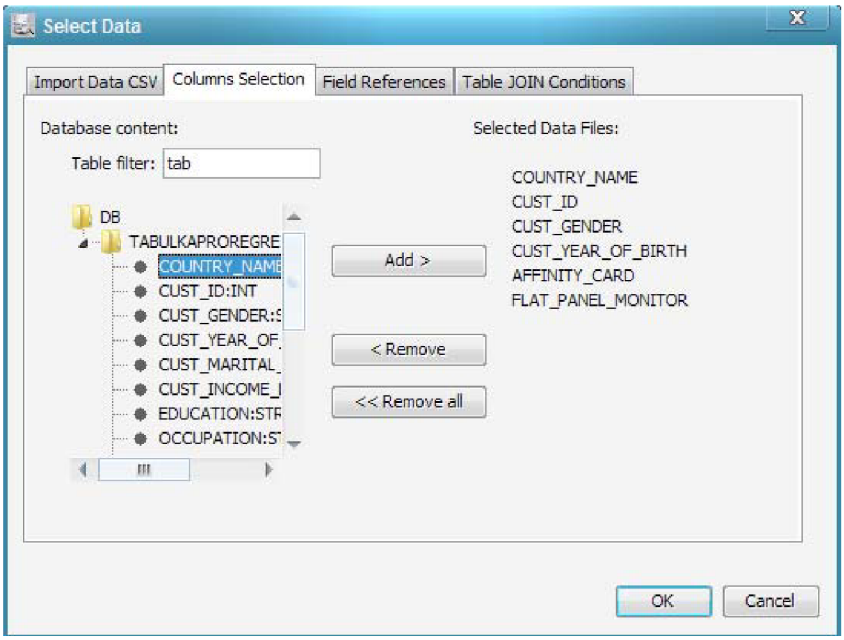
Vytvoření nového projektu a postup při přihlášení je stejný jako u nelineární regrese.

8.2.2 Nadefinování komponent v grafu

Stejně tak nadefinování komponent v grafu je zcela dostačující v takové míře, jak bylo použito v kapitole 8.1.2.

8.2.3 Vstupní data a parametry

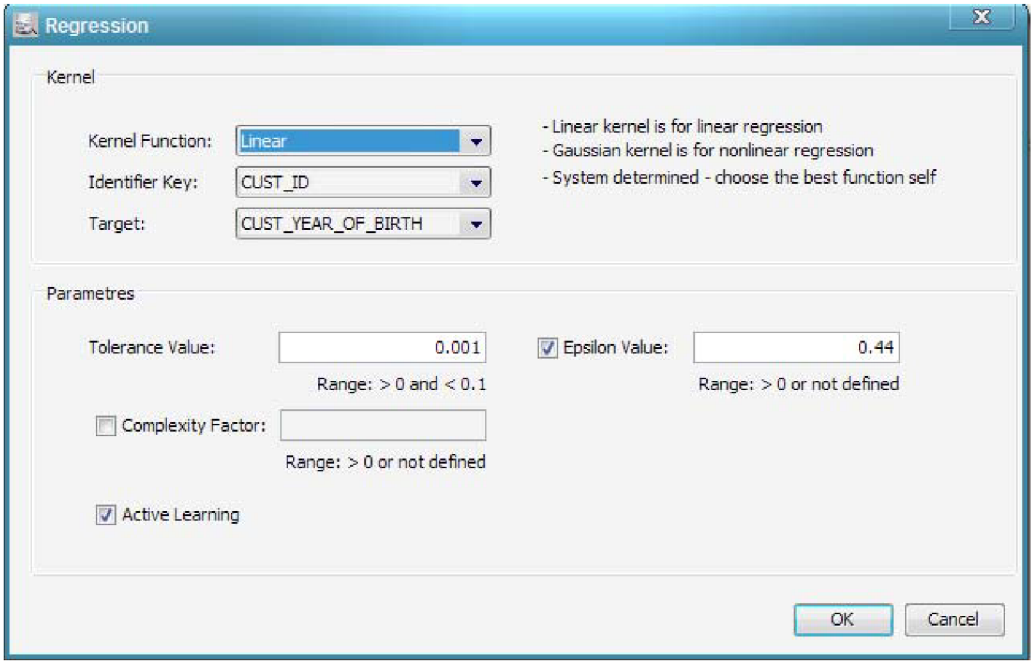
V komponentě *SelectData* zvolíme záložku *Columns Selection* a v levé části okna najdeme tabulku TABULKAPROREGRESI. Přesuneme položku CUST_ID, ta je nutná pro dolovací úlohu jako jednoznačný identifikátor, položku CUST_YEAR_OF_BIRTH, hodnoty pro tento atribut budeme chtít predikovat a například ještě pár dalších položek, sloužících jako prediktory, viz obrázek 8.9. Klikneme na tlačítko Ok. Vrátíme se na graf s komponentami, klikneme pravým tlačítkem na komponentu *SelectData* a zvolíme položku Run, proběhne výběr daných položek z tabulky.



Obrázek 8.9 Komponenta Select Data – výběr vstupních dat

Nyní je nutné rozdělit získaná data ze zvolené tabulky do dvou tabulek. Otevřeme komponentu *Reduce* a rozdělíme data podobně jako tomu bylo v kapitole 8.1.3. Pak přejdeme k predikčnímu modulu. Otevřeme komponentu *Regression*, otevře se panel pro zadání vstupních parametrů, obrázek 8.10. Zvolíme *Kernel funkci Linear*, jelikož chceme provést lineární regresi (pokud uživatel nemá velké zkušenosti s vytvářením dolovacích úloh je více než vhodné zvolit položku *Systém determined*, která způsobí, že systém zvolí ideální nastavení automaticky v závislosti na vstupních datech). Jako položku *identifier key* (jednoznačný identifikátor) zvolíme CUST_ID. Hodnotu položky *Target* zvolíme CUST_YEAR_OF_BIRTH. Ostatní parametry a jejich hodnoty zadejte například tak, jak je popsáno na obrázku 8.10, (význam parametrů je popsán v kapitole 7.2.2), klikneme na tlačítko Ok a pokud je vše zadáno správně můžeme začít dolovat. V grafu klikneme na daný modul

pravým tlačítkem a zvolíme Run, tím se spustí dolovací úloha. Na začátku samotného dolování se u lineární regrese provede normalizace vstupních dat (nutný krok pro lineární kernel). Dále pokračuje stejnými fázemi jako při nelineární regresi – vytvoření modelu v rámci učení a otestování modelu.



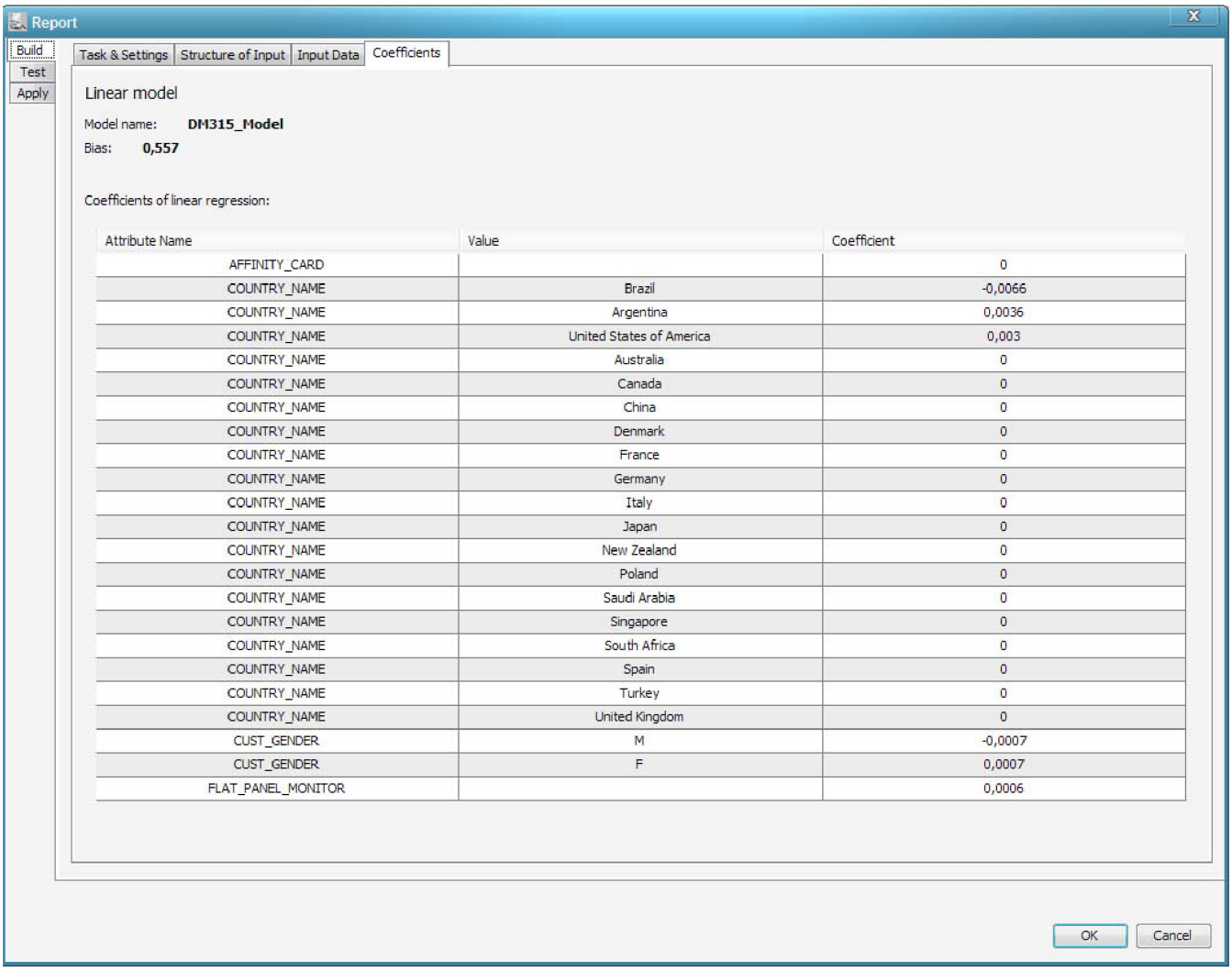
Obrázek 8.10 Komponenta Regression – panel pro zadání vstupních parametrů

8.2.4 Prezentace výsledků

Dolovací úloha proběhla úspěšně a nyní je možné přistoupit k výsledkům vydolovaných znalostí. Otevřeme komponentu *Report*.

8.2.4.1 Build fáze

V záložce *Task & Settings* jsou zobrazeny informace o vstupních parametrech dolovací úlohy, informace o trvání trénovací úlohy a v neposlední řadě tabulka s informacemi o vytvořeném modelu. Záložka *Structure of Input* zobrazuje tabulku s metadaty vstupní tabulky pro učení, záložka *Input Data* zobrazuje data vstupní tabulky a poslední záložka *Coefficients* zobrazuje atributy modelu, jejich hodnoty a koeficienty hodnot, viz obrázek 8.11. (Coefficienty jsou dostupné pouze v rámci lineární regrese).

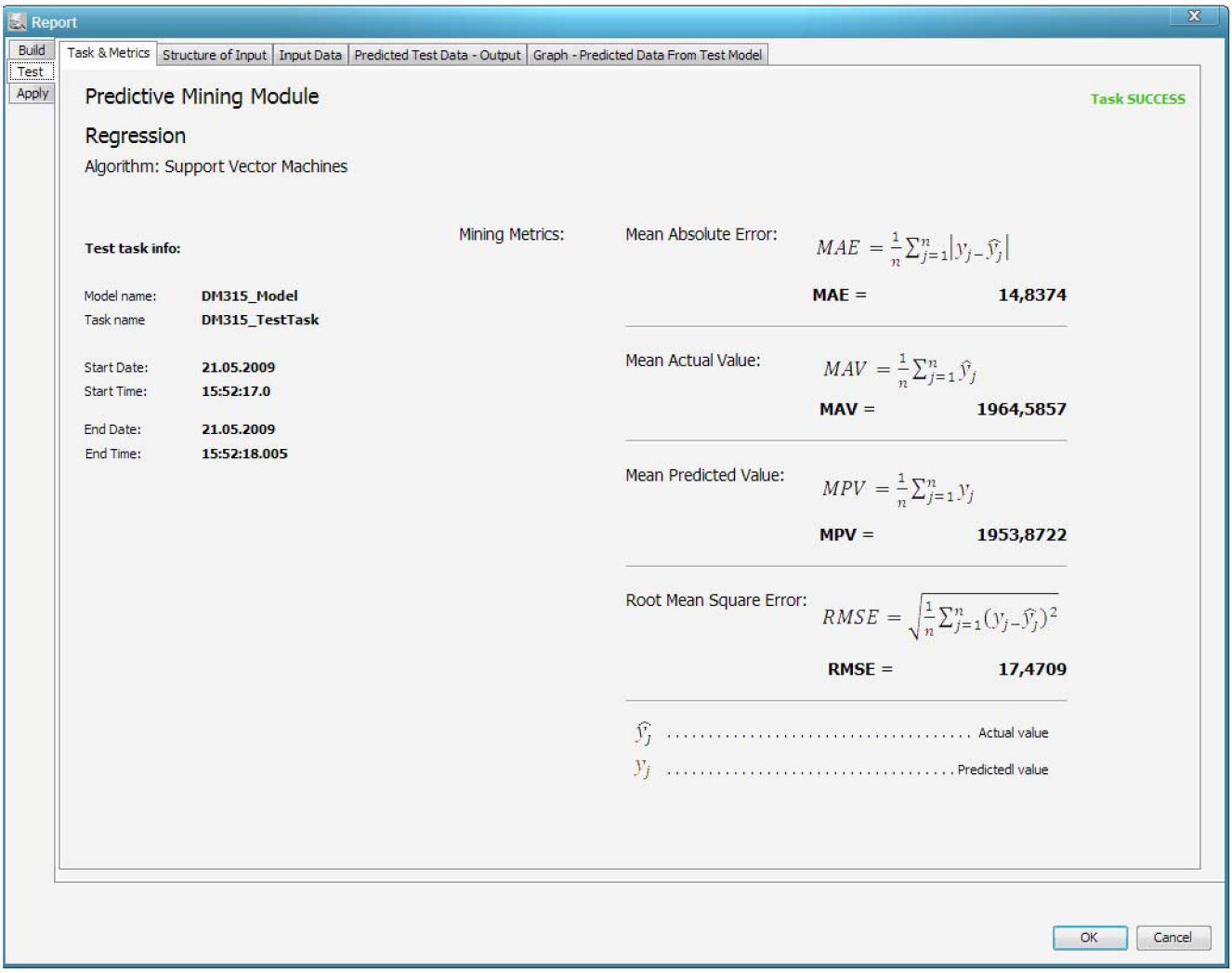


Attribute Name	Value	Coefficient
AFFINITY_CARD		0
COUNTRY_NAME	Brazil	-0,0066
COUNTRY_NAME	Argentina	0,0036
COUNTRY_NAME	United States of America	0,003
COUNTRY_NAME	Australia	0
COUNTRY_NAME	Canada	0
COUNTRY_NAME	China	0
COUNTRY_NAME	Denmark	0
COUNTRY_NAME	France	0
COUNTRY_NAME	Germany	0
COUNTRY_NAME	Italy	0
COUNTRY_NAME	Japan	0
COUNTRY_NAME	New Zealand	0
COUNTRY_NAME	Poland	0
COUNTRY_NAME	Saudi Arabia	0
COUNTRY_NAME	Singapore	0
COUNTRY_NAME	South Africa	0
COUNTRY_NAME	Spain	0
COUNTRY_NAME	Turkey	0
COUNTRY_NAME	United Kingdom	0
CUST_GENDER	M	-0,0007
CUST_GENDER	F	0,0007
FLAT_PANEL_MONITOR		0,0006

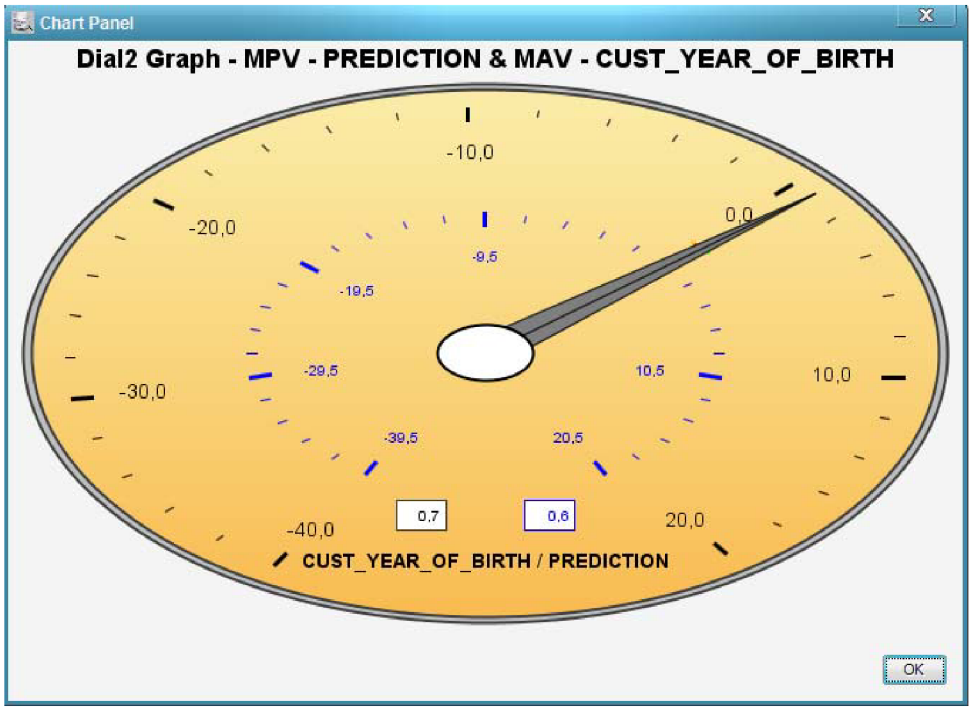
Obrázek 8.11 Komponenta Report – záložka Build – informace o koeficientech prediktorů

8.2.4.2 Test fáze

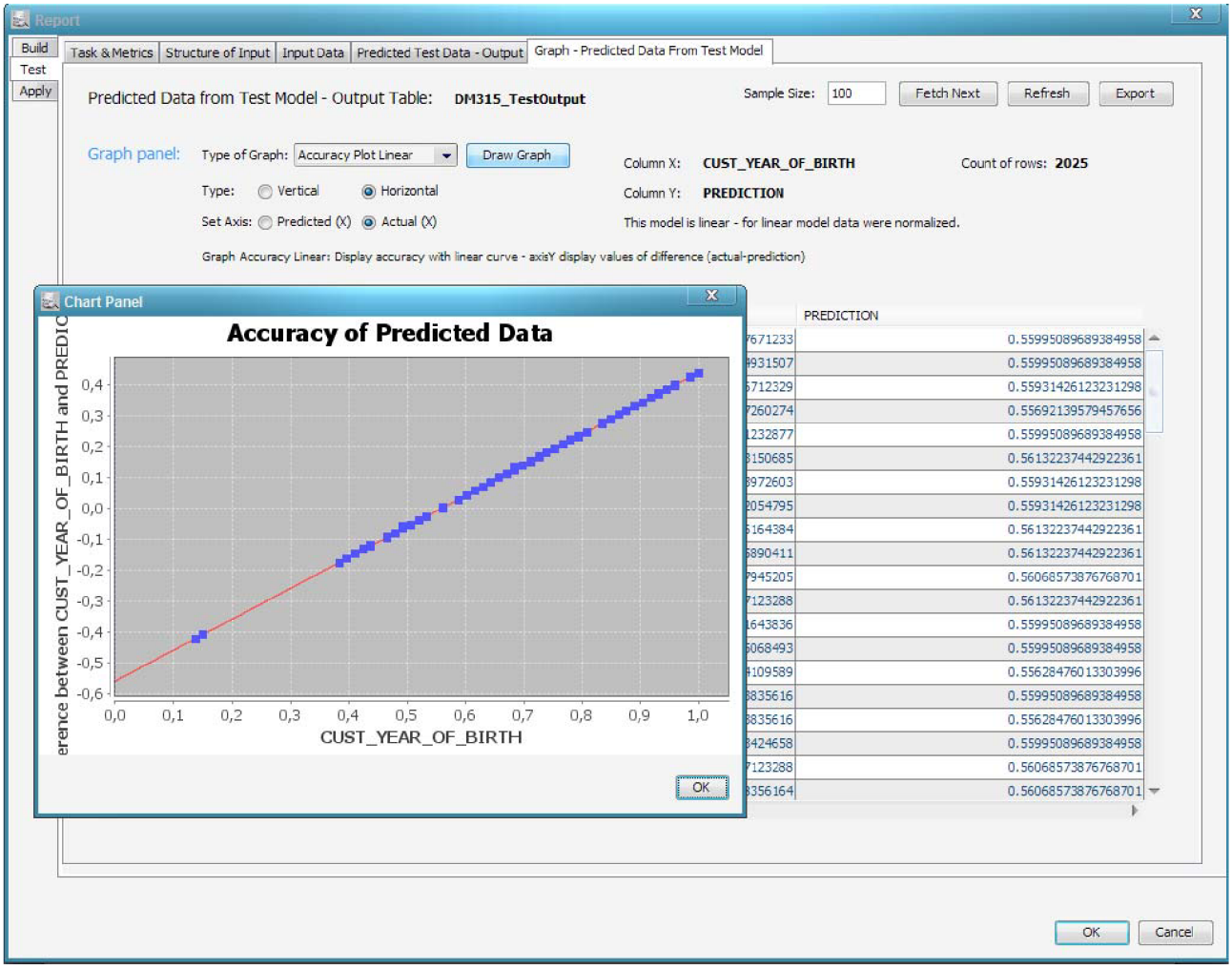
V záložce *Task & Metrics* jsou zobrazeny informace o trvání testovací úlohy a také získané metriky z otestovaného modelu viz obrázek 8.12. Záložka *Structure of Input* zobrazuje tabulku s metadat, záložka *Input Data* zobrazuje data vstupní tabulky. Záložka *Predicted Test Data – Output* zobrazuje aktuální data i data predikovaná v rámci testování a jejich porovnání, přesnost predikce a rozdíl obou hodnot. Poslední záložka *Graph – Predicted Data From Test Model* zobrazuje tabulku pouze predikovaných a aktuálních hodnot sloužící jako zdroj dat pro různé typy grafů, které zde lze vytvořit pro jednoduchou demonstraci viz obrázek 8.13 popisující metriky MPV a MAV a obrázek 8.14 ilustruje poslední záložku i s grafem popisujícím výslednou lineární regresní funkci. V posledních dvou záložkách jsou zobrazena data v normalizované podobě, protože v rámci lineární regrese jsou tak zpracovány.



Obrázek 8.12 Komponenta Report – záložka Test – informace o získaných metrikách a průběhu testovací úlohy



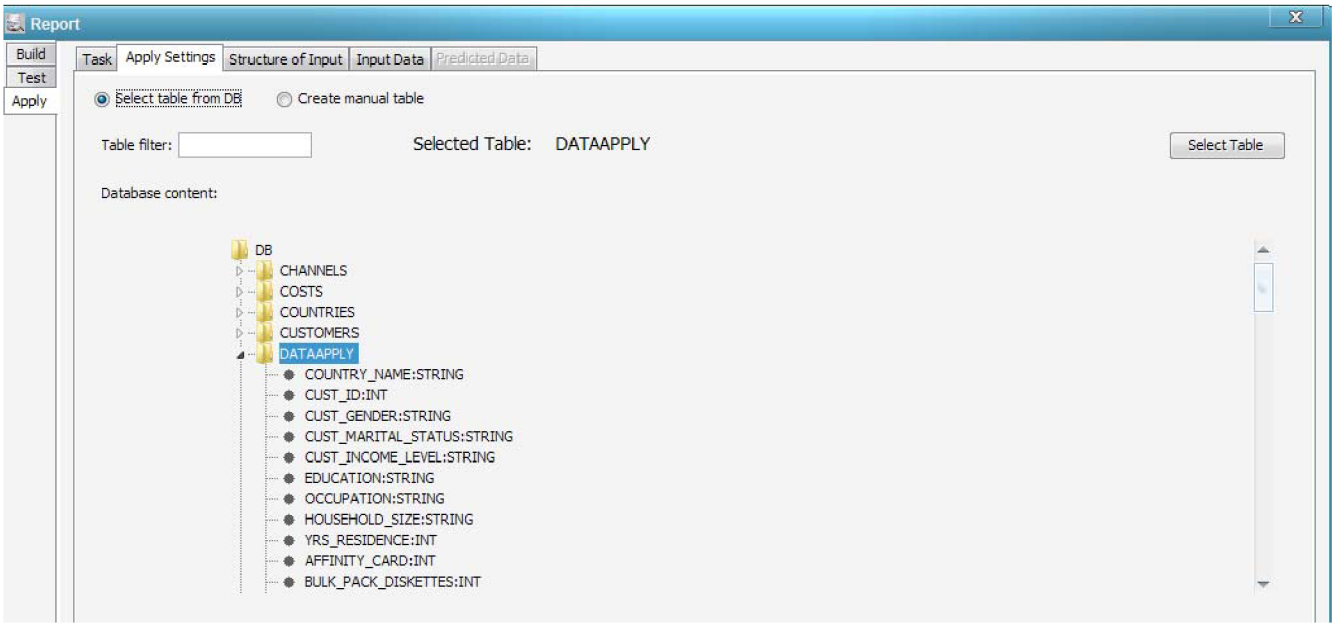
Obrázek 8.13 Komponenta Report – záložka Test – vykreslený graf Dial2 popisující metriky MAV a MPV



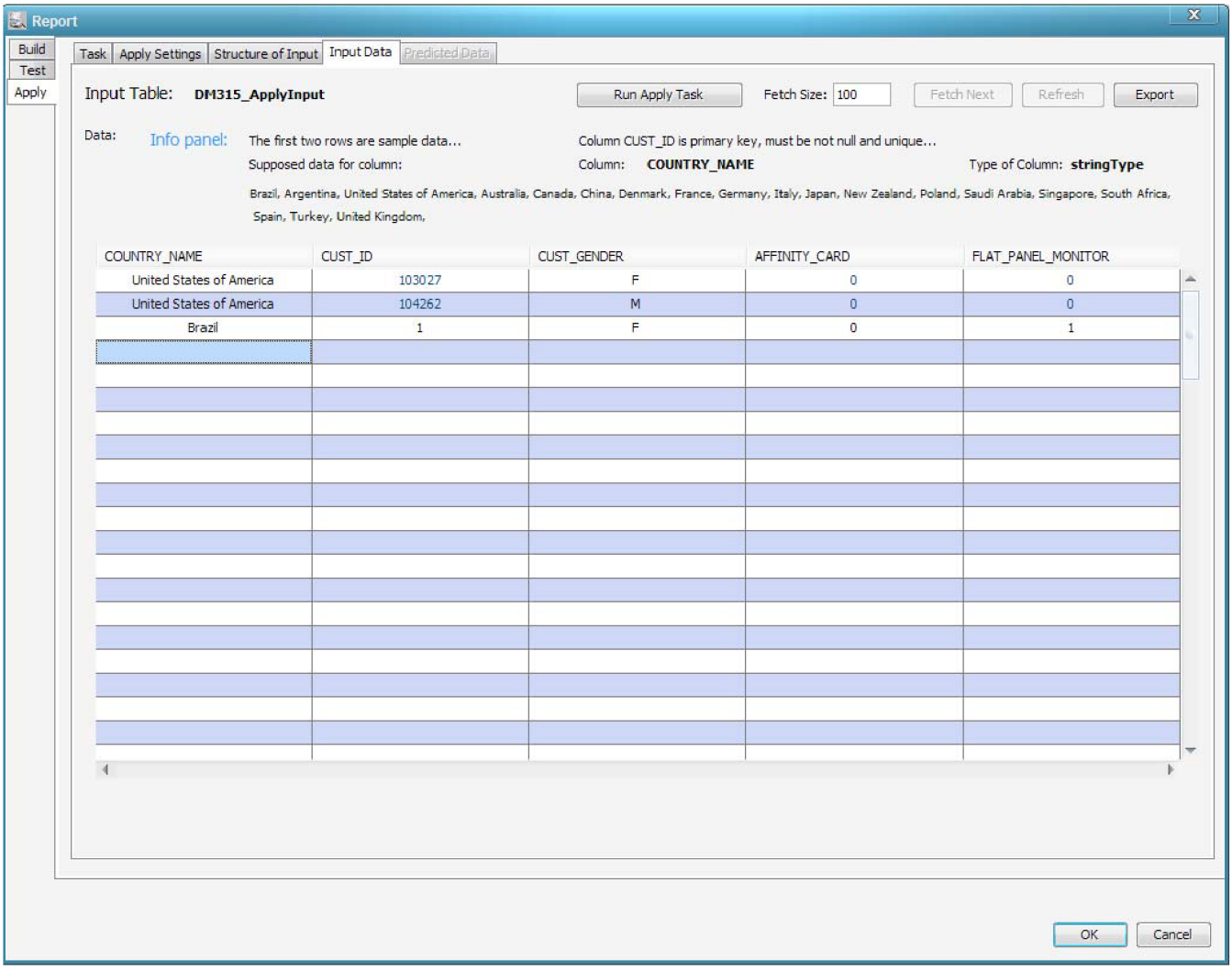
Obrázek 8.14 Komponenta Report – záložka Test – záložka Graph s vykresleným grafem Accuracy Plot Linear

8.2.5 Aplikační fáze

Aplikační fáze je, jak už bylo mnohokrát řečeno, volitelnou fází. Není tedy nutné tuto fázi provádět, pokud nechceme získat nové hodnoty. V záložce *Apply Settings* obrázek 8.15 zvolíme možný vstup a to z uložené tabulky na serveru nebo si můžeme vytvořit prázdnou a data vložit ručně. Zvolíme vytvořit prázdnou. Zobrazí se nám záložka *Input Data* s tabulkou, do které můžeme vkládat data. První dva řádky jsou pro ukázku vyplněny. Snahou bylo pomoci uživateli objasnit, jaká data vkládat viz obrázek 8.16. Poté spustíme proces tlačítkem *Run Apply Task*. Pokud vše proběhlo v pořádku zobrazí se záložka *Predicted Data* s predikovanými daty. Záložka *Task* popisuje informace o provedené úloze, záložka *Structure of Input* popisuje metadata vstupních dat, záložka *Input Data* zobrazuje veškerá data ze vstupní tabulky.



Obrázek 8.15 Komponenta Report – záložka Apply – Apply Settings – výběr aplikační tabulky



Obrázek 8.16 Komponenta Report – záložka Apply – manuální vložení vstupních dat

9 Závěr

9.1 Dosažené výsledky

Cílem této diplomové práce bylo vytvořit predikční modul v systému pro dolování z dat na *platformě NetBeans*. Nejprve bylo nutné nastudovat obecný princip dolování, jednotlivé jeho kroky a zaměřit se na predikci. Popisem těchto problematik se v dostatečné míře zabývám v první části této práce. Dále bylo nutné se seznámit se systémem pro získávání znalostí z dat, který je vyvíjen na *VUT FIT* a pro který je určen tento modul. K tomu mi posloužili práce Ing. Doležala [5], Ing. Gáleta [6] a hlavně Ing. Krásného [2]. Právě práce Ing. Krásného mi byla předlohou pro vytvoření a zakomponování nového modulu do stávajícího systému. K implementaci predikčního modulu byla vybrána metoda regrese a dolovací algoritmus Support Vector Machines.

Systém pracuje s relační databází firmy Oracle, která nabízí podporu dolování v rámci *Oracle Data Mining*. Tento implementovaný predikční modul zmiňovanou podporu využívá. Podstatnou částí mé práce, bylo nastudování práce s *ODM*, možnost podpory dolování pomocí algoritmu *Support Vector Machines* a využití *Java API*. První vážnější problém, se kterým jsem se během práce setkal, bylo zprovoznění vzorových příkladů, které Oracle poskytuje. Ty využívají tzv. *SH schémat*, což jsou vzorová data, ke kterým v rámci studentských účtů není přístup (vyžadují vysoký stupeň oprávnění). Dále bylo nutné navrhnout rozšíření definice jazyka DMSL o popis elementů *DataMiningTask* a *Knowledge*, které uchovávají kompletní informace o vstupních parametrech pro nastavení dolovací úlohy, zdrojích dat a také o získaných znalostech.

Během implementace bylo zjištěno několik problémů. Jedním z vážnějších byl problém s některými balíčky potřebnými k vytvoření lineární regrese. Tyto balíčky systém obsahoval, ale jádro k nim zamezovalo přístup z modulu. Snažil jsem se klást velký důraz nejen na samotný proces dolování, ale i na zajímavou, užitečnou a podrobnou prezentaci výsledků. Proto byl navrhnut výsledný panel v podobě, která popisuje nejen získané znalosti, ale i informace spojené s průběhem každé fáze dolování, informace o vstupních datech a metadatech. Získané znalosti je možné porovnávat, zobrazit názorně pomocí více jak patnácti typů grafů. Aby se dal model a získané informace z fází trénování a testování rozumně využít, byla navrhnuta a implementována i *aplikační fáze*. Ta je fází volitelnou a lze ji spustit z panelu pro prezentaci výsledků. Díky ní je možné na základě vstupních dat a získaných znalostí predikovat data pro zvolený cílový atribut.

Vytvořený dolovací modul je funkční a plně využitelný pro predikci. Rozšiřuje tak stávající dolovací systém o další možnost využití. Společně s dalšími diplomovými pracemi tak zvyšuje počet dolovacích modulů v systému. Přesto však možnosti tvorby dalších modulů nejsou vyčerpány.

9.2 Možné úpravy a vylepšení

Práce s více projekty

Při vytvoření nebo otevření dvou či více projektů nastává problém s aktuálním DMSL dokumentem. Výstupnímu panelu není přidělen nový, ale zůstává DMSL dokument z projektu předešlého. Jedinou možností je vypnutí a opětovné zapnutí programu.

Odstranění projektu

Projekt je možné zavřít nikoliv odstranit. Bylo by vhodné zavést možnost pro odstranění projektu a s tím spojenou možnost zrušení všech tabulek z databáze v projektu vytvořených.

Rozšíření dolovacích modulů

Rozšířit počet dostupných dolovacích modulů. Program je prozatím chudý co do výčtu modulů pro dolování. A to i přesto, že v letošním roce bylo v rámci diplomových prací vytvořeno hned několik nových modulů.

Možnost provést transformace a funkce VIMEO nad aplikační tabulkou

U prediktivních modulů, u kterých je možné v reportu provést aplikační úlohu nad nějakou vstupní tabulkou, by bylo vhodné zavést možnost aplikace transformace a funkcí VIMEO.

Závislost dolovacích modulů na ODM

Časem by bylo vhodné oprostit se od závislosti dolovacích modulů na ODM a samotné dolování provádět i jiným způsobem (v rámci PL/SQL)

Literatura

[1] Han, J., Kamber, M. *Data Mining: Concepts and Techniques. Second Edition*. Elsevier Inc., 2006, 770p.

[2] Krásný, M.: *Systém pro dolování z dat v prostředí Oracle,* diplomová práce, Brno, 2008, Vysoké Učení Technické, Fakulta Informačních Technologií.

[3] Zendulka J., Bartík V., Lukáš R., Rudolfová I.: *Získávání znalostí z databází,* studijní opora, Brno, 2006, Vysoké učení technické v Brně, Fakulta informačních technologií.

[4] Wikipedie: Lineární regrese – Wikipedie, Otevřená encyklopedie. 2008, [Online navštíveno 2. 2. 2009]. URL http://cs.wikipedia.org/wiki/Line%C3%A1rn%C3%AD_regrese

[5] Doležal J.: *Jádro systému pro dolování z dat v prostředí Oracle*, diplomová práce, Brno, 2006, Vysoké učení technické v Brně, Fakulta informačních technologií.

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

[7] Kotásek, P.: *DMSL: Data Mining Specification Language*, disertační práce, Brno, 2003, Vysoké učení technické v Brně, Fakulta informačních technologií.

[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] Oracle® Data Mining Application Developer's Guide 10g Release 2 (10.2), *Oracle* [online]. 2004, URL http://download.oracle.com/docs/html/B14340_01/toc.htm

[10] Oracle® Data Mining Concepts 10g Release 2 (10.2), *Oracle* [online]. 2004, URL http://download.oracle.com/docs/html/B14340_01/toc.htm

[11] Java Data Mining: JSR-73: JavaDoc, *Oracle* [online], 2007. URL http://www.oracle.com/technology/products/bi/odm/JSR-73/index.html

[12] NetBeans: *Wikipedie*, [online], 2007. URL http://cs.wikipedia.org/wiki/NetBeans

[13] XML – DTD: *An introduction to XML Document Type Definitions*, [online], 2009. URL http://xmlfiles.com/dtd

Seznam příloh

Příloha A: Úpravy a návrh specifikace DMSL

```
<!-- Data Mining Task -->
<!-- ################ -->

<!ELEMENT DataMiningTask (Regression?) >
<!ATTLIST DataMiningTask name CDATA #REQUIRED
                         language CDATA #REQUIRED
                         languageVersion CDATA #IMPLIED
                         type CDATA #IMPLIED >

<!ELEMENT Regression (InputDataType, Parametres)>
<!ATTLIST Regression algorithm CDATA #REQUIRED>

<!--  Input Data Type -->
<!-- ################ -->

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

<!--    Parametres    -->
<!-- ################ -->

<!ELEMENT Parametres (KernelFunction, Tolerance, ActiveLearning, Epsilon?,
ComplexityFactor?, StandartDeviation?, CacheSize?)>

<!ELEMENT KernelFunction (Gaussian|Linear|System determined)>

<!ELEMENT Tolerance (%REAL-NUMBER;)>

<!ELEMENT ActiveLearning (true|false)>

<!ELEMENT Epsilon (%REAL-NUMBER;)>

<!ELEMENT ComplexityFactor (%REAL-NUMBER;)>

<!ELEMENT StandartDeviation (%REAL-NUMBER;)>

<!ELEMENT CacheSize (%INT-NUMBER;)>
```

```
<!--     Knowledge     -->
<!-- ################  -->

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

<!--     BuildTask     -->
<!-- ################  -->

<!ELEMENT BuildTask (Model)>
<!ATTLIST BuildTask name CDATA #REQUIRED
                    model CDATA #REQUIRED
                    input CDATA #REQUIRED
                    start %DATE-TIME; #REQUIRED
                    end %DATE-TIME; #REQUIRED
                    status CDATA; #REQUIRED >

<!ELEMENT Model (ModelSignatureAttribute)>
<!ATTLIST Model name CDATA #REQUIRED
                bias %REAL-NUMBER #IMPLIED
                isLinear (true|false) #REQUIRED >

<!ELEMENT ModelSignatureAttribute (AttributeProperties*) >
<!ATTLIST ModelSignatureAttribute name CDATA #REQUIRED
                                  dataType CDATA #REQUIRED
                                  miningType CDATA #REQUIRED >

<!ELEMENT AttributeProperties EMPTY>
<!ATTLIST AttributeProperties value CDATA #REQUIRED
                              coefficient %REAL-NUMBER; #REQUIRED >

<!--     TestTask     -->
<!-- ################  -->

<!ELEMENT TestTask Metrics>
<!ATTLIST TestTask name CDATA #REQUIRED
                   model CDATA #REQUIRED
                   input CDATA #REQUIRED
                   output CDATA #REQUIRED
```

```
                    start %DATE-TIME; #REQUIRED
                    end %DATE-TIME; #REQUIRED
                    status CDATA; #REQUIRED >

<!ELEMENT Metrics EMPTY>
<!ATTLIST Metrics MAE %REAL-NUMBER; #REQUIRED
                  MAV %REAL-NUMBER; #REQUIRED
                  MPV %REAL-NUMBER; #REQUIRED
                   RMSE %REAL-NUMBER; #REQUIRED >

<!--     ApplyTask     -->
<!-- ################  -->

<!ELEMENT ApplyTask EMPTY>
<!ATTLIST ApplyTask name CDATA #REQUIRED
                    model CDATA #REQUIRED
                    input CDATA #REQUIRED
                    output CDATA #REQUIRED
                    start %DATE-TIME; #REQUIRED
                    end %DATE-TIME; #REQUIRED
                    status CDATA; #REQUIRED >
```

Příloha B: Obsah přiloženého CD

1. dist – spustitelna distribuce aplikace DataMiner
2. doc – textová část diplomové práce
3. sample – ukázkové příklady
4. workspace – zdrojové kódy aplikace i s modulem asociačních pravidel
5. dmsl.xml – ukázka DMSL dokumentu
6. readme.txt - popis obsahu CD a ukázkových příkladů