VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

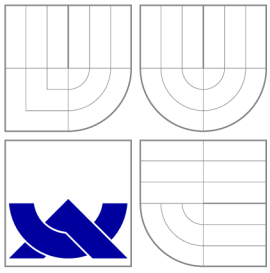
ROZŠÍŘENÍ FUNKCIONALITY SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

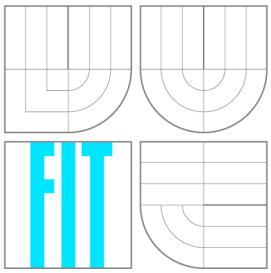
AUTOR PRÁCE
AUTHOR

Bc. MICHAL ŠEBEK

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

ROZŠÍŘENÍ FUNKCIONALITY SYSTÉMU PRO DOLOVÁNÍ Z DAT NA PLATFORMĚ NETBEANS

FUNCTIONALITY EXTENSION OF DATA MINING SYSTEM ON NETBEANS PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL ŠEBEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ROMAN LUKÁŠ, Ph.D.

BRNO 2009

Zadání diplomové práce/9059/2008/xsebek00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2008/2009

Zadání diplomové práce

Řešitel: **Šebek Michal, Bc.**
Obor: Informační systémy
Téma: **Rozšíření funkcionality systému pro dolování z dat na platformě NetBeans**
Kategorie: Databáze

Pokyny:

1. Seznamte se s problematikou získávání znalostí z databází. Seznamte se s implementací jádra systému řešeného v diplomové práci Ing. M. Krásného.
2. Po dohodě s vedoucím diplomové práce navrhněte rozšíření funkcionality jádra systému v oblasti předzpracování dat.
3. Navrženou část implementujte, integrujte se zbytkem systému a ověřte funkčnost.
4. Zhodnoťte dosažené výsledky a diskutujte další možný vývoj projektu.

Literatura:

- Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Second Edition. Elsevier Inc., 2006, 770 p.
- Krásný, M.: Systém pro dolování z dat v prostředí Oracle. Diplomová práce. FIT VUT v Brně. 2007.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Lukáš Roman, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 22. září 2008
Datum odevzdání: 26. května 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Databáze se neustále rozrůstají o nová data. Za účelem analýzy těchto dat byl definován proces získávání znalostí z databází. Pro podporu tohoto procesu vznikla řada nástrojů. Vývojem jednoho z těchto nástrojů se zabývá tato práce.
Hlavním cílem je analyzovat stávající implementaci systému na přenositelné platformě Java NetBeans a databázovém serveru Oracle a rozšířit ji o algoritmy z oblasti předzpracování a analýzy vstupních dat. Podrobně je popsána implementace jednotlivých komponent pro předzpracování dat a provedené změny v jádře systému.

Abstract

Databases increase by new data continually. A process called Knowledge Discovery in Databases has been defined for analyzing these data and new complex systems has been developed for its support. Developing of one of this systems is described in this thesis.
Main goal is to analyse the actual state of implementation of this system which is based on the Java NetBeans Platform and the Oracle database system and to extend it by data preprocessing algorithms and the source data analysis. Implementation of data preprocessing components and changes in kernel of this system are described in detail in this thesis.

Klíčová slova

dolování z dat, získávání znalostí z databází, platforma NetBeans, předzpracování dat, statistická a korelační analýza, Oracle Data Mining, DMSL, systém pro dolování

Keywords

data mining, knowledge discovery in databases, NetBeans Platform, data preprocessing, statistics and correlation analysis, Oracle Data Mining, DMSL, datamining system

Citace

Michal Šebek: Rozšíření funkcionality systému pro dolování z dat na platformě NetBeans, diplomová práce, Brno, FIT VUT v Brně, 2009

Rozšíření funkcionality systému pro dolování z dat na platformě NetBeans

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Romana Lukáše, Ph.D.

.....
Michal Šebek
12. května 2009

Poděkování

Na tomto místě bych rád poděkoval Ing. Romanu Lukáši Ph.D. za poskytnutí cenných rad a ochoty při řešení problémů při tvorbě této práce a Doc. Ing. Jaroslavu Zendulkovi, CSc. za poskytnuté informace k vyvíjenému systému.

Obsah

1 Úvod **4**

2 Získávání znalostí z databází **6**
2.1 Dolování z dat jako proces . . . 6
2.2 Předzpracování dat . . . 7
2.3 Dolování dat a typy dolovacích úloh . . . 7
2.3.1 Popis konceptu/třídy . . . 7
2.3.2 Dolování frekventovaných vzorů a asociačních pravidel . . . 7
2.3.3 Klasifikace a predikce . . . 8
2.3.4 Shluková analýza . . . 8
2.3.5 Analýza odlehlých objektů . . . 8
2.3.6 Evoluční analýza . . . 8
2.4 Vyhodnocení a prezentace výsledků . . . 8

3 Předzpracování dat **9**
3.1 Charakteristika a statistiky nad vstupním souborem dat . . . 9
3.1.1 Míry středu – průměr, medián a modus . . . 9
3.1.2 Míry variace dat . . . 10
3.1.3 Kvantily . . . 10
3.1.4 Vizualizační techniky pro charakteristiky dat . . . 10
3.2 Čištění dat . . . 13
3.2.1 Chybějící hodnota . . . 13
3.2.2 Zašuměná data . . . 14
3.3 Transformace dat . . . 15
3.3.1 Vyhlazování . . . 15
3.3.2 Agregace . . . 15
3.3.3 Generalizace . . . 15
3.3.4 Normalizace . . . 16
3.3.5 Konstrukce atributů . . . 17
3.4 Redukce dat . . . 17
3.4.1 Přístupy k redukci dat . . . 17
3.4.2 Vzorkování . . . 17
3.4.3 Diskretizace kvantitativních atributů . . . 18
3.4.4 Analýza korelace atributů . . . 18

4 Systém pro dolování z dat na platformě NetBeans **20**
4.1 Použité technologie . . . 20
4.1.1 Jazyk DMSL . . . 20
4.1.2 Podpora pro dolování dat systému Oracle . . . 20
4.1.3 Platforma NetBeans a NetBeans Visual Library . . . 21
4.2 Vývoj systému . . . 21
4.3 Poznatky ze stávající implementace . . . 22
4.3.1 Jádro systému . . . 22
4.3.2 Komponenty GUI . . . 23

5 Návrh rozšíření existujícího systému v oblasti předzpracování dat **24**
5.1 Čištění dat, VIMEO . . . 24
5.2 Transformace dat . . . 24
5.3 Podpora analýzy dat – komponenta „Insight“ . . . 25
5.4 Komponenta redukce dat . . . 26
5.5 Řešení dalších nedostatků . . . 26

6 Modifikace stávajících komponent předzpracování dat **27**
6.1 Komponenta výběru dat . . . 27
6.1.1 Proces výběru dat . . . 27
6.1.2 Import CSV dat . . . 28
6.1.3 Výběr sloupců tabulek a nastavení referencí . . . 28
6.2 Komponenta VIMEO funkcí . . . 29
6.2.1 Přepracování zadávání funkcí v systému a jejich vyhodnocení . . . 29
6.2.2 Úpravy VIMEO komponenty . . . 32
6.2.3 Vyhodnocování VIMEO funkcí . . . 32
6.3 Komponenta transformací . . . 33
6.3.1 Reference sloupců do číselníků . . . 33
6.3.2 Diskretizace hodnot a koncept realizace transformací . . . 34
6.3.3 Vyhlazení hodnot sloupce . . . 36
6.3.4 Konstrukce (odvození) nového atributu . . . 36

7 Implementace nových komponent předzpracování dat **37**
7.1 Komponenta pro analýzu dat . . . 37
7.1.1 Základní funkčnost komponenty . . . 37
7.1.2 Výpočet statistik . . . 38
7.1.3 Korelační analýza . . . 39
7.1.4 Vizualizace pomocí grafů . . . 40
7.1.5 Export dat z dolovacího procesu . . . 41
7.1.6 Povolení zobrazení *Insight* z kontextového menu . . . 41
7.2 Komponenta redukce a rozdělení dat . . . 42
7.2.1 Návrh komponenty . . . 42
7.2.2 Redukce dat . . . 42
7.2.3 Optimalizační problémy redukce dat . . . 43
7.2.4 Rozdělení dat . . . 44
7.2.5 Modifikace třídy `MiningPiece` . . . 45
7.2.6 Úpravy v DMSL . . . 46

8 Návrh a implementace změn v jádře systému **48**
8.1 Abstrakce přístupu k databázi z jádra 48
8.2 Řešení vícevláknového běhu výpočtu a obsluhy GUI 49
8.3 Odlišení databázových tabulek pro více projektů 51
8.4 Úprava procesu připojování k databázi 52
8.5 Konvence maximální délky názvu komponenty jádra 53

9 Návrh dalších rozšíření projektu **54**

10 Závěr **55**

Literatura **56**

Seznam použitých zkratek a symbolů **58**

Seznam příloh **60**

A Uživatelský manuál k předzpracování dat **61**
A.1 Instalace systému 61
A.2 Práce s nástrojem a založení projektu 61
A.3 Komponenta výběru dat *Select Data* 62
A.4 Komponenta analýzy dat *Insight* 64
A.5 Definice uživatelských funkcí 65
A.6 Komponenta čištění dat *VIMEO* 66
A.7 Komponenta transformací *Transformations* 67
A.8 Komponenta redukce a rozdělení dat *Reduce/Partition* 68
A.9 Dolování z dat 68

B Úpravy definice dokumentu DMSL **69**

C Obsah přiloženého CD **70**

Kapitola 1

Úvod

V dnešní době se potýkáme s obrovským nárůstem množství uchovávaných dat. Nabízí se tedy otázka, jestli nelze ze zaznamenaných dat získat i další poznatky, které by se mohly ve velkém objemu dat objevit. Typickým příkladem je využití pro účely marketingu, kde by bylo možné volit reklamní kampaně tak, aby se zaměřily na nejpravděpodobnějšího zákazníka s použitím dat o spádových oblastech zákazníků, jejich věku, historii nákupů, apod.

Za tímto účelem vznikl proces ***získávání znalostí z databází*** (angl. *Knowledge Discovery in Databases* – KDD), nebo také zkráceně *dolování z dat* (angl. *Data Mining*). Celý proces je poměrně náročný, neboť příslušné algoritmy jsou prováděny nad velkým objemem dat. Proto je vhodné získávání znalostí řešit automatizovaně pomocí komplexních nástrojů, které analytika provedou celým procesem dolování a poskytnou mu maximální možnou podporu. Jedná se ale spíše o komplexní komerční projekty.

Proto je na Fakultě informačních technologií Vysokého učení technického v Brně vyvíjen systém na přenositelné platformě Java, který by umožnil studentům si celý proces dolování prakticky vyzkoušet. Ze systému, jehož vývoj započal v roce 2006, je implementováno jádro s možností připojení dolovacích modulů. Náplní této práce bude systém zdokonalit v oblasti předzpracování dat.

Zanesením problematiky získávání znalostí z databází do širšího kontextu se zabývá **druhá kapitola** této práce. Následně definuje jednotlivé kroky dolovacího procesu a stručně je charakterizuje.

V **kapitole třetí** je podrobně charakterizován krok týkající se předzpracování dat. Podrobně jsou popsány všechny problémy, které se řeší. Především jde o postupy při čištění dat, jejich transformací, redukcí a statistické analýze.

Současný stav projektu je charakterizován v **kapitole čtvrté**. Nejprve jsou popsány základní technologie, které jsou dále v práci použity. Následně je stručně shrnuta historie projektu a stav, do kterého byl systém doveden. Charakterizovány jsou stěžejní části jádra, jeho datové struktury a přístup k vytváření nových komponent GUI.

Ze získaných poznatků v rámci teoretického úvodu práce a současného stavu se **kapitola pátá** zabývá návrhem rozšíření systému především v oblasti předzpracování dat a návrhem odstranění nedostatků, které s problematikou souvisí a doposud nebyly vyřešeny.

Šestá kapitola shrnuje práce, které byly provedeny na již existujících komponentách. Je zde popsáno rozšíření funkčnosti komponenty výběru dat. Hlavními body jsou ale reimplementace komponenty VIMEO a transformací s důrazem na optimalizaci výkonu jejich provádění.

V **sedmé kapitole** je popsána implementace nových komponent předzpracování, které

v systému doposud absentovaly. Jedná se o komponentu pro analýzu a vizualizaci dat. Další komponentou, jejíž vývoj je v kapitole popsán, je komponenta redukce a rozdělení dat.

Kapitola osmá se zabývá dalšími změnami, které musely být v rámci jádra provedeny. Jedná se především o problematiku vyčlenění databázové vrstvy z jádra systému, vícevláknového běhu aplikace a dalších.

Další možný vývoj a rozšíření systému je popsán v **kapitole deváté** a **závěrečná kapitola** shrnuje rozšíření provedená v systému při řešení tohoto diplomového projektu a diskutuje výsledky práce.

Práce navazuje na závěry Semestrálního projektu, který se zabýval prvními dvěma body zadání. Jeho výsledky odpovídají přibližně druhé až páté kapitole této práce, konkrétně charakteristice získávání znalostí z databází, analýze předzpracování dat, rozboru původního systému a diskuzi jeho možných rozšíření.

Kapitola 2

Získávání znalostí z databází

Samotné vymezení pojmu „získávání znalostí z databázi“ je jednoduché – jedná se o získávání nějakých netriviálních a užitečných informací z databáze, které jsme do ní však explicitně neukládali a nelze je získat triviálním dotazem. Toto si můžeme představit tak, že pokud například uchováváme seznam navštívených webových stránek uživatelem, tak získáme informaci, že pokud uživatel přečte nějaký článek, pak pravděpodobně přečte i nějaký jiný konkrétní článek. Primárním účelem databáze je zde uchovávat data o navštívených stránkách, avšak sekundárně se z těchto dat dají zjistit informace o chování uživatelů.

Samotné získávání znalostí z dat však není jednoduchou atomickou operací. Jedná se spíše o ucelený proces, kde jeho výstupním produktem je objevení „uživatelsky zajímavé“ informace. Tento proces popíši v následujících podkapitolách. Více ke každému tématu se pak lze dočíst v [2].

2.1 Dolování z dat jako proces

Jak bylo naznačeno v úvodu kapitoly, získávání znalostí z databází je ucelený proces.

Charakteristika jednotlivých kroků procesu:

1. **čištění dat** – odstranit chybná a zašuměná data,
2. **integrace dat** – pokud máme více různých zdrojů dat (různých databází, tabulek),
3. **výběr dat** – označíme pouze data, která jsou relevantní pro dolování,
4. **transformace dat** – úpravy dat spočívající ve změně rozsahu hodnot atributů, vytvoření agregovaných atributů z jiných atributů,
5. **dolování dat** – aplikace určitého algoritmu nad daty pro získání vzorů charakterizujících data (vytvoření modelu dat),
6. **vyhodnocení vzorů** – ne všechny výsledky dolování dat jsou uživatelsky zajímavé, proto je třeba z výsledků identifikovat ty nejzajímavější,
7. **prezentace znalostí** – úprava získaných znalostí do podoby prezentovatelné koncovému uživateli (vizualizační, prezentační techniky, ...).

První čtyři bodu procesu spadají právě do oblasti tzv. **předzpracování dat**, které bude těžištěm této diplomové práce. V následujících podkapitolách se podrobněji zabývám

jednotlivými kroky procesu s uvedením základních technik, které se v konkrétní oblasti používají.

2.2 Předzpracování dat

Jde o velmi důležitou oblast procesu. Kvalita vstupních dat může kvalitu výsledku procesu dolování do znační míry ovlivnit. Proto je nezbytné data vhodným způsobem předzpracovat – jedná se především o integraci dat z více zdrojů, odstranění šumu v datech, omezení sady vstupních dat do algoritmu a podobně.

Jelikož na oblast předzpracování dat je tato práce zaměřena, je tato problematika podrobně rozebrána v samostatné kapitole 3.

2.3 Dolování dat a typy dolovacích úloh

Vzhledem k rozmanitosti datových zdrojů je nezbytné určit, jaký typ dolovací úlohy užít, resp. jaký typ vzoru (modelu) v datech hledat. Typy úloh lze rozčlenit do dvou základních kategorii na

- **deskriptivní** (charakterizuje základní vlastnosti dat)
- a **prediktivní** (používá analýzy dat v databázi k predikci dalšího chování).

Avšak i přes definici dolovacích úloh musejí být nástroje pro dolování značně flexibilní, neboť v řadě případů uživatel ani není schopen říct, jaké vyskytující se vzory v datech jej mohou zajímat, případně na jaké úrovni abstrakce a podobně.

V dalších podkapitolách jsou jednotlivé typy dolovacích úloh charakterizovány.

2.3.1 Popis konceptu/třídy

Různá data asociujeme k třídám a ty pak popíšeme. To lze provést pomocí

- **charakterizace** – sumarizujeme charakteristiky a vlastnosti cílové třídy,
- **diskriminace** – porovnáváme vlastnosti dat jedné třídy vůči jedné či více vlastnostem ostatních tříd.

2.3.2 Dolování frekventovaných vzorů a asociačních pravidel

Frekventovaným vzorem obvykle myslíme nějakou množinu vzorů, která se v datech vyskytuje dostatečně často. Z těchto vzorů pak určujeme samotná **asociační pravidla**. Pro rozlišení, kde je hranice zajímavých pravidel, je pro asociační pravidlo definována *minimální podpora* a *minimální spolehlivost*. Multidimenzionální asociační pravidlo může vypadat například takto:

$$pohlavi(X, "muz") \wedge prijem(X, " > 25000") \rightarrow koupi(X, "pocitac")$$
$$[podpora = 20\,\%, spolehlivost = 85\,\%]$$

Pro získání asociačních vzorů se užívá například algoritmus *Apriori* s řadou heuristik, nebo například algoritmus *FP tree*. Tyto metody jsou podrobně popsány v [2], kapitole páté.

2.3.3 Klasifikace a predikce

Klasifikace je proces, při kterém je nalezen model dat, který data rozděluje do tříd. Použití pak spočívá v určení třídy k datům, jejichž třída je nám neznámá.

Například klasifikujeme vlastnost člověka splácet půjčku podle jeho příjmů, zaměstnání, věku, apod. Model vytvoříme na základě dat o stávajících klientech a chování klientů nových pak lze již předpovědět a na základě toho rozhodnout o žádosti o půjčku.

Ke klasifikaci lze užít metod *rozhodovacího stromu*, *neuronových sítí*, *Bayesovských klasifikátorů*, *SVM*, *klasifikátorech založených na genetických algoritmech* a dalších.

Predikce narozdíl od klasifikace využívá sestaveného modelu k předpovědi určité **spojité** hodnoty spojitého atributu. Pro tyto účely se používají metody *jednoduché lineární regrese*, *nelineární regrese* a dalších.

2.3.4 Shluková analýza

Podobně jako klasifikace i shluková analýza data rozděluje. Avšak u klasifikace jsou jasně vymezené třídy, do kterých data rozdělujeme. Shluková analýza tyto třídy vytváří na základě vzdálenosti mezi jednotlivými objekty v prostoru atributů tak, aby si v jedné třídě byly objekty co nejpodobnější. Metody jsou výhodné v tom, že uživatel příliš nemusí znát povahu dat.

Shlukové analýzy se dá dobře využít i v předzpracování dat pro určení odlehlých hodnot v datech, kdy většina vstupních dat vytvoří s ostatními shluky, pouze nejodlehlejší hodnoty se k žádnému shluku nepřiřadí (i když záleží na druhu shlukovací metody, u některých naopak dojde po shlukování k poškození výsledku shlukování). Zpracování takto identifikovaných hodnot je diskutováno v kapitole věnované předzpracování dat.

2.3.5 Analýza odlehlých objektů

V některých případech ale nemusí být snahou odlehlé hodnoty identifikovat a odstranit jako zašuměné. Někdy naopak jejich vyhledání je důležité k identifikaci nějakého nestandardního chování – například pokud se někdo snaží uskutečnit podvod.

K jejich určení lze použít například statistické metody nad daty či vzdálenostní metriky.

2.3.6 Evoluční analýza

V některých aplikacích máme objekty, které se v čase mění. Úkolem dolování je pak zjišťování modelu trendů těchto změn objektů. Pro tyto účely lze použít jak standardní metody, tak i některé speciální.

2.4 Vyhodnocení a prezentace výsledků

Výsledkem dolovací úlohy je množina vzorů popisujících data. Z těchto vzorů je především třeba vybrat ty, které jsou zajímavé a užitečné pro uživatele. Pro objektivní posouzení zajímavosti vzorů existují právě míry podpory a spolehlivosti zmíněné v části 2.3.2.

Pro koncového uživatele by ale výstup algoritmů nebyl příliš čitelný, proto je třeba získané vzory vhodně zpracovat – například vytvořit sadu čitelných pravidel popisujících data, nebo získané vzory vhodně vizualizovat (tabulky, grafy, ...).

Kapitola 3

Předzpracování dat

Pokud budeme chtít analyzovat data z nějaké databáze, musíme počítat s tím, že tato data nebudou pravděpodobně k přímé analýze dolovacími algoritmy vhodná. Budou zašuměná, rozdělena do více databází či datových skladů, hodnoty jejich atributů budou mít různé rozsahy a data budou dimenzionálně i obsahově značně rozsáhlá. Všechny uvedené problémy se snaží odstranit nebo alespoň eliminovat proces předzpracování dat. Celý tento proces je značně výpočetně náročný, neboť většina operací se provádí nad velkým množstvím „surových" dat a bude se třeba zabývat i efektivním řešením těchto úloh.

V následujících podkapitolách budou popsány techniky, jakými lze nejdříve analyzovat data v databázi, provést čištění, integraci z více zdrojů, transformace dat a zredukovat soubor dat k analýze. Témata jsou podrobně rozvedena v [2] a [20].

3.1 Charakteristika a statistiky nad vstupním souborem dat

Aby si uživatel vytvořil základní představu o datech, která bude analyzovat, musí se v procesu předzpracování vyskytnout podpora pro statistické vyhodnocení vlastností dat v databázi.

3.1.1 Míry středu – průměr, medián a modus

Průměr je nejčastěji používaným vyjádřením středu vzorku dat. Je definován vztahem

$$\overline{X} = \frac{\sum_{i=1}^{N} x_i}{N}. \tag{3.1}$$

Nevýhodou jeho použití ovšem je, že může být snadno ovlivněn výskytem odlehlých hodnot v datech. Proto se s výhodou používají jiné míry.

Medián je další mírou středu, kde hodnota mediánu odpovídá hodnotě, která se v seřazené posloupnosti dat nachází uprostřed v případě lichého počtu hodnot nebo průměru dvou centrálních prvků v případě sudého počtu prvků.

Modus opět popisuje střed souboru dat tak, že jeho hodnota je hodnotou nejčastěji se vyskytujícího prvku. Zde není vyloučeno, že stejnou frekvenci výskytu má více hodnot, nebo se žádná hodnota nevyskytuje vícekrát a v tom případě modus neexistuje.

Mezi výše uvedenými středovými hodnotami existuje empirická závislost vyjádřitelná vztahem

$$mean - mode = 3 * (mean - median). \tag{3.2}$$

3.1.2 Míry variace dat

V pravděpodobnostním počtu se pro charakteristiku variability rozdělení pravděpodobnosti náhodné veličiny používá tzv. *druhý centrální moment* zvaný též **rozptyl**. Ten je definován vztahem

$$\sigma^2 = \frac{1}{N}\sum_{1}^{N}(x_i - \overline{x})^2 = \frac{1}{N}(\sum_{1}^{N} x_i^2) - \overline{x}^2, \tag{3.3}$$

statistickou disperzi vyjadřuje **empirická směrodatná odchylka** definovaná pomocí rozptylu z předchozího vztahu jako

$$\sigma = \sqrt{\sigma^2}. \tag{3.4}$$

Další veličiny, kterými lze popsat míru variace, je i **minimum** a **maximum** ze souboru hodnot, nebo například **mezikvartilová vzdálenost** označovaná jako IQR, která bude definovaná v podkapitole 3.1.3.

Více k popisu náhodné veličiny lze nalézt v [1], [17] a dále [18], [19].

3.1.3 Kvantily

Zdroj [16] definuje **kvantily** jako body z hranic pravidelných intervalů kumulativní distribuční funkce náhodné veličiny. To prakticky znamená, že v případě q-kvantilu rozdělíme seřazené hodnoty na q pravidelných intervalů pomocí $(q-1)$ hraničních hodnot – ty označíme postupně jako k-tý kvantil.

Mezi často užívaná q patří hodnoty 100 (hodnoty označíme jako *percentily*), 10 (*decily*) nebo 4 (*kvartily*).

Pomocí **kvartilu** se definuje již zmíněná **mezikvartilová vzdálenost (IQR)** jako

$$IQR = Q_3 - Q_1. \tag{3.5}$$

Mezikvartilové vzdálenosti lze využít k definici *odlehlých hodnot* tak, že jsou to hodnoty vzdálené $1,5 * IQR$ nad Q_3 nebo pod Q_1.

3.1.4 Vizualizační techniky pro charakteristiky dat

Pro vytvoření si představy o rozložení dat se často používají různé vizualizační techniky, které usnadní interpretaci zmíněných veličin popisujících hodnoty atributů souboru dat.

Poznámka Pro následující podkapitoly se v příkladech grafů používá následující soubor dat, kde každá dvojice představuje dvojici atributů [A,B]: *[1,21], [1,23], [1,22], [3,25], [5,30], [6,30], [6,35], [6,36], [5,30], [9,50], [10,50], [10,70], [11,80], [11,80], [11,83], [15,90], [15,95], [30,100].*

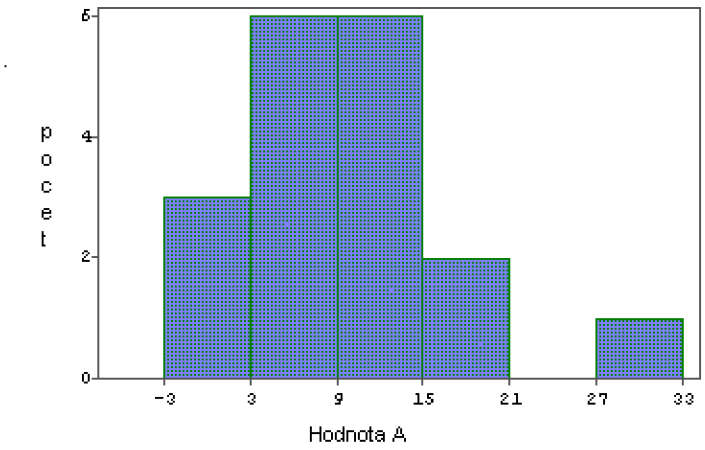
Grafy jsou vytvořeny za pomoci nástroje *SAS Enterprise Miner*.

Histogram

Histogram poskytuje základní možnost pro zobrazení rozložení hodnot nějakého atributu. Pro numerický atribut nejčastěji rozdělíme interval hodnot datového souboru na stejně široké části a histogramem zobrazíme počet hodnot v kažném intervalu. Pro kategorický atribut je sumarizace do sloupců provedena pro možné hodnoty kategorického atributů. V tomto případě mluvíme spíše o **sloupcovém grafu**. Počet lze vyjádřit buď absolutně nebo procentuálně.

Histogramy (pro numerické atributy) lze vytvářet různými způsoby:

- **Intervaly stejné šířky** – šířka každého intervalu je stejná. Možná podoba tohoto histogramu je na obrázku 3.1.
- **Intervaly stejné hloubky (frekvence)** – intervaly jsou vytvořeny tak, že mají (přibližně) stejný počet hodnot.



Obrázek 3.1: Histogram se stejně širokými intervaly atributu A

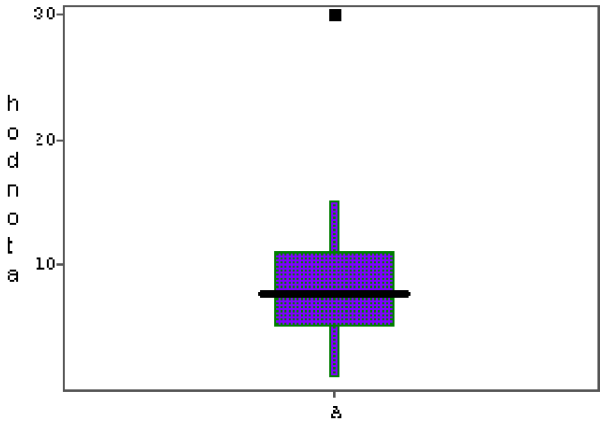
Krabicový graf

Krabicový graf přehledně zobrazuje rozložení hodnot v souboru tím, že do jednoho grafu zobrazuje odlehlé hodnoty pomocí využití IQR, meze kvartilů Q_1 a Q_3 a medián. Struktura tohoto grafu je velmi výhodná pro porovnávání rozložení hodnot více atributů v jednom grafu.

Příklad grafu je na obrázku 3.2. Výrazný obdélník představuje interval hodnot mezi kvartily Q_1 a Q_3, vodorovná úsečka značí výskyt mediánu atributu. Užší obdélníky nad a pod hlavním obdélníkem označují intervaly $1,5 * IQR$ nad Q_3 nebo pod Q_1. Body mimo tento interval jsou označeny výskyty odlehlých hodnot.

Kvantilový graf a q-q graf

Kvantilový graf popisuje opět rozložení hodnot. Předpokládejme, že máme seřazený soubor hodnot atributu a_i pro $i = 1..n$, kde n je počet hodnot atributu. Definujme pak funkci

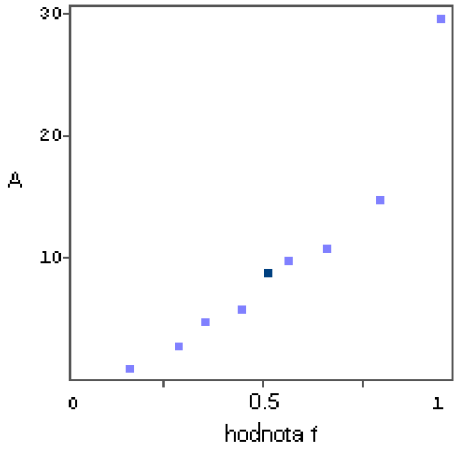


Obrázek 3.2: Krabicový graf atributu A

$$f_i = \frac{i - 0,5}{n}. \tag{3.6}$$

Hodnoty funkce f_i vynesené na osu x a odpovídající hodnoty a_i vynesené na osu y grafu nazveme kvantilovým grafem.

Výsledkem je graf, který lze interpretovat jako procentuální počet hodnot menších než hodnota na ose y. Příklad kvantilového grafu je na obrázku 3.3.



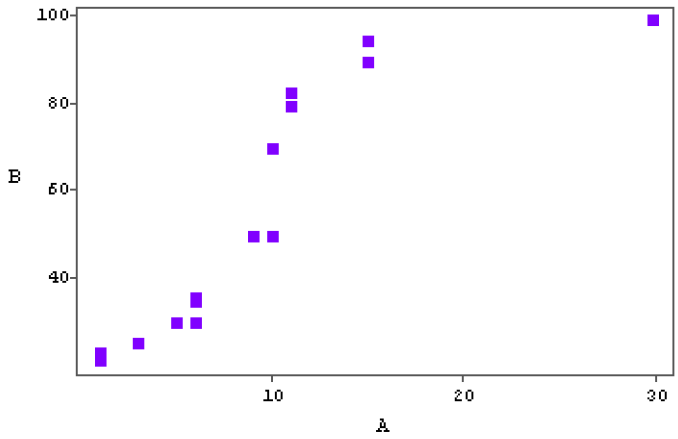
Obrázek 3.3: Kvantilový graf atributu A (přibližný)

Obdobou kvantilového grafu pro 2 atributy je tzv. **Q-Q graf**. Ten vytvoříme tak, že na osy x a y vyneseme hodnoty atributů tak, aby si pro tyto hodnoty odpovídaly příslušné funkce f_i každého atributu.

Bodový graf

Bodový graf nejjednodušším způsobem zobrazuje vztah 2 atributů. Jedna osa představuje hodnoty jednoho atributu, druhá osa hodnoty druhého atributu. Bod ve vymezené rovině

představuje hodnoty obou atributů této dvojice. Příklad bodového grafu je na obrázku 3.4.



Obrázek 3.4: Bodový graf atributu A

3.2 Čištění dat

Data, která jsou uložena v databázi, musí většinou projít procesem čištění. V praxi totiž v databázi u některých záznamů chybějí hodnoty některých atributů nebo jsou tam hodnoty, které jsou zašuměné. V této kapitole budou popsány způsoby, jak lze tyto problémy řešit.

3.2.1 Chybějící hodnota

Problém chybějících dat je třeba řešit především proto, že některé algoritmy by se s chybějící hodnou nedokázaly vyrovnat, nebo by byl výsledek zkreslený. K řešení problému s chybějící hodnotou lze užít tyto přístupy:

- **Ignorovat celou n-tici** – tento způsob není nejvhodnější hned z několika důvodů. Ignorováním celé n-tice především ztrácíme důležitá data k analýze, ale může taky nastat situace, kdy chybějící hodnoty jsou v nezanedbatelném počtu n-tic a nezbyl by dostatečný vzorek dat k dolování.
- **Nechat doplnit hodnoty manuálně uživatele** – není příliš použitelný přístup, pokud je chybějících hodnot hodně. Navíc vyžaduje, aby měl uživatel dostatek znalostí k doplnění nejsprávnější hodnoty.
- **Dosadit zvolenou globální konstantu** – v tomto případě provede algoritmus automatickou náhradu za nějakou hodnotu například významu *Unknown*, ale při častém použití této hodnoty by například klasifikační algoritmus mohl na základě této hodnoty vytvořit klasifikační pravidlo a zkreslit tak výsledek.
- **Doplnit průměrnou hodnotou atributu** – algoritmus spočítá průměr hodnot atributu a použije jej k doplnění chybějících hodnot.
- **Doplnit průměrnou hodnotu atributu dat patřících do stejné třídy jako n-tice s chybějící hodnotou** – nepočítá průměr všech hodnot atributu v souboru dat, ale pouze těch, u kterých třída odpovídá třídě n-tice s chybějící hodnotou.

- **Doplnit nejpravděpodobnější hodnotu** – metoda zavádějící do problematiky již jistý stupeň inteligence, kde se algoritmus snaží na základě ostatních atributů n-tice předpovědět (predikovat) chybějící hodnotu n-tice.

  V tomto přístupu v podstatě použijeme metod dolování dat již v procesu předzpracování (užít lze klasifikační a predikční metody – např. rozhodovací strom, regrese, ...).

3.2.2 Zašuměná data

Při zadávání dat do databáze může dojít k nejrůznějším anomáliím (chyba zadání, senzoru, ...) jejichž výsledkem vznikají zašuměné hodnoty v datech. Proto se snažíme provést *vyhlazení dat* pomocí následujících technik.

Plnění (binning)

Tato technika spočívá v tom, že seřazenou posloupnost hodnot rozdělíme do daného počtu košů. Rozdělení provádíme nejčastěji tak, aby každý koš obsahoval přibližně stejný počet hodnot. Potom provedeme vyhlazení nějakou z následujících metod:

- **Vyhlazení koše průměrem** – tato metoda nahradí všechny hodnoty v koši jejich průměrem.
- **Vyhlazení koše mediánem** – metoda nahradí všechny hodnoty v koši hodnotou mediánu příslušného koše.
- **Vyhlazení pomocí hranic koše** – v koši určíme minimální a maximální hodnoty, které reprezentují hranice. Každá hodnota v koši je nahrazena jednou z těchto dvou hodnot podle toho, ke které je blíže.

Regrese

V tomto přístupu uvažujeme, že pro dva nebo více atributů, které jsou spolu svázány, lze užít k vyhlazení atributu **lineární regrese**, příp. **vícenásobné lineární regrese**. Lineární regrese aproximuje hodnoty atributu pomocí nalezené lineární funkce mající tvar

$$y = a + bx, \tag{3.7}$$

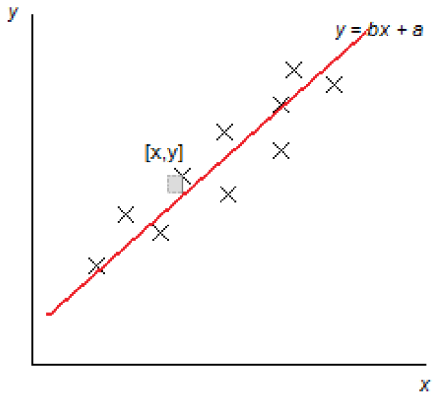
kde x je hodnota atributu, který známe a y hodnota vyhlazovaného (či neznámého) atributu. Koeficienty rovnice a a b se určují metodou nejmenších čtverců tak, že

$$b = \frac{\sum_{i=1}^{S}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{S}(x_i - \overline{x})^2} \tag{3.8}$$

$$a = \overline{y} - b\overline{x}, \tag{3.9}$$

kde $\overline{x}$ je průměrná hodnota atributu x a $\overline{y}$ je průměrná hodnota atributu y. Geometrický význam lineární regrese je znázorněn na obrázku 3.5.

Vícenásobná lineární regrese se narozdíl od jednoduché používá k určení hodnoty y více než jedním předpovídajícím atributem.



Obrázek 3.5: Jednoduchá lineární regrese

Regresi lze užít k více aplikacím předzpracování dat. Zde ji zmiňuji v souvislosti s vyhlazováním zašuměných dat. Obdobně jde ale použít i k predikci chybějící hodnoty i přesto, že nejsou známy všechny sumární hodnoty. Více se této problematice věnuje [10].

Shlukování

Přístup podobný nalezení odlehlých hodnot volí vhodná metoda shlukování. Pokud uvažujeme, že zašuměná hodnota je hodnota odlehlá od ostatních (pomyslně spadajících do jednotlivých shluků – tříd), pak řešíme úlohu jako shlukovou analýzu a hodnoty, které nebudou náležet žádnému shluku, označíme jako zašuměné. S takto identifikovanými hodnotami lze provést nahrazení jinou hodnotou určenou automaticky, ignorovat ji, nebo nechat upravit uživatelem.

3.3 Transformace dat

Pokud již máme soubor dat kompletní, jsou odstraněny chybějící hodnoty a vybrána data k dolování, měli bychom ještě data vhodně transformovat.

Transformace jsou prováděny z různých důvodů, čemuž odpovídají použité způsoby.

3.3.1 Vyhlazování

Problematika byla zmíněna v souvislosti s odstraněním šumu v datech v podkapitole 3.2.2.

3.3.2 Agregace

Data (resp. hodnoty) lze přes jednu či více dimenzí agregovat. Typicky se provádí u datových skladů.

3.3.3 Generalizace

Data v databázi mohou být uložena na jemnější úrovni *konceptuální hierarchie*, než je třeba. Proto můžeme hodnoty atributů transformovat například z numerické hodnoty *výšky člověka* na diskrétní hodnoty *malý*, *střední*, *velký*.

3.3.4 Normalizace

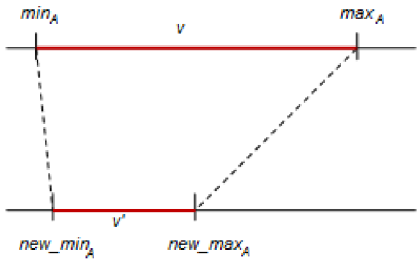
Problematika souvisí například s různými rozsahy hodnot jednotlivých atributů. Kupříkladu mějmě atribut s běžným rozsahem hodnot 0-1 a jiný atribut s rozsahem 0-1000. Pokud bychom na tyto atributy použili metodu shlukování založenou na vzdálenosti mezi body, byla by váha absolutní vzdálenosti pro tyto atributy odlišná jen z důvodu řádově rozdílných rozsahů.

Proto je snaha transformovat numerické atributy tak, aby měly stejné rozsahy hodnot. Užít lze tyto metody:

- **Min-max normalizace** – provádí jednoduchou lineární transformaci vyjádřenou obrázkem 3.6 z rozsahu hodnot $\langle min_A, max_A \rangle$ do nového (zvoleného) intervalu $\langle new\_min_A, new\_max_A \rangle$ dle vzorce:

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A \tag{3.10}$$

  Typicky volený interval může být $\langle 0,0; 1,0 \rangle$.



Obrázek 3.6: Min-max normalizace

- **Normalizace z-score** – tato metoda může být užita například tehdy, pokud nejsou známy meze intervalu hodnot. Pokud označíme průměr hodnot zkoumaného atributu A jako $\overline{A}$, standardní odchylku hodnot atributu A jako σ_A, pak definujeme transformaci hodnoty v na v' jako

$$v' = \frac{v - \overline{A}}{\sigma_A}. \tag{3.11}$$

- **Normalizace dekadickou změnou měřítka** – provádí normalizaci do intervalu $(-1, 1)$ posunem desetinné čárky všech hodnot atributu

$$v' = \frac{v}{10^j}, \tag{3.12}$$

  pro nejmenší možné $j \in \mathbb{N}_0$, aby platilo $max(|v'|) < 1$.

3.3.5 Konstrukce atributů

Ze stávajících atributů vytváříme nové. Například pokud by nás pro dolování zajímal výsledek studenta z hlediska toho, zda prospěl, či ne a měli k dispozici v databázi počet získaných bodů, mohli bychom zkonstruovat nový dvouhodnotový atribut *absolvoval* odvozený od počtu získaných bodů.

3.4 Redukce dat

Pokud je množina vstupních dat příliš obsáhlá, je vhodné ji určitým způsobem redukovat.

3.4.1 Přístupy k redukci dat

Vybrané metody redukce dat spočívají v následujících principech (ostatní metody lze nalézt v [2]).

- **Výběr podmnožiny atributů** – spočívá v nezahrnutí irelevantních atributů do analýzy a tím zmenšení dimenzionality dat. Přestože robustní algoritmy dolování z dat by neměly být příliš ovlivněny atributy bez informační hodnoty, jejich zpracování by bylo zbytečně časově náročné. Princip těchto metod spočívá v postupném odstraňování či přidávání jednotlivých atributů k množině pro analýzu.
- **Redukce dimenzionality** – používá odlišný přístup, kdy jsou data vhodným způsobem kódována a tím je snížena jejich dimenzionalita. Jde o konstrukci aproximace původních dat například pomocí metod *diskrétní vlnkové transformace* nebo metody *PCA (Principal Components Analysis)*.
- **Redukce počtu hodnot** – data jsou nahrazena jejich menší reprezentací. Z této oblastí bych zmínil především **vzorkování** *(sampling)*, neboť jej bude nezbytné použít v implementační fázi například pro zobrazení souhrných charakteristik dat.
- **Diskretizace dat a konceptuální hierarchie** – diskretizaci dat aplikujeme na atributy s numerickými hodnotami, abychom snížili velký počet numerických hodnot na menší počet intervalů. Pokud použijeme diskretizaci rekurzivně na jeden (i kategorický) atribut vícekrát, dostáváme tzv. *konceptuální hierarchii*, kde můžeme využít různých úrovní abstrakcí dělení. Některé vybrané metody, které mohou být později implementovány jako součást této diplomové práce, jsou popsány v podkapitole 3.4.3. Více pak v literatuře [2].

3.4.2 Vzorkování

Spočívá v tom, že vybereme nějakou podmnožinu n-tic velkého souboru dat, která celou množinu reprezentuje. Většinou je snaha, aby vzorek dat měl charakteristiky podobné celé množině dat. Volí se tyto přístupy:

- **Jednoduchý náhodný vzorek bez návratu (SRSWOR)** – jednoduchý náhodný výběr s n-tic, kdy je kopírovaná n-tice z celého vzorku dat odstraněna a nemůže být tedy znovu vybrána.

- **Jednoduchý náhodný vzorek s návratem (SRSWR)** – podobně jako předchozí přístup, pouze je n-tice ponechána v původní celé množině dat a v dalším výběru může být opět zkopírována do vzorku dat.
- **Vzorek shluků** – data jsou rozdělena na M vzájemně disjunktních shluků, z nichž je potom vybráno s shluků jednoduchým náhodným výběrem.
- **Stratifikovaný vzorek** – celý vzorek dat je rozdělen na tzv. *strata*. což jsou vzájemně disjunktní množiny dat. Strata mohou být vytvořena například podle příslušnosti dat do tříd. Z každého strata se vyberou data jednoduchým náhodným výběrem tak, že zůstanou zachovány pravděpodobnosti výskytu dat náležejících do jednotlivých strat.

3.4.3 Diskretizace kvantitativních atributů

Rozdělení do košů – princip této metody byl charakterizován v části 3.2.2 pro čištění dat. Proces diskretizace spočívá v rozdělení do košů a následně je každý koš chápán jako hodnota diskrétního atributu, kde hodnota je buď průměr, či medián koše.

Analýza histogramu – data jsou opět rozdělena na skupiny tak, aby každá měla stejný počet hodnot (*tzv. equal-frequency*), nebo tak, že intervaly dělení jsou stejně široké (*tzv. equal-width*). Je zřejmé, že tento proces úzce souvisí s problematikou histogramů (tomuto tématu se věnovala více podkapitola 3.1.4).

Shluková analýza – způsob, kdy je shlukovací algoritmus aplikován na nějaký atribut a tak jsou přirozenou cestou objeveny shluky v datech představující diskrétní hodnoty nového atributu.

Diskretizace přirozeným dělením – je zajímavou metodou popsanou podrobně v [2] a vychází z toho, že uživatel pravděpodobně chce raději přirozeně rozdělené (diskretizované) hodnoty, kdy jsou intervaly vymezeny co nejlogičtěji – nejlépe tedy intervaly po číslech jako 10, 50, 100, apod.

3.4.4 Analýza korelace atributů

Při hledání redundantních atributů může být značně výhodné znát korelaci jednotlivých atributů datového souboru. Korelaci lze určit pomocí tzv. *korelační analýzy.*

Pro **numerické atributy** vyjádříme *korelační koeficient* – $r_{A,B}$ jako

$$r_{A,B} = \frac{\sum_{i=1}^{N}(a_i b_i) - N\overline{AB}}{N\sigma_A\sigma_B}, \tag{3.13}$$

kde N je počet dvojic, a_i a b_i je i-tá dvojice atributů A a B, $\overline{A}$ a $\overline{B}$ označuje průměrné hodnoty atributů a σ_A aσ_B jsou standardní odchylky atributů A a B. Výsledkem je hodnota korelačního koeficientu $r_{A,B}$ v intervalu $-1 \leq r_{A,B} \leq 1$. Hodnota 0 znamená nekorelaci atributů, záporná hodnota zápornou korelaci a kladná hodnota kladnou korelaci.

Při odvození vycházíme z definice hodnoty *tzv. kovariace* dané vztahem

$$cov_{A,B} = E((A - E(A)) * (B - E(B))), \tag{3.14}$$

kde $E(X)$ značí *střední hodnotu* veličiny (resp. atributu) X.

V případě **kategorických atributů** můžeme použít ke zjištění korelace χ^2*-test dobré shody*. K výpočtu je třeba nejdříve sestavit *kontingenční tabulku* pro zjišťované atributy A a B. Kontingenční tabulka má řádky (resp. sloupce) pro každou různou hodnotu a_i (resp. b_i) vyskytující se v souboru dat a na průsečíku v tabulce je číslo značící počet n-tic s výskytem právě těchto hodnot. Hodnotu χ^2 určíme jako

$$\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{l} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{3.15}$$

kde o_{ij} je empirická četnost výskytu a e_{ij} očekávaná definovaná jako

$$e_{ij} = \frac{count(A = a_i) * count(B = b_j)}{N}, \tag{3.16}$$

kde N je počet n-tic a hodnota $count(A = a_i)$ značí počet výskytů hodnoty a_i pro atribut A (analogicky $count(B = b_j)$ pro atribut B).

Tento statistický test ověřuje hypotézu, že *hodnoty atributů A a B jsou nezávislé*. Hodnota χ^2 se zvětšuje čím více se počty výskytů liší od očekávané hodnoty. Zjištěnou hodnotu χ^2 test srovnává s hodnotou $\chi_{(1-\alpha)}(n)$, kde α je *hladina významnosti* testu a parametr n značí *stupeň volnosti* vyjádřitelný jako

$$n = (k - 1)(l - 1). \tag{3.17}$$

Hodnotu lze vyčíst buď z tabulek, nebo ji lze aproximovat pomocí vztahu (podrobnosti v [11])

$$\chi^2_P \approx n \left(1 - \frac{2}{9n} + u_P \sqrt{\frac{2}{9n}}\right)^3, \tag{3.18}$$

kde u_P jsou odpovídající kvantily *standardizovaného normálního rozdělení* $N(0, 1)$. Tuto hodnotu lze nalézt rovněž v tabulkách, nebo využijeme následující metodiky výpočtu (přesný postup lze nalézt v [1]).

Uvědomíme si, že potřebujeme najít r-kvantil k tak, aby platil vztah $\Phi(k) = r$, kde

$$\Phi(k) = \int_{-\infty}^{k} \frac{1}{\sqrt{2\pi}} * e^{-\frac{u^2}{2}} du. \tag{3.19}$$

Funkci e^x vyjádříme nekonečnou řadou, zintegrujeme a řadu omezíme na konkrétní počet členů dle požadované přesnosti a dostáváme například

$$\Phi(k) \doteq \frac{1}{\sqrt{2\pi}} * (k - \frac{k^3}{3 * 2^1} + \frac{k^5}{5 * 2^2} * \frac{1}{2!} - \frac{k^7}{7 * 2^3} * \frac{1}{3!} + \cdots + \frac{k^{21}}{21 * 2^{10}} * \frac{1}{10!}). \tag{3.20}$$

$\Phi(k)$ je hodnota odpovídající hodnotě P z hledaného u_P. Z rovnice tedy potřebujeme získat k, které je právě hledaným u_P. Pro nalezení použijeme iterativního zjišťování postupným přibližováním k hledané hodnotě například metodou půlení intervalu, kde víme, že $k \in \langle -6, 6 \rangle$ s přesností téměř 100 %.

Bohužel při praktických pokusech aproximační vzorec 3.20 nedosahoval potřebné přesnosti pro nízké počty členů a při vyšších přetékala celočíselná aritmetika.

Příklady a vysvětlení statistického testu čtenář nalezne v [2], dále v [11] a [1].

Kapitola 4

Systém pro dolování z dat na platformě NetBeans

Úkolem této práce je rozšíření stávající verze systému vyvíjeného na VUT FIT. Systém je vystavěn na dvouvrstvé architektuře, kde roli serveru představuje databázový server Oracle a klientskou částí je aplikace implementovaná v jazyce Java.

4.1 Použité technologie

Tato podkapitola objasní čtenáři základní technologie, které se dále používají v textu a je na nich založena implementace systému. Částečně vychází z [4], dále z konkrétních zdrojů zmíněných v textu.

4.1.1 Jazyk DMSL

DMSL *(Data Mining Specification Language)* je jazyk odvozený od XML. Cílem jazyka je postihnout celý proces dolování od dat až po výsledné znalosti. Je definován tak, aby umožňoval uložení všech podstatných informací o procesu dolování [3].

DMSL je založen na pěti základních elementech:

Datový model reprezentuje schéma platformě nezávislých vstupních dat.

Model dolování dat reprezentuje vhodně upravená data určená jako vstup do dolování.

Doménové znalosti umožňují zadefinovat znalosti o doméně v libovolném jazyce.

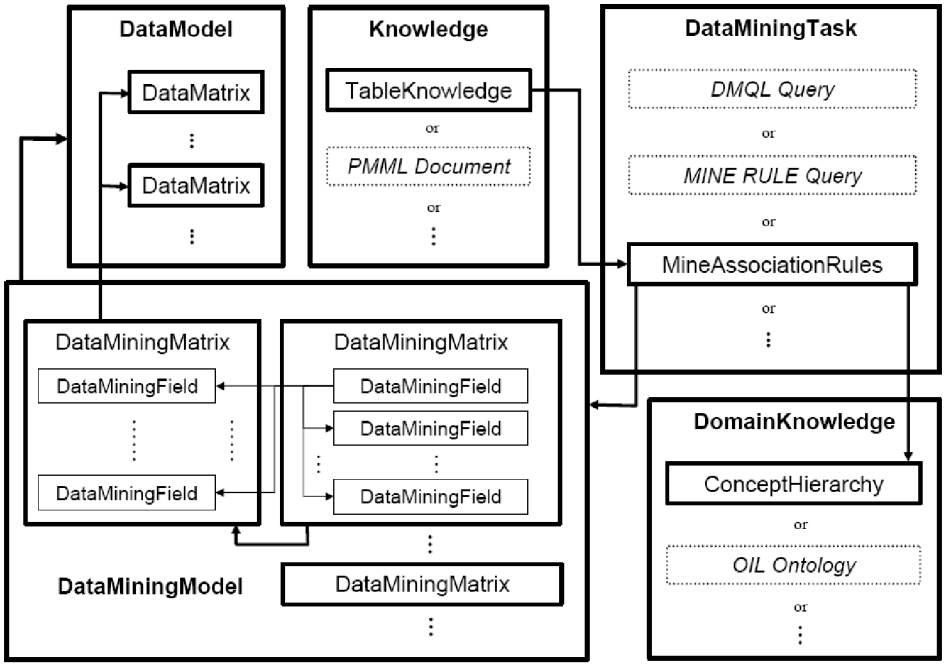
Úloha dolování dat specifikuje typ dolovací úlohy v daném jazyce.

Znalost je výstupem dolovací úlohy. Ty mohou být reprezentovány v libovolném jazyce.

Kompletní schéma závislostí DMSL elementů je na obrázku 4.1.

4.1.2 Podpora pro dolování dat systému Oracle

Databázový systém Oracle, na kterém je současný projekt vystavěn, obsahuje vlastní implementace algoritmů pro dolování z dat. V projektu je použito verze *Oracle 10g Release 2 značená (10.2)*, která zásadním způsobem přebudovala původní API. Počínaje touto verzí



Obrázek 4.1: Závislosti DMSL elementů, převzato z [3]

se technologie označuje jako *Oracle Data Mining (ODM)* nebo jak bylo uváděno v původních pracích *Data Mining Engine (DME)*.

V poskytovaných funkcí je pro tuto práci stěžejní především poskytování funkcí pro předzpracování dat. Jde především o algoritmy pro plnění, normalizace a ořezávání (angl. clipping). Kompletní API je dostupné na [9].

4.1.3 Platforma NetBeans a NetBeans Visual Library

NetBeans je Open Source projekt založený firmou *Sun Microsystems*. V rámci projektu existují 2 produkty – vývojové prostředí *NetBeans IDE* (pro systém není až tak podstatné) a vývojová platforma *NetBeans Platform*. Ta poskytuje modulární rozšiřitelný základ pro vytváření rozsáhlých aplikací. Více lze nalézt v [6] a [15].

NetBeans Visual Library vychází z původního systému Graph Library. Nyní poskytuje především nástroje pro zobrazování a podporu grafového modelování. Záměr je sjednotit zobrazování aplikací postavených na platformě NetBeans (jak API, tak i vzhledově) [7].

4.2 Vývoj systému

Vývoj systému započal v roce 2006 Ing. Doležal. Výsledkem jeho práce bylo jádro systému umožňující přidávat dolovací moduly. Systém byl však značně omezený, omezení spočívala především v definici samotných kroků dolovacího procesu, kde pořadí operací předzpracování a následného dolování bylo pevně stanoveno. Práci rozšířil o rok později Ing. Galét o kvalitní grafické uživatelské rozhraní založené na knihovně NetBeans Visual Library. Díky

této nadstavbě je možné definovat jednotlivé kroky procesu dolování formou spojování uzlů v grafu.

Poslední stávající implementaci přinesl Ing. Krásný [4] svou prací v roce 2008. Jeho práce spočívala v modifikaci jádra systému tak, aby byl skutečně využit celý potenciál aplikace, především ve směru definice dolovacího procesu a následné rozšiřitelnosti systému.

Modifikoval jádro aplikace tak, aby bylo možné volně spojovat komponenty grafu dolovacího procesu (s ohledem na logická omezení samotného procesu). Byla zrušena jádrem daná omezení při tvorbě grafů – například je možné z uzlu grafu směrovat výstup do více podstromů grafu.

Kvalitně byly přepracovány i datové struktury jádra uchovávající stav procesu dolování. Dříve mohlo dojít k nekonzistentním situacím, nyní je vše koncentrováno do jednoho místa v aplikaci s výjimkou situace práce s dialogovým oknem měnícím stav. Globální struktura není aktualizována průběžně, ale až v okamžiku uložení změn při uzavírání dialogu, aby se předešlo problémům se stornováním operace.

Podstatně je nutné zmínit ještě způsob ukládání projektu představujícího vytvořený proces dolování. Za tímto účelem je od počátku používán jazyk DMSL (charakterizován v podkapitole 4.1.1, definován prací [3]). Pro potřeby systému byla specifikace mírně upravena, aktuální změny jsou vystiženy v [4].

4.3 Poznatky ze stávající implementace

Stav systému zde analyzovaný odráží výsledky práce Ing. Krásného [4].

4.3.1 Jádro systému

Systém lze logicky rozčlenit na jádro (z dřívějšího pohledu jádro pro připojování modulů a k němu doimplementované uživatelské rozhraní) a jednotlivé dolovací moduly (v současné chvíli jsou rozpracovány). Aby mohl být modul připojen, byl implementován virtuální systém souborů, ze kterého systém zjišťuje existující moduly. Souborový systém je implementován pomocí XML souboru `layer.xml`. Tento způsob umožňuje připojit (či změnit) modul i bez nutnosti celé jádro rekompilovat.

Diagram tříd jádra a jejich vztahů je zobrazen na obrázku 4.2.

Datové struktury rozpracovaného projektu

Jádro systému uchovává data o právě rozpracovaném projektu dolování. Dříve tuto úlohu zajišťovala třída `Kernel.java`, nicméně z důvodu velice těsné spjatosti reprezentace procesu dolování (jeho parametrů apod.) byly tyto struktury přesunuty do třídy `Dmsl.java`. Tato třída tedy zajišťuje jak zpracování uloženého projektu ve formátu DMSL, tak i reprezentaci aktuálního projektu pomocí vytvořených datových struktur, které v sobě obsahuje. V podstatě datové struktury třídy `Dmsl.java` odpovídají definici dokumentu DMSL.

Připojení k databázi

Podstatnou částí, kterou řeší a následně poskytuje jádro, je připojení k databázi. Tato část projektu ještě není zcela funkční, neboť díky *timeoutu* může dojít k přerušení spojení. Aplikace vytváří 2 spojení. Jeden pro práci s daty samotnou aplikaci a druhý pro využití Oracle DME.

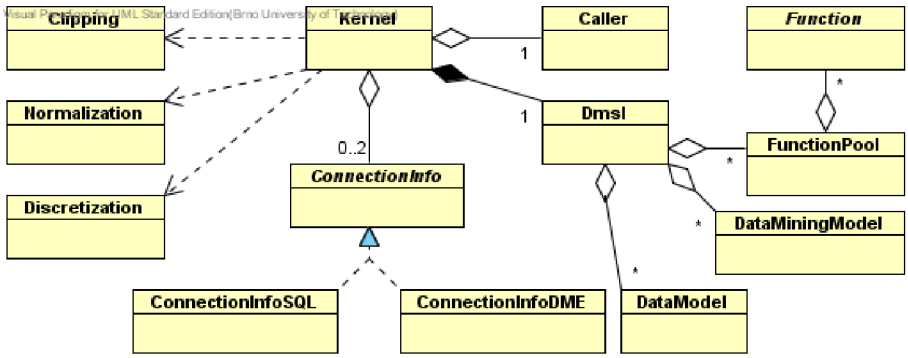
Vytváření funkcí, VIMEO a transformace

V rámci jádra je implementováno i zpracování funkcí pro čištění a transformaci dat. Definice interních funkcí je přímo uložena v DMSL. U takové funkce je v DMSL zapsána jak hlavička, tak i tělo funkce. Druhou skupinou jsou funkce externí. Nepřeložená funkce je v systému po načtení uchovávána z pomoci třídy `FunctionPool`. Z této reprezentace funkce je generován kód jazyka Java, přeložen a volán v podobě externí třídy za pomoci třídy `Caller`. Ve výsledku se tímto způsobem dosahuje zrychlení interpretace těchto funkcí.

Takto definované funkce lze užít jak pro čištění dat – v systému označovaném jako tzv. **VIMEO** funkce (Valid, Invalid, Missing, Empty, Outlier). Prakticky to znamená, že ve funkci nadefinujeme predikát, na jehož základě zjistíme, o jakou VIMEO hodnotu se jedná. Následně nastavíme chování pro tyto typy hodnot (např. použij nezměněný, ignoruj n-tici, atd.).

Podobným způsobem lze definovat funkci provádějící transformaci nad daty. Například tedy udělá sumu několika hodnot a vytvoří hodnotu nového atributu.

Jak již bylo zmíněno, samotný DME poskytuje některé algoritmy pro normalizaci, diskretizaci a clipping a u těchto transformací se volí tento přístup. Podrobněji bude toto téma diskutováno v kapitole o návrhu změn 5.



Obrázek 4.2: Diagram základních tříd jádra

4.3.2 Komponenty GUI

K připojení komponenty k systému je nezbytné, aby třída komponenty rozšiřovala abstraktní třídu `MiningPiece`. Tím je vynucena implementace některých povinných metod pro každou komponentu (či modul). Při tvorbě dialogů komponent je doporučeno použít prostředků platformy NetBeans pro vytváření dialogů.

Z pohledu této práce jsou důležité především komponenty **SelectData** (výběr dat), **Vimeo** (definice a použití VIMEO funkcí), **Transformations** (aplikace transformací nad daty) a **Insight** (prohlížení dat).

Tato práce by měla především implementovat doposud neřešené problémy z oblasti předzpracování – tedy těchto uvedených komponent. Doposud jsou řešeny pouze problémy čištění dat VIMEO, normalizace a ořezání (clipping). Modul **Insight** pouze zobrazuje data z databáze a neprovádí nad nimi žádnou analýzu (což je pro správné provedení předzpracování nutné).

Částečná řešení problematiky předzpracování tedy aplikace poskytuje, ale nejsou řádně dokončeny. Návrhy úprav plynoucí ze současného stavu jsou v kapitole 5.

Kapitola 5

Návrh rozšíření existujícího systému v oblasti předzpracování dat

Změny popsané v této kapitole vycházejí především z návrhu dalších změn uvedených v [4]. Cílem této práce je předzpracování dat v systému implementovat do finální podoby a vyřešit problémy, které doposud zůstavávají v oblasti předzpracování neřešené či rozpracované.

5.1 Čištění dat, VIMEO

Čištění dat systém řeší již nyní poměrně komplexně podporou VIMEO funkcí. VIMEO funkce tak řeší mimo jiné problematiku **chybějících hodnot**. Systém nyní dovoluje celou n-tici *ignorovat* a nebo *dosadit globální zvolenou konstantu* za tuto hodnotu. Funkce pro nahrazení *průměrnou hodnotou atributu* nebyla prozatím implementována.

V tomto směru tedy navrhuji reakce na VIMEO funkce rozšířit o možnosti popsané v podkapitole 3.2.1. Jde tedy o *dosazení průměrné hodnoty atributu* a *dosadit průměrnou hodnotu atributu pro danou třídu n-tice*. Vhodná by byla i varianta *dosazení nejpravděpodobnější hodnoty*, ale k tomu by bylo třeba mít již implementované vhodné dolovací algoritmy, kterých se může využít k predikci.

Definice funkcí (jak pro VIMEO, tak i pro konstrukce atributů) je v projektu poměrně nedořešená oblast. Ing. Krátký zmiňuje především nevhodné uživatelské rozhraní a při testování současné implementace docházelo k pádům aplikace. Bude třeba kód týkající se definice funkcí kompletně revidovat a reimplementovat do funkčního řešení.

V rámci čištění dat budou rovněž implementovány algoritmy pro **odstranění šumu dat**. Konkrétně se jedná o algoritmy vyhlazování popsané v podkapitole 3.2.2 – především *pomocí rozdělení do košů.*

5.2 Transformace dat

Současný stav implementace je takový, že systém podporuje normalizaci, clipping a konstrukci atributů. Pro normalizaci a ořezání (clipping) používá systém algoritmy implementované systémem *Oracle Data Mining*. V předchozích pracích bylo konstatováno, že **diskretizaci** se nepodařilo vyřešit. Její implementace je nezbytná pro další používání aplikace, neboť některé algoritmy dolování pracují pouze nad diskrétními hodnotami.

V této oblasti tedy existují v podstatě 2 možnosti řešení. Buď budou algoritmy popsané v kapitole 3.3 pro normalizaci a diskretizaci implementovány na straně klientské aplikace, nebo použít implementované algoritmy poskytované systémem ODM. První varianta přináší naprostou kontrolu nad algoritmy a bude možné je v případě nutnosti vhodně modifikovat pro potřeby aplikace. Navíc by bylo možné předzpracování řešit i nad databázovým serverem, který by neobsahoval podporu dolování. Druhá možnost využít stávající implementace (jak tomu je doposud) naopak poskytuje zřejmě rychlejší zpracování operací.

Z tohoto důvodu nelze jednoznačně říci, které řešení je jednoznačně výhodnější. Proto bude vhodné v systému definovat nutné rozhraní pro implementaci těchto operací a ponechat možnost implementace oběma způsoby (použit bude návrhový vzor *„Strategy“*). V první fázi bude řešení vycházet ze stávající koncepce projektu řešit možné úlohy na straně serveru.

5.3 Podpora analýzy dat – komponenta „Insight“

Podstatnou součástí této práce bude implementace nezbytných statistických funkcí dostupných v komponentě **Insight**. Koncepce by mohla částečně vycházet ze zpracování, které je v podobném modulu dostupné například v komerčním řešení programu *SAS Enterprise Miner*.

Návrh funkčnosti modulu

Ve stávající podobě modul pouze zobrazuje tabulkou obsah výstupu předcházející komponenty v grafu, na kterou je připojen. Bude třeba implementovat statistiky nad jedním atributem i nad více atributy (pro dvojici atributů). Pro jeden atribut se jedná o následující statistické ukazatele (popsané v kapitole 3.1).

Statistické charakteristiky atributu jako jsou především **minimum**, **maximum**, **průměrná hodnota**, **medián**, **modus**, **kvantilové hodnoty** – především pro kvartily Q_1 a Q_3 a rozložení decilů, **míry variace** – rozptyl a směrodatná odchylka.

Vizualizaci hodnot atributu pomocí klasických vizualizačních technik: **histogram** (a jeho různé varianty), **krabicový graf**, **kvantilový graf**.

Statistiky nad více atributy budou použity především pro analýzu závislostí příslušných 2 veličin, které atributy reprezentují. Vizualizační nástroje, které za tímto účelem lze pro dvě veličiny použít, jsou především **q-q graf** a **bodový graf**. V případě bodového grafu je vhodné implementovat zobrazování interpolačních polynomů a splajnů do tohoto grafu. Vhodné je také implementovat aproximaci pomocí přímky (např. metodou nejmenších čtverců).

Pro dvě veličiny budou implementovány algoritmy pro zpracování *korelační analýzy* tak, jak byla popsána v kapitole 3.4.4. Za vstup budou akceptovány 2 atributy (oba stejného typu – buď kategorické, nebo numerické) a výsledkem budou příslušné koeficienty (korelační, kovariační, χ^2).

Možnosti implementace návrhu

Podobně jako v předchozích změnách jsou operace poměrně náročné na přenos dat mezi serverem a klientem. Proto bude vhodné některé operace nad souborem dat implementovat jako **uložené procedury** na straně databázového serveru (více v [13]). Tato varianta

bude pro každý řešený podproblém analyzována během implementační fáze. Operace, které nebude možné na straně serveru zpracovávat, budou zpracovány na straně klienta nad přenesenými daty z databáze.

V případě vizualizace dat bude třeba zvážit implementaci vlastní knihovny pro zobrazování potřebných statistických grafů oproti možnosti využít nějaké stávající knihovny (vhodná se jeví implementace *JFreeChart* [8]). Funkčnost knihovny se jeví jako plně dostačující pro práci s grafickými statistickými daty.

5.4 Komponenta redukce dat

Z problematiky redukce dat popsané v kapitole 3.4 v systému chybí komponenta pro výběr podmnožiny dat zvolenou metodou. Pokud by množina dat na vstupu byla příliš rozsáhlá, je vhodné, aby systém obsahoval komponentu, která by řešila výběr reprezentativního vzorku dat vhodnou metodou *vzorkování*.

Uživatel by tak získal kontrolu nad množstvím dat, které vstupují do dolovacího procesu. Implementace této komponenty je na hranici předzpracování dat a samotné úlohy dolování. Je možné, že tato komponenta bude následně i vytvářet rozdělení dat do množin potřebných dolovacími algoritmy (trénovací a testovací množiny dat). Konečná podoba této komponenty bude diskutována s řešiteli dolovacích modulů systému.

5.5 Řešení dalších nedostatků

Problematika udržování připojení k databázi stále není v projektu finálně vyřešena. Během práce dochází k odpojování od serveru z důvodu nečinnosti a nastavování připojovacích údajů je řešeno stále poněkud provizorně (nutnost ručního zásahu do nastavení projektu). Tuto problematiku bude vhodné v rámci práce dovést do stabilního stavu.

Zdroj vstupních dat je nyní pouze existující tabulka na straně databázového serveru. Navrhuji do projektu zavést možnost importu ze standardních datových souborů na straně klienta, jako je například soubor ve formátu CSV, apod.

Kapitola 6

Modifikace stávajících komponent předzpracování dat

Oblasti předzpracování dat nebyla doposud věnována v rámci projektu ucelená pozornost. Proto sice existují komponenty, které na jisté úrovni řeší úlohy z oblasti předzpracování dat, ale implementace není vždy plně funkční nebo není vhodná pro následné užívání (nedobře zpracované GUI, neošetřené reakce na neplatné vstupy, nezachycené výjimky, apod.). Aby mohla být aplikace dobře použitelná, je nezbytné provést úpravy k doladění těchto komponent.

Proto jsem v této kapitole popsal implementační změny, které proběhly, aby všechny komponenty předzpracování byly maximální měrou funkční a použitelné. Jedná se o komponenty výběru dat, VIMEO funkcí a transformací dat.

6.1 Komponenta výběru dat

Komponenta výběru dat (*Select Data*) je převzata již ze zcela první verze systému. O nutných úpravách této komponenty se zmiňují Ing. Krásný i Ing. Mader v svých pracích [4][5]. Vzhledem k tomu, že požadavky na funkčnost této komponenty se vývojem systému značně změnily, přepracoval jsem komponentu kompletně tak, aby reflektovala aktuální požadavky.

Nová implementace odráží poněkud pozměněnou ideu této komponenty. Původně byla komponenta zaměřena čistě na výběr existujících dat na serveru a k správě dat na serveru bylo třeba vždy ještě další aplikace. Nová implementace klade důraz na to, aby uživatel nebyl nucen používat v procesu přípravy dat žádnou další aplikaci ani v případě importu samotných dat.

6.1.1 Proces výběru dat

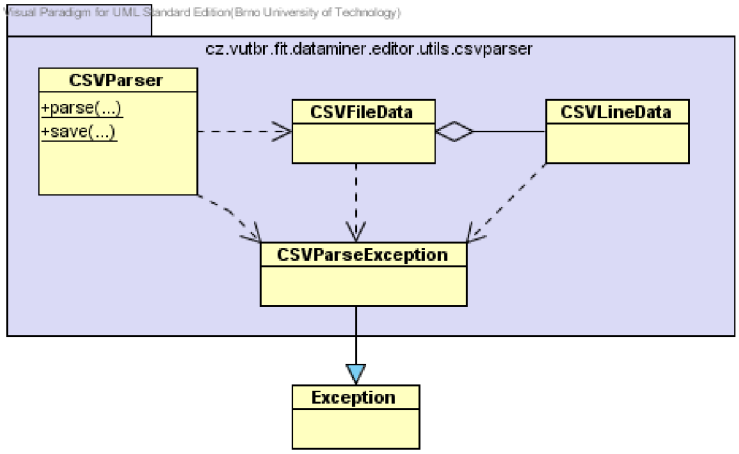
Komponentu jsem reimplementoval tak, aby uživatele provedla jednotlivými kroky výběru dat pro dolování následovně:

1. import dat z lokálního souboru,
2. výběr sloupců pro dolování,
3. nastavení číselníků sloupců s odkazy (reference cizími klíči),
4. určení spojovacích podmínek v případě dolování z více databázových tabulek.

6.1.2 Import CSV dat

Jelikož aplikace pracuje vždy jen s daty, která jsou uložena v tabulkách databáze na straně databázového serveru, je třeba mít možnost tyto data na server importovat přímo pomocí aplikace. Jako vhodný formát, který aplikace bude zpracovávat, jsem zvolil formát CSV (textová data oddělená čárkou či středníkem), více o formátu v [14].

Jelikož jsou po CSV parseru požadovány některé speciální funkce, nebyla pro parsování vybrána žádná dostupná implementace, ale navrhl jsem nový balíček poskytující podporu zpracování CSV souborů `cz.vutbr.fit.dataminer.editor.utils.csvparser`, jehož diagram tříd je na obrázku 6.1.



Obrázek 6.1: Diagram tříd pro CSV parser

Parsování provádí statická metoda `parse()` hlavní třídy parseru `CSVParser`. Formát CSV je poměrně volný. Přestože by se k oddělování jednotlivých sloupců mělo užívat symbolu *čárka*, v praxi se v souborech často používá jako oddělovače i symbolu *středníku*. Proto je proces zpracování souboru parametrizován zadáním tohoto symbolu. Výsledkem zpracování je objekt třídy `CSVFileData` obsahující zpracovaná data z CSV souboru.

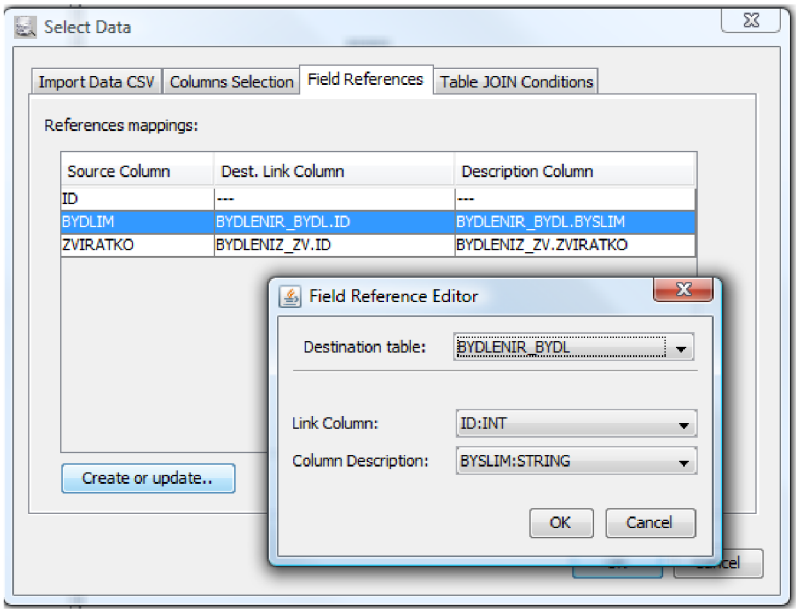
Problém, který je třeba při importu dat řešit, je rozlišení typů sloupců importované tabulky. Proto je v rámci parseru implementována funkce pro rozpoznání nejvhodnějšího typu pro daný sloupec na základě analýzy formátu dat. Vhodný typ je nabídnut k akceptaci uživateli. Uživatel tak může v přehledu typ sloupce změnit, případně zvolit, zdali bude daný sloupec na server importován, či bude importem ignorován.

Výsledkem importu je odpovídající databázová tabulka na straně serveru odpovídající datům v lokálním datovém souboru.

6.1.3 Výběr sloupců tabulek a nastavení referencí

Tuto část jsem upravil z původní verze. Původní verze sice byla funkční, ale její ovládání bylo značně neintuitivní. Sloupce musely být přidávány postupně a nastavování referencí probíhalo přímým zápisem spojovacího výrazu (uživatel musel znát detailně struktury tabulek). Proto úprava znamenala především přepracování GUI a ošetření uživatelských vstupů. Přidání sloupců dolování nyní může probíhat hromadně – i přímo pro celou vybranou databázovou tabulku.

Pro nastavení referencí do číselníků byl implementován jednoduchý průvodce nastavením referencí. Referencím do číselníků se bude více věnovat kapitola transformací dat 6.3.1. Ukázka uživatelského rozhraní komponenty je na obrázku 6.2.



Obrázek 6.2: GUI komponenty *Select Data*

Poslední důležitou součástí této komponenty je možnost definovat spojovací výrazy při dolování z více tabulek. Jestliže jsou totiž vybrány sloupce z různých tabulek, potom se při vytváření datové matice, která představuje výstup této komponenty, provede kartézský součin záznamů v těchto tabulkách. Aby bylo možné ovlivnit spojení těchto tabulek, je možné definovat tzv. *Conditions* na základě rovnosti hodnot ve vybraných sloupcích příslušných tabulek. Zde jsem opravil původní implementaci spojení na úrovni sestavování SQL dotazu, neboť byla chybná a mohla vytvořit neplatný dotaz.

6.2 Komponenta VIMEO funkcí

Komponenta realizující v systému čištění dat v procesu předzpracování dat je komponenta *VIMEO funkcí*. Princip VIMEO funkcí byl charakterizován v kapitole 4.3.1. V původní verzi systému byla komponenta implementována, ale nebyly řešeny některé nezbytné reakce na VIMEO funkce jako především náhrada hodnoty za průměrnou hodnotu nebo náhrada za průměr hodnot patřících pouze do stejné třídy. Navíc tato komponenta zpracovává hodnoty vždy na straně klientské aplikace, což je velmi časově náročné.

6.2.1 Přepracování zadávání funkcí v systému a jejich vyhodnocení

Princip je v tom, že systém umožňuje definovat jednoduché funkce, které se vztahují ke zpracování jednotlivých řádků vstupní datové tabulky. Funkci lze přiřadit množinu parametrů, které lze navázat na konkrétní sloupce, a vyhodnocení funkce pro každý řádek proběhne s hodnotami parametrů odpovídajícím hodnotám v příslušném řádku. Takto definované funkce jsou rozlišeny na dva typy:

- **VIMEO funkce**, kdy funkce vrací jednu hodnotu z množiny VIMEO (Valid, Invalid, Missing, Empty, Outlier);
- **standardní funkce**, kdy návratovým typem je jeden ze standardních datových typů (celé číslo, řetězec, ...).

Stávající koncept a jeho nedostatky

Tělo interní funkce (externí nyní neuvažujeme) je vždy definováno jako *množina podmínek nad jednotlivými argumenty* a pokud je podmínka splněna, je navrácena definovaná návratová hodnota (což může být hodnota, výraz – proběhne jeho vyhodnocení, nebo stav VIMEO v případě VIMEO funkce) – dle definice v [3] jde o tzv. `TUPLE`. Pokud není splněna ani jedna podmínka, vrací funkce implicitní hodnotu.

Celý tento koncept je velmi dobrý a tvoří komplexní způsob přístupu k uživatelsky definovaným funkcím, avšak jeho zpracování obsahuje několik nedostatků.

- Za první závažný nedostatek považuji pouze **částečně zpracovanou kontrolu typů** při definici funkcí a především u jejich navazování na jednotlivé sloupce, nad kterými budou vyhodnocovány. Například při definici výrazů k vyhodnocení lze zadat i kombinaci operace dělení s datovým typem řetězec.

  Propracovaná typová kontrola je hlavně v rámci definice podmínek argumentů a návratových hodnot, kdy systém skutečně nevalidní zadání odmítne.

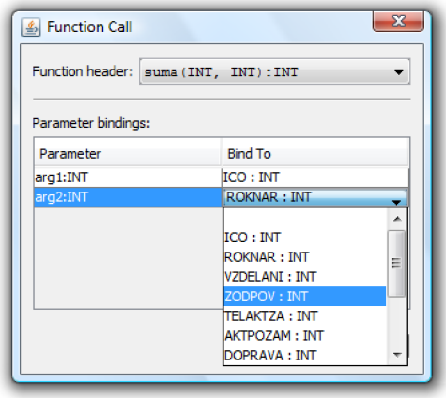
- Další (i když méně závažný) problém je ve **způsobu zadávání funkcí**. Pravděpodobně proto, že systém počítá s možností definice pouze hlavičky tzv. externí funkce, je uživatel nucen i při zadávání interní funkce po zadání hlavičky dialog zavřít a následně znova ten stejný otevřít, aby mohl zadat i tělo interní funkce.

- Hlavní problém stávajícího řešení však spočívá ve **způsobu vyhodnocování zadaných interních funkcí**. Jak u komponenty VIMEO, tak i u komponenty transformací dat se totiž vyhodnocování provádí na straně klienta. V praxi to znamená, že celá vstupní množina dat se nejdříve přenese na stranu klienta, zde je vyhodnocena příslušnou komponentou, a následně po jednotlivých záznamech vkládána zpět do výsledné tabulky na serveru. Tímto vzniká **velká časová náročnost** vyhodnocování těchto funkcí.

Přepracování GUI zadávání funkcí

Na základě této analýzy jsem v této iteraci vývoje systému způsob vyhodnocování funkcí přepracoval tak, aby tyto problémy byly odstraněny. Především se jedná o přepracování formulářů GUI tak, aby nedocházelo ke vzniku nekonzistentních stavů funkcí a jejich vyhodnocování.

Formulář pro přiřazování sloupců k jednotlivým parametrům funkce byl změněn tak, aby umožnil přehledné navázání parametrů vyhodnocení těchto funkcí. Rovněž je kladen důraz na typovou kontrolu tak, aby na numerické parametry bylo možné navázat pouze vstupní sloupce numerických typů. Příklad přiřazovacího dialogu parametrů pro odvození sloupce v rámci transformační komponenty je na obrázku 6.3.



Obrázek 6.3: GUI pro zadávání parametrů funkcí

Přepracování způsobu vyhodnocování funkcí

Stěžejní změnou v rámci práce s funkcemi je však způsob jejich vyhodnocování. Jak již bylo zmíněno, systém dříve umožňoval definici interních i externích funkcí a oba typy byly vyhodnocovány na straně klienta. Avšak časově výhodnější by nesporně bylo funkce vhodně přeložit do dotazů jazyka SQL (příp. PL/SQL) a jejich vyhodnocení zcela ponechat na databázovém serveru, který má data přímo k dispozici (jsou uloženy na jeho straně). Tento překlad jsem v rámci této práce naimplementoval, avšak je možné jej aplikovat pouze na interní funkce, jejichž definice je známá přímo v dokumentu DMSL. Externí funkce samozřejmě takto řešit nelze, neboť používají již přeložené `.class` soubory a volání jejich funkcí. Z tohoto důvodu byl koncept externích funkcí z projektu vypuštěn, a pokud by byl třeba, bylo by jej nutné integrovat do nového řešení. Tím by se ovšem naprosto eliminovala výhodnost řešení na serveru.

Algoritmus převodu interní funkce definované pomocí elementů `TUPLE` v DMSL dokumentu vychází z myšlenky, že funkce je popsatelná podmínkami a následně akcí, která se má provést se záznamy, když podmínka platí – jinak řečeno, pokud hodnoty v řádku tabulky splňují tuto podmínku. Zde je patrno, že pokud je možnost převést podmínky na predikát v klauzuli `WHERE`, a pokud je možnost překladu příslušné akce na dotazy DML typu `UPDATE` a `DELETE`, pak je celá interní funkce přeložitelná na posloupnost SQL dotazů (posloupnost z důvodu, že jedna funkce může obsahovat více `TUPLE`).

Samotnou podmínku lze definovat více způsoby: parametr **je roven** hodnotě, parametr **je v množině** hodnot, parametr **leží v intervalu** (u numerických parametrů). Zjednodušený převod na predikát může vypadat dle schématu algoritmu 6.1.

Algoritmus 6.1

1. Všechny parametry jedné `TUPLE` převést na příslušné jednoduché podmínky takto:
   - pokud je parametr testován na rovnost, pak je převod jako `jmenoParametru = hodnota`
   - pokud je parametr testován náležitost množině, pak je převod jako `jmenoParametru IN (hodnota1, hodnota2, ...)`

- pokud je parametr testován náležitost intervalu, pak převod lze schématicky vyjádřit jako
  `jmenoParametru [>|>=] levaMez` a `jmenoParametru [<|<=] pravaMez`,
  definice obou mezí je volitelná a rozlišuje se uzavřenost mezí intervalů.

2. Predikáty pro všechny parametry spojit operátorem `AND`, neboť musí být splněny zároveň.

3. Pokud je návratovou hodnotou implicitní hodnota funkce, tak může být akce provedena ještě v případě, že nebyla splněna žádná jiná podmínka, což lze symbolicky vyjádřit jako
   `OR NOT( cond1 OR cond2 OR ... )`

4. Nahradit jména parametrů za skutečná jména sloupců získaná z elementů `FieldRef`.

Takto lze funkci složenou z několika `TUPLE` vyjádřit pomocí příslušných predikátů, které se použijí v klauzuli `WHERE` dotazů. Reálně překlad ještě spočívá v optimalizaci tak, že se pro stejné návratové hodnoty vytváří jeden společný predikát, takže musí proběhnout sloučení příslušných predikátů, pro zjednodušení tento krok v algoritmu neuvádím. Vyjádření samotných akcí, které mohou být provedeny, bude popsáno až v konkrétních kapitolách o VIMEO funkcích a transformaci odvození sloupce, kde je uvedeno i srovnání výhodnosti této implementace.

6.2.2 Úpravy VIMEO komponenty

V původní verzi jádra neměla komponenta vyřešenou kompletní funkčnost. Sice měla implementované reakce ignorování záznamu a nahrazení globální konstantou, ale další důležité reakce chyběly. **Nahrazení průměrem hodnot** daného sloupce či **nahrazení průměrem hodnot ze záznamů patřících do stejné třídy** jsem doimplementoval až v rámci této práce.

Nahrazení průměrem celého sloupce je zřejmé, u průměru hodnot ze záznamů patřících do stejné třídy je při zvolení této možnosti ještě nutné vybrat sloupec dolovací matice, které obsahuje rozlišení (identifikace) příslušných tříd.

V rámci přepracovávání komponenty jsem částečně reimplementoval i GUI, neboť v předchozí variantě docházelo k problémům s otevíráním modálního dialogu pro zadávání parametrů volání funkce – formulář výběru parametrů přestal reagovat, neboť zde byl nestandardně použit `PopupMenuListener`. Koncept původního způsobu nastavení VIMEO však zůstal zachován.

6.2.3 Vyhodnocování VIMEO funkcí

Jak jsem již zmínil v předchozích podkapitolách, problémem komponenty VIMEO bylo i řešení vyhodnocení VIMEO funkcí na straně klienta. Vyhodnocení probíhalo tak, že se nejdříve záznamy postupně získávaly ze serveru (pomocí standardního `SELECT` dotazu), po získání byly nad hodnotami vyhodnoceny VIMEO funkce, a pokud záznam neměl být ignorován, tak byl vložen do výsledné dolovací datové matice. Pokud uvažujeme, že se VIMEO funkce budou používat především k čištění dat (například ignorování záznamu s chybějícími hodnotami), potom pro celé vyhodnocení byl z pohledu časové náročnosti nejhorší právě případ, kdy všechny hodnoty byly správné a všechny záznamy opět vloženy do výsledné tabulky.

Právě toto bylo hlavní motivací reimplementace celého systému vyhodnocování těchto funkcí tak, aby byly provedeny na straně databázového serveru. Princip překladu interní VIMEO funkce byl podrobně popsán v kapitole 6.2.1. K popisu zbývá pouze doplnit, jak se transformují konkrétní reakce na VIMEO funkce.

Po získání predikátu pro klauzuli `WHERE` ještě zpracujeme příslušnou reakci pro každý ze stavů VIMEO dle toho, co má být provedeno:

- v případě ignorování záznamu se provede dotaz typu `DELETE` nad řádky tabulky, které jsou vybrány příslušným predikátem;
- v případě nahrazení konstantou či jinou hodnotou se provede dotaz typu `UPDATE` konkrétního sloupce s nahrazující hodnotou.

Provedení takto zkonstruovaných SQL dotazů je značně časově výhodnější. Porovnání časů vyhodnocení jedné a dvou stejných VIMEO funkcí je uvedeno v tabulce 6.1. Je patrné, že řešení úlohy na straně klienta má časovou závislost striktně lineární na počtu zpracovávaných záznamů. U řešení na serveru není složitost přesně určena, neboť plyne z možností optimalizátoru při provádění dotazu, avšak proces se pro zkoumaný vzorek urychlí více než stonásobně. To výrazně zvyšuje praktickou použitelnost aplikace.

Metoda	**Počet záznamů vstupu**				
	500	**1000**	**2000**	**3000**	**4000**
Klient, 1 funkce	8,01 s	16,18 s	31,10 s	45,98 s	64,81 s
Klient, 2 funkce	8,00 s	18,15 s	35,69 s	55,25 s	69,03 s
Server, 1 funkce	0,11 s	0,10 s	0,10 s	0,10 s	0,15 s
Server, 2 funkce	0,11 s	0,12 s	0,13 s	0,15 s	0,12 s

Tabulka 6.1: Čas vyhodnocení VIMEO funkcí lokálně a na serveru

6.3 Komponenta transformací

Rovněž i základ této komponenty *transformací* byl implementován v předchozí verzi systému. Komponenta sdružuje funkce, které se aplikují na celé sloupce (narozdíl od VIMEO, kde se jednalo o zpracování jednotlivých hodnot). V původní verzi komponenty byla implementována funkce normalizace a oříznutí (pomocí podpory ODM) a funkce odvození sloupce podle uživatelsky definované funkce. Aby byla komponenta reálně použitelná chyběla, především implementace diskretizačních metod. Navíc rovněž i zde je odvození sloupce pomocí definované funkce zpracováváno na straně klienta, což opět vytváří neúnosné časové nároky.

Kromě řešení výše uvedených nedostatků byla komponenta přepracována i z pohledu GUI tak, aby lépe zapadala do celkového konceptu. Původní verze především nekomfortně řešila zobrazování transformací, kterými byl daný sloupec odvozen. Hlavní změny ve funkčnosti jsem podrobně rozebral v následujících podkapitolách.

6.3.1 Reference sloupců do číselníků

Při implementaci modulu pro dolování asociačních pravidel [5] se projevila nutnost zavést do systému koncept odkazů hodnot do číselníků. Důvodem je především fakt, že podpora

pro dolování ODM pracuje právě nad odkazy a ne nad řetězcovými hodnotami přímo.

Koncept (zavedený a podrobně popsaný v [5]) spočívá v tom, že sloupce obsahující řetězcové hodnoty (v tomto případě nejčastěji typ `VARCHAR2`) jsou převedeny na sloupce obsahující celočíselnou hodnotu, která je cizím klíčem do jiné tabulky obsahující právě číselník – dvojici identifikátoru a konkrétní původní hodnoty. Proces dolování tedy pracuje s požadovanými celočíselnými hodnotami a až při prezentaci výsledků se odkazy převedly na konkrétní hodnoty.

Data v tomto formátu se ale musela vytvořit externí aplikací a jednotlivé tabulky importovat. Navázání se provedlo v *komponentě výběru dat* pomocí `FieldReferences` a systém s takto zkonstruovanými sloupci pracoval. Pokud bychom však vycházeli z myšlenky, že systém bude podporovat všechny kroky předzpracování tak, aby nemusel být použít žádný další nástroj, musí systém implementovat i *možnost vytvoření těchto číselníků v rámci procesu předzpracování dat.*

Funkce vytvoření číselníků a transformace hodnot jakožto odkazů jsem tedy implementoval následujícími kroky:

1. **vytvoření příslušných tabulek** číselníků ve tvaru *(vid: number, vdesc: varchar2)*,
2. **získání seznamu vyskytujících se hodnot** pro vstupní sloupec s řetězcovými hodnotami,
3. pro každou hodnotu **vytvoření záznamu v číselníku a vytvoření odkazů** na příslušných řádcích výsledné tabulky.

Příklad odpovídající sekvence konkrétních SQL dotazů (pro transformaci se využívá pomocná tabulka s explicitním označení záznamů pomocí hodnot `ROWID`, neboť tabulky v procesu dolování nemají definovaný *primární klíč*):

```
-- 1.
CREATE TABLE dm119_l_bydlim( vid NUMBER, vdesc VARCHAR2(128) )
-- 2.
SELECT bydlim FROM dm119_transformations1_rowids GROUP BY bydlim
-- 3.a
INSERT INTO dm119_l_bydlim VALUES ( 1, 'Praha')
-- 3.b
UPDATE dm119_transformations1 SET bydlim = 1
  WHERE row_id IN (
    SELECT row_id FROM dm119_transformations1_rowids
    WHERE dm119_transformations1_rowids.bydlim = 'Praha'
  )
```

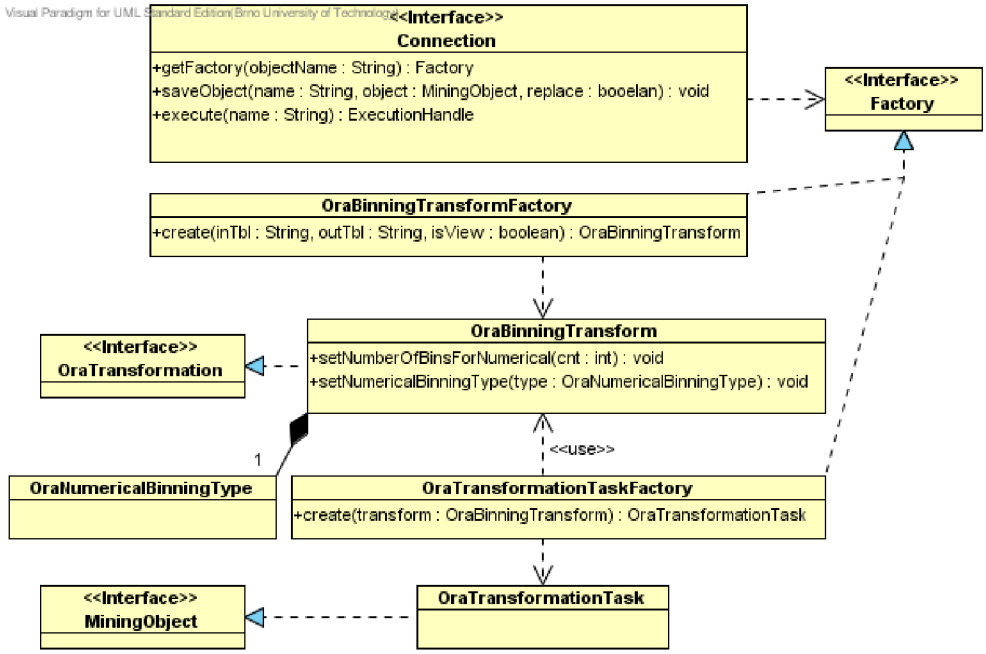
6.3.2 Diskretizace hodnot a koncept realizace transformací

Ze standardních transformačních metod v systému nebyla implementována podpora diskretizace. Systém sice obsahoval třídu poskytující rozhraní diskretizace, ale samotné metody nebyly dořešeny. Jelikož *Oracle Data Mining* poskytuje podporu metod diskretizace, bude použita tato implementace.

Podpora diskretizace v Oracle Data Mining

Oracle Data Mining poskytuje komplexní rozhraní pro podporu diskretizace v prostředí Oracle. Za stěžejní třídu lze v tomto rozhraní považovat `OraBinningTransform`, která uchovává stav připravované úlohy diskretizace. Podporována je jak diskretizace numerických atributů, tak i atributů kategorických. V projektu je využita pouze podpora pro diskretizaci numerických atributů. Diagram tříd použitých pro diskretizaci v projektu je znázorněn na obrázku 6.4.

Parametry diskretizace, které lze nastavit úloze `OraBinningTransform`, jsou počet košů, seznam sloupců, které se nebudou diskretizovat, názvy vstupní a výstupní tabulky pro diskretizaci a především pomocí `OraNumericalBinningType` lze nastavit typ diskretizace, která bude provedena – použity jsou diskretizace o stejné šířce intervalů a stejné hloubce. Rovněž lze definovat i volitelné koše, toho by se dalo využít např. pro implementaci diskretizace 2-3-4 pravidlem.



Obrázek 6.4: Diagram tříd z ODM použitých pro diskretizaci

Implementace diskretizace

Jak bylo uvedeno výše, samotný proces diskretizace je prováděn na úrovni ODM popsaného v kapitole 6.3.2. Avšak diskretizovaný sloupec je datového typu řetězec a obsahuje zápis intervalu, do kterého patřila původní hodnota. Do celkového konceptu systému tento výstup příliš nezapadá, neboť dolování je prováděno pouze nad celými čísly jakožto odkazy do číselníků. Proto se po získání výsledků diskretizace provede ještě převod na číselníky podobně, jako bylo popsáno v kapitole 6.3.1. Teprve až takto převedené hodnoty na číselníky jsou předány jako výsledek diskretizace.

6.3.3 Vyhlazení hodnot sloupce

Do komponenty jsem integroval i novou funkci – vyhlazení sloupce – přestože se nejedná přímo o transformaci, ale spíše o metodu čištění dat. Avšak do konceptu komponenty čištění dat VIMEO vyhlazování logikou nezapadá a vzhledem k tomu, že jde o proces prováděný nad celým sloupcem vstupních dat, je řešen právě touto komponentou.

Z metod pro vyhlazování jsem implementoval metodu plnění (angl. binning). Implementace jsem řešil tak, že se množina hodnot nejříve seřadí vzestupně. Následně je soubor rozdělen na koše o (přibližně) stejném počtu hodnot a hodnoty v každém koši nahrazeny buď průměrem, nebo mediánem příslušného koše. Implementace celé procedury je na úrovní posloupnosti SQL dotazů tak, aby provedení bylo ponecháno na databázovém serveru.

6.3.4 Konstrukce (odvození) nového atributu

Právě z důvodu podpory konstrukce nového atributu bylo původní řešení transformací zpracováváno na klientovi a data musela být (podobně jako v komponentě VIMEO) přenášena zpět na server. Nová implementace spočívá ve vhodné konstrukci SQL dotazů DDL a DML. Odvozený sloupec je nejdříve vytvořen jako prázdný a následně posloupností dotazů typu `UPDATE` naplněn hodnotami. Na výstupu uživatelské funkce může být

- **konstantní hodnota**, která se do dotazu použije přímo,
- nebo **výraz nad stávajícími numerickými sloupci**, kdy se dotaz zkonstruuje i včetně tohoto vhodně upraveného výrazu.

Tímto přístupem jsem dosáhl značného zrychlení výpočtu transformační komponenty; porovnání s původním řešením je uvedeno v tabulce 6.2. U řešení na klientovi je opět lineární časová závislost na počtu záznamů. U řešení na straně serveru jsou časy vyhodnocení přijatelné a jejich růst není tak rychlý jako u lokálního vyhodnocování.

Metoda	**Počet záznamů vstupu**			
	250	**500**	**750**	**1000**
Klient, odvození a normalizace	7,91 s	11,05 s	16,83 s	19,15 s
Server, odvození a normalizace	1,95 s	2,06 s	1,82 s	2,07 s
Server, odvození a normalizace (+ diskretizace)	4,09 s	4,00 s	3,82 s	4,11 s

Tabulka 6.2: Čas vyhodnocení transformací normalizace a odvození sloupce lokálně a na serveru

Kapitola 7

Implementace nových komponent předzpracování dat

Kapitola se zabývá návrhem a především způsobem implementace nových komponent, které dosud v systému chyběly. Jedná se o komponenty analýzy dat a redukce dat.

7.1 Komponenta pro analýzu dat

Podstatnou roli při přípravě dat hraje u analytika samotné poznání dat, se kterými pracuje. V předchozích verzích obsahoval projekt jen minimální možnosti v tomto směru. Jedinou možností bylo v podstatě zobrazit výstup předchozí komponenty v podobě jednoduché tabulky s daty pomocí původní *Insight* komponenty.

Proto byl v kapitole 5.3 zpracován návrh funkčnosti nové komponenty pro analýzu dat, která by poskytovala uživateli i potřebné statistické a další funkce. Komponentu jsem implementoval v souladu s tímto návrhem. Podrobnosti k implementaci jednolivých částí komponenty jsou rozebrány v následujících podkapitolách.

7.1.1 Základní funkčnost komponenty

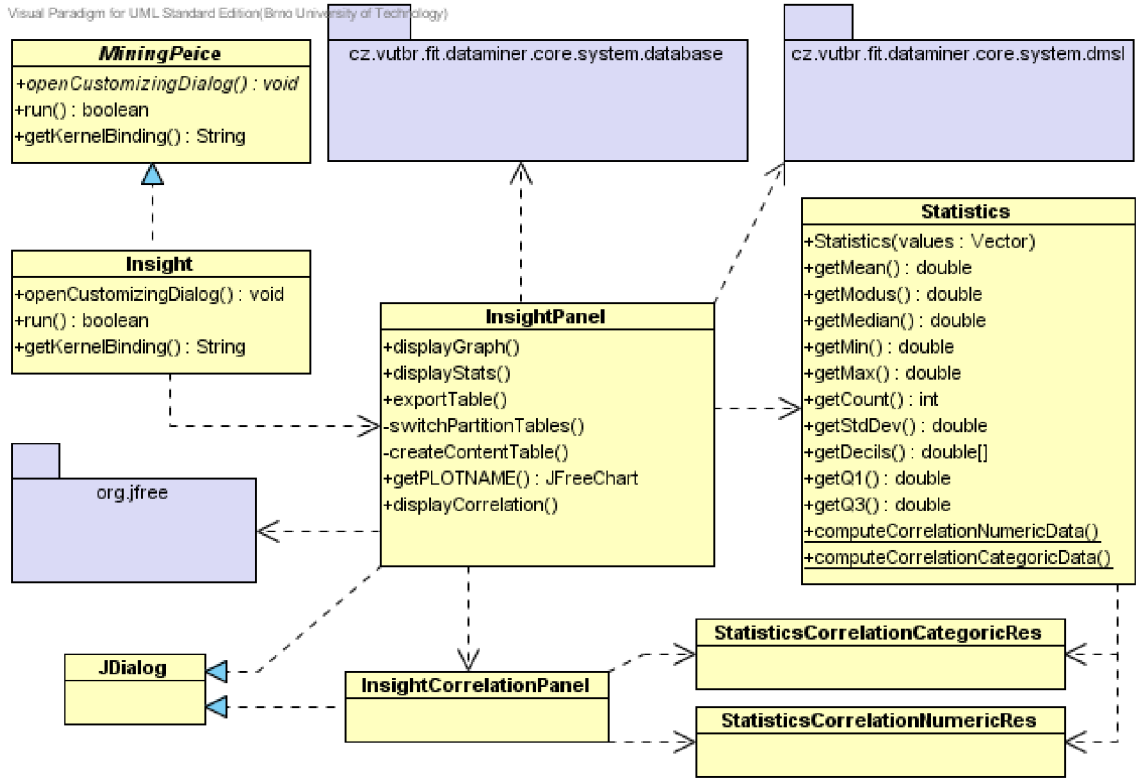
Základní zobrazení výstupních dat připojené komponenty zůstalo zachováno. Komponenta byla navržena tak, aby umožňovala být přímo součástí grafu dolovacího procesu jako samostatný uzel (tak tomu bylo i dříve) a nově byla přidána možnost zobrazení výstupu jiné komponenty pomocí *Insight* panelu i v kontextovém menu funkcí *Explore* příslušného uzlu. Není tedy třeba uzel *Insigh* přidávat do dolovacího stromu, kde místy působil až příliš nepřehledně. Způsob aktivování je popsán v kapitole 7.1.6.

Při implementaci komponenty jsem zvažoval, zdali by měla komponenta data číst vždy on-line z databáze, nebo datový vstup načíst do interních struktur a dále manipulovat pouze s touto reprezentací dat. Za vhodnější jsem zvolil druhý způsob implementace, neboť v průběhu práce s komponentou jsou data zpracovávána mnohokrát (při požadavku o zobrazení statistik, grafů nebo požadavku na export). V případě on-line práce by musela být vždy znovu načtena, což se ukázalo být časově velmi náročné. Komponenta vždy načte data k zobrazení do struktury `DBTableContent`, která je optimalizována přímo k rychlé práci nad jednotlivými sloupci (struktuře se věnuje kapitola 8.1).

Uvedený princip umožňuje navíc snadno přepnout mezi různými strukturami k zobrazení, čehož je s výhodou využito pro implementaci možnosti zobrazení dat i s náhradou

referencí sloupců. To je nezbytné, neboť v datech připravených k dolování se mohou vyskytovat v hlavní tabulce pouze číselné odkazy do číselníků a jejich zobrazení a analýza by neměly smysl. Proto komponenta umožňuje mezi pohledy s využitím referencí a bez využití přepínat.

Diagram tříd se základními metodami, které komponenta využívá, je znázorněn na obrázku 7.1. Komponenta je standardně začleněna pomocí třídy `Insight`, která následně při volání metody `run` vytváří dialog se zobrazením `InsightPanel`.



Obrázek 7.1: Zjednodušený diagram tříd komponenty *Insight*

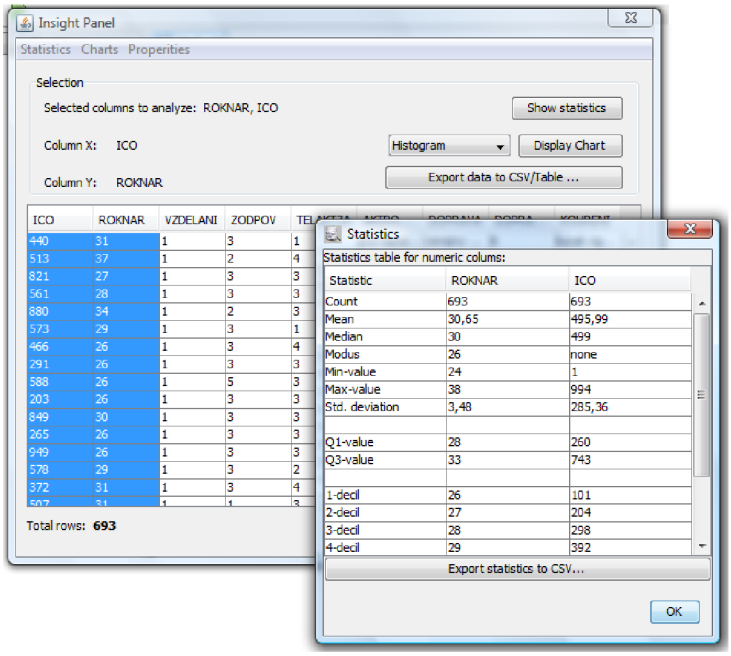
7.1.2 Výpočet statistik

V rámci řešení komponenty jsem implementoval i modul statistických výpočtů – konkrétně jde o třídu `cz.vutbr.fit.dataminer.editor.utils.Statistics`. Třída je optimalizována pro výkon na (i opakované) zjišťování statistických hodnot způsobem, že všechny statistické veličiny jsou předpočítány v rámci konstruktoru objektu třídy `Statistics` a příslušné *get*-metody pouze vrací tyto předpočítané hodnoty. Optimalizace výkonu byla provedena tak, že všechny statistiky, které vyžadují projití celého vzorku dat, jsou počítány v jednom průchodu. Třída poskytuje tyto statistické veličiny:

- **průměr**, **medián** a **modus** – metody `getMean()`, `getMedian()` a `getModus()`
- **standardní odchylka** – metoda `getStdDev()`
- **minimální, maximální hodnota** – metody `getMin()` a `getMax()`

- **kvartily a decily** – metody `getQ1()`, `getQ2()`, `getQ3()` a `getDecils()`

Při implementaci uživatelského rozhraní komponenty musela být řešena problematika výběru sloupců k analýze. Za nejvhodnější jsem zvolil označení příslušných sloupců jednoduše kliknutím přímo v zobrazených datech. Je tak patrné, které sloupce budou statisticky vyhodnoceny, a přístup je intuitivní. Počet vybraných sloupců pro statistickou analýzu není omezen. Příklad výběru a zobrazení statistik je na obrázku 7.2.



Obrázek 7.2: GUI komponenty *Insight* pro analýzu dat

7.1.3 Korelační analýza

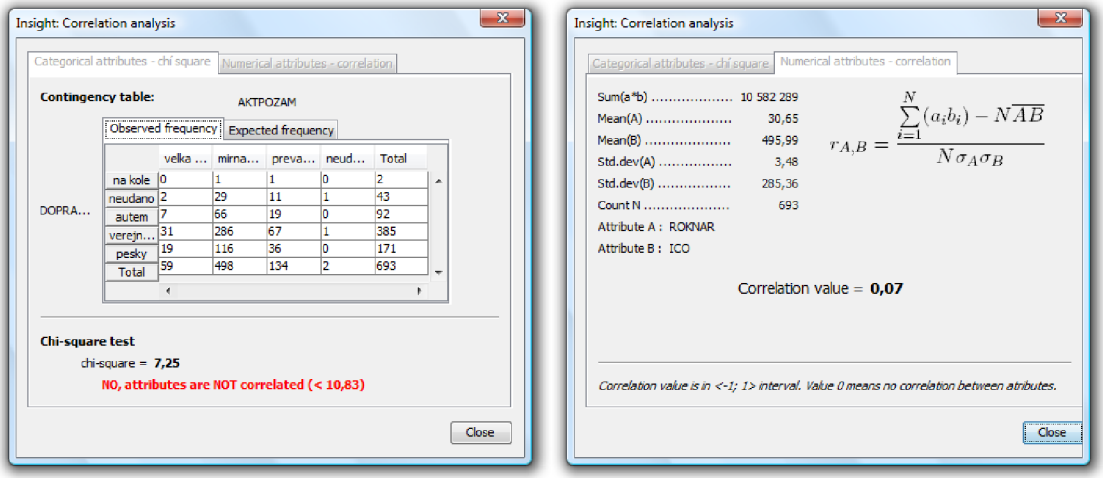
Podstatnou částí analýzy dat je i hledání atributů, které na sobě závisí. K tomuto účelu jsem implementoval v komponentě *Insight* podporu **korelační analýzy**. Analýza je prováděna vždy pro 2 vybrané atributy, kde bylo zavedeno omezení, že oba musí být zároveň kategorické, nebo zároveň numerické. Toto omezení z důvodu použití různých statistických metodik k výpočtu příslušných korelačních hodnot. Matematický základ korelační analýzy byl popsán v kapitole 3.4.4.

Pro **kategorický atribut** je třeba spočítat kontingenční tabulku očekávaných i reálných výskytů hodnot jednotlivých atributů. Jelikož z těchto tabulek je dobře patrno, kde případná závislost vzniká, bylo zobrazení implementováno tak, aby uživatel měl obě tabulky k dispozici k porovnání. Výsledkem celé analýzy je hodnota χ^2, na jejímž základě lze určit závislost příslušných 2 atributů.

V případě **numerického atributu** proběhne výpočet korelačního koeficientu z numerických atributů. Zpracování uživatelského rozhraní korelační analýzy pro oba typy atributů je na obrázku 7.3.

Implementace je řešena tak, jak je znázorněno diagramem tříd na obrázku 7.1. Konkrétní výpočet řeší třída statistických výpočtů `Statistics`. K předání výsledku je použito tříd

`StatisticsCorrelationCategoricRes` a `StatisticsCorrelationNumericRes` dle toho, zda se jedná o výpočet nad numerickými, nebo kategorickými atributy. Výsledek analýzy je zobrazen pomocí dialogového formuláře `InsightCorrelationPanel`.



Obrázek 7.3: GUI panelů korelační analýzy (kategorické a numerické atributy)

7.1.4 Vizualizace pomocí grafů

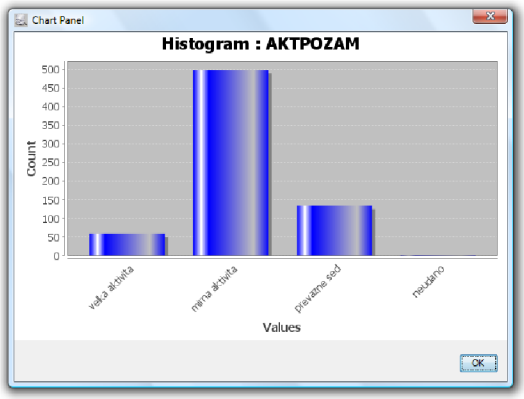
Pro vizualizaci statistických veličin v podobě grafů jsem využil knihovnu pro vykreslování grafů **JFreeChart**, více v [8]. Knihovna je přímo určena ke zpracování statistických dat a požadované grafy přímo podporuje. Knihovna je poskytována pod licencí *GNU Lesser General Public Licence (LGPL)*[1], což účelům projektu plně vyhovuje.

Práce s knihovnou probíhá na principu úpravy dat a vypočtení statistických hodnot ve vlastní režii (s výjimkou hodnot pro *krabicový graf*), zpracovanými daty jsou naplněny interní struktury knihovny a předány k vizualizaci. Implementoval jsem zobrazování těchto grafů:

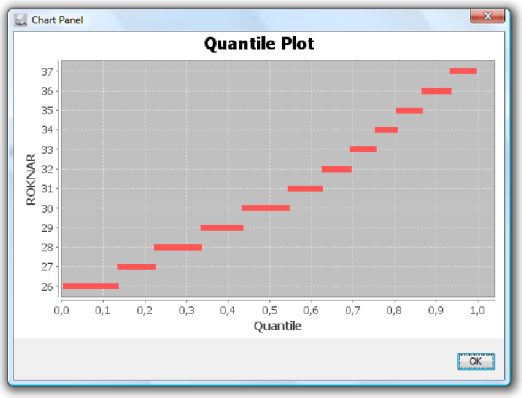
- **Histogram** – vykreslení histogramu nad jedním atributem. Bylo třeba realizovat vykreslení pro kategorické i numerické atributy. U kategorických jsou sloupce histogramu přirozeně jednotlivé kategorické hodnoty. Pro numerické hodnoty je celý interval hodnot atributu rozdělen na zvolený počet intervalů a spočteny do nich spadající hodnoty. Příklad grafu je na obrázku 7.4.
- **Kvantilový graf** – není knihovnou podporován. Lze jej však nahradit grafem bodovým, kde hodnoty atributu jsou závislé na jednotlivých kvantilech. Lze jej spočíst pouze nad numerickým atributem, ukázka je na obrázku 7.5.
- **Krabicový graf** – je určen pro porovnávání charakteristik více atributů, proto lze zvolit obecně neomezený počet atributů pro vykreslení ve společném grafu. Podporovány jsou ale pouze numerické atributy.

[1] http://www.gnu.org/licenses/lgpl.html

- **Bodové grafy (s podporou regrese)** – je jednoduchým typem grafů, který ukazuje závislost jednoho na druhém. Implementována i podpora zobrazení grafu s vykreslením lineární regrese a kvadratické regrese.
- **Q-Q graf** – je rovněž emulován pomocí bodového grafu, kdy příslušné dvojice hodnot pro X a Y se konstruují dle definice v kapitole 3.1.4.



Obrázek 7.4: Příklad histogramu



Obrázek 7.5: Příklad kvantilového grafu

7.1.5 Export dat z dolovacího procesu

Již Ing. Krásný ve své práci [4] upozornil na vhodnost funkce, která by umožňovala data po provedení transformací systémem v jistém bodě **exportovat pro další analýzu jinými nástroji** či možnosti opakovaného použití konkrétního vzorku dat. Tuto funkci jsem integroval do komponenty *Insight*, neboť právě ji lze začlenit na libovolné místo pracující s dolovacími maticemi a implicitně má již data potřebná k exportu načtena. Export jsem řešil jak do souboru ve formátu CSV pomocí třídy `CSVParser`, tak i do nové databázové tabulky na serveru. Komponenta exportuje vždy ta data, která právě zobrazuje – tzn. včetně případné aplikace nahrazování referencí sloupců, apod.

Funkce by mohla být s výhodou využita například tak, že by se data vhodně transformovala a poskytla druhé osobě již v upravené, předzpracované podobě.

7.1.6 Povolení zobrazení *Insight* z kontextového menu

Povolení funkce zobrazení výstupu komponenty *Explore* z kontextového menu bylo implementováno tak, aby mohl být zobrazen v podstatě libovolný dialog určený k prozkoumání výstupních dat. Pro povolení zobrazení funkce je třeba předefinovat metodu `hasExploreDialog()` tak, aby návratová hodnota byla `true` (implicitně je funkce pro komponentu vypnutá), a redefinovat metodu `openExploreDialog()`, aby otevřela panel *Insight*, což lze například takto:

```
public boolean hasExploreDialog()  { return true; }
public boolean openExploreDialog() {
  InsightPanel.openInsightDialog(getKernel(), this.getKernelBinding());
  return true;
}
```

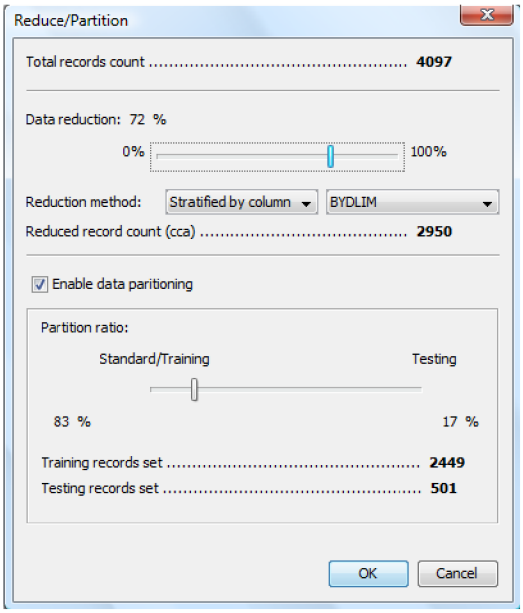
7.2 Komponenta redukce a rozdělení dat

Při reálném provozu je pravděpodobné, že budou zpracovávány velké objemy dat. Systém již obsahuje komponentu *Transformations*, pomocí které lze redukovat či transformovat počet atributů (sloupců) tabulek, avšak doposud nemá mechanizmus, jak redukovat i množství samotných dat ve smyslu řádků tabulky, které do dolovacího procesu vstupují.

7.2.1 Návrh komponenty

V rámci této práce byla tedy nově navržena a implementována komponenta, která umožňuje i manipulaci se samotnými řádky tabulek. Při současném vývoji dolovacích modulů využívajících klasifikaci a predikci se ukázalo, že systém rovněž postrádá komponentu, která by poskytovala možnost množinu vstupních dat rozdělit na více podmnožin. Jelikož problematika redukce dat a zmíněného rozdělení dat podobným způsobem manipulují se vstupní množinou (mění počet řádků tabulky na výstupu), řešil jsem problematiku redukce a rozdělení dat společně v rámci této komponenty.

Při vykonávání je chování komponenty implementováno tak, že se nejdříve provede redukce vstupních dat. Podrobně se implementací mechanizmu redukce zabývá kapitola 7.2.2. Teprve potom je výsledná redukovaná množina dat rozdělena do dvou výstupních tabulek, pokud byla tato volba v nastavení komponenty zvolena, což je rozebráno v kapitole 7.2.4. Návrh grafického rozhraní komponenty je na obrázku 7.6.



Obrázek 7.6: GUI komponenty redukce a dělení dat

7.2.2 Redukce dat

Používané metody vzorkování pro redukci množství dat byly podrobně charakterizovány v kapitole 3.4.2. V systému jsem v závislosti na této analýze implementoval více přístupů k redukci množství dat. Každá metoda je parametrizována pomocí procentuálního vyjádření, jak velký vzorek na výstupu relativně vůči vstupu uživatel požaduje.

První základní implementovanou metodou pro výběr dat je **náhodný výběr vzorků bez návratu**. Výhoda této metody je, že ji lze velice snadno (i na různých databázových systémech) řešit pouze jedním SQL dotazem. V podstatě stačí náhodně seřadit množinu vstupních dat a zduplikovat pouze příslušný počet řádků do výstupní množiny.

Další řešenou metodou je **náhodný výběr vzorků s návratem**, kdy může již dříve vybraný vzorek být opět vybrán. Při řešení je tedy třeba zabezpečit, aby výběr probíhal cyklicky náhodně vždy z celé množiny vstupních dat.

Třetí implementovanou metodou je **stratifikovaný výběr** vzorku. Ten kromě procentuálního poměru výstupních hodnot vyžaduje ještě parametr sloupce, dle kterého jsou vstupní data rozčleněna do konkrétních tříd. Tedy například, pokud bychom měli atribut *Kraj* u tabulky *Zákazníci*, tak příslušné třídy zákazníků mohou být podle náležitosti k jednotlivým krajům. Z tohoto důvodu tedy algoritmus vyžaduje zadat ještě parametr odpovídající sloupci, který data člení do jednotlivých tříd. Řešení následně automaticky určí jednotlivé třídy a v rámci každé třídy aplikuje náhodný výběr bez návratu tak, že počet vybraných hodnot proporcionálně odpovídá počtu záznamů ve třídě vůči celé vstupní množině.

7.2.3 Optimalizační problémy redukce dat

Pro metodu náhodného vzorku s návratem a stratifikovaného vzorku byly porovnány dva přístupy k implementaci (pro metodu bez návratu by uvažování těchto optimalizací nemělo smysl, neboť je řešena triviálně):

- *aplikačně* opakovaným volání SQL dotazu
- nebo pomocí *procedury PL/SQL* na straně serveru.

Oba přístupy v podstatě řeší problém vždy stejným algoritmem, pouze v prvním případě jsou řídicí konstrukce algoritmu na straně klienta a dochází k opakovanému volání SQL dotazů, které zařídí naplnění příslušnými daty, a v druhém případě je algoritmus přepsán v jazyce PL/SQL a dojde pouze k volání uložené procedury s příslušnými parametry, s jejichž využitím se sestaví a provedou příslušné dotazy dynamického SQL přímo na straně databázového serveru.

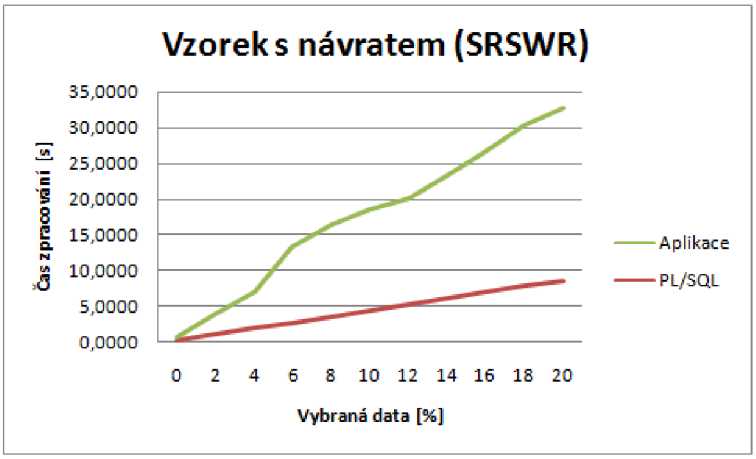
Experimentální ověření časové náročnosti

Experimentální měření proběhla pro výběr z množiny 4000 záznamů postupně pro objemy vzorků od 0 % do 20 % a porovnával se průměrný čas nutný ke zpracování aplikačně s časem zpracování pomocí PL/SQL procedury. Výsledky jsou znázorněny grafy na obrázcích 7.7 a 7.8.

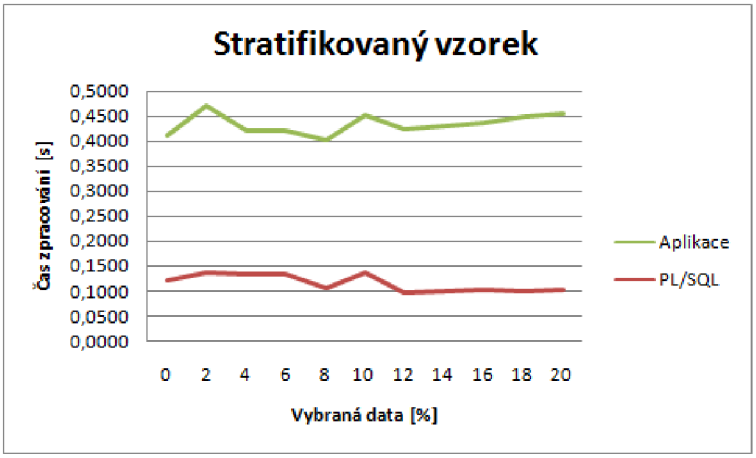
Je patrné, že přestože byl experiment proveden na rychlém připojení, je řešení *výběru metodou SRSWR* na straně serveru pomocí PL/SQL **ve výsledku rychlejší**, přestože obě implementace mají lineární časovou závislost na velikosti výsledného vzorku. Na pomalém připojení by se rozdíl rychlostí ještě více zvětšoval.

U stratifikovaného výběru je časová složitost pro výběr dle konkrétního atributu srovnatelná, což odpovídá způsobu implementace (měnila by se v závislosti vstupní množiny a počtu tříd). Podobně jako u předchozí metody je ale implementace PL/SQL procedurou **výrazně rychlejší**.

Je nezbytné dodat, že pro server implementace pomocí PL/SQL nepředstavuje o mnoho vyšší výpočetní zátěž, neboť až na drobnou režii s řídicími strukturami se provádějí podobné



Obrázek 7.7: Graf časové náročnosti výběru vzorku metodou SRSWR



Obrázek 7.8: Graf časové náročnosti výběru vzorku metodou stratifikovaného vzorku

operace s databází (nikoliv např. složité výpočty, které by server přetěžovaly při větším množství klientů).

7.2.4 Rozdělení dat

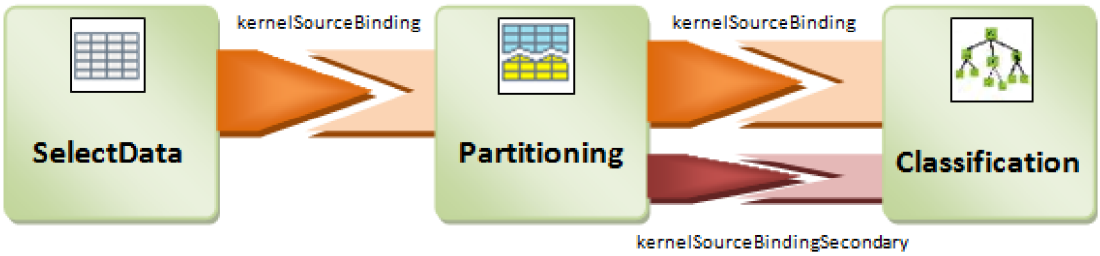
Jelikož souběžně s touto prací probíhá i vývoj dolovacích predikčních a klasifikačních modulů, byl zjištěn nedostatek aplikace, který spočívá v tom, že v rámci posloupnosti kroků dolovacího procesu se předává vždy pouze odkaz na jednu tabulku (tzv. `kernelBinding` předchozí komponenty v grafu dolovacího procesu). Avšak zmíněné dolovací metody vyžadují ke své implementaci 2 vstupní množiny dat – tzv. množinu trénovací a množinu testovací. Toto dělení je nezbytné, neboť na první množině se algoritmus učí a pomocí druhé testuje chybu, které dosahuje při klasifikaci neznámého vzorku dat.

Pro řešení tohoto problému byly váženy 2 možnosti – buď bude každý dolovací modul provádět rozdělení vstupních dat ve své režii, nebo se vytvoří komponenta, která by dělení prováděla. Za výhodu prvního způsobu implementace se nabízel pouze fakt, že dolovací

modul by mohl plně řídit proces rozdělení dle jeho potřeb. Avšak nevýhodami je několik různých implementací v podstatě stejného problému, horší udržovatelnost a v neposlední řadě i těžká porovnatelnost výsledků, pokud by dolovací algoritmy pracovaly nad množinami, které vznikly různým rozdělením dat.

Proto byl zvolen přístup společné implementace rozdělení dat v systému v rámci komponenty pro redukci množiny dat. Funkčnost je navázána na komponentu redukce tak, že po provedení redukce se provede ještě volitelně i operace **rozdělení dat do 2 obsahově disjunktních množin**. Podmínka disjunkce obou množin je *nutná* pro algoritmy pracující s trénovací a testovací množinou.

Rovněž jsem se zabýval řešením problému druhého vstupu těchto dolovacích modulů a otázkou zpětné kompatibility s již implementovanými kódy. Proto jsem navrhl následující řešení. Zachovat i nadále princip, že každá komponenta předává svým následníkům pouze jeden výstup. Avšak v rámci tohoto výstupu vytvořit v podstatě **2 kanály** – *primární* (plně zpětně kompatibilní s původní implementací) a *volitelný sekundární*. Komponenty, které budou po jádře požadovat původní implementací vstup, dostanou vždy data primárního výstupního kanálu. Zároveň komponenty mající pouze jeden výstup budou původním způsobem výstup produkovat a půjde v podstatě o primární výstupní kanál. Naopak komponenty používající 2 vstupní množiny mají možnost získat i druhou množinu. Celý princip je znázorněn ilustrací na obrázku 7.9.



Obrázek 7.9: Diagram principu návaznosti vstupů a výstupů komponent

Z této implementace plyne, že vytvořený sekundární kanál je viditelný pouze pro následníka komponenty, která sekundární tok vytvořila (v našem případě komponenty *Redukce a rozdělení dat*). Pokud následující komponenta tento tok nezpracuje, tak je zahozen a dále se již nepředává. Způsob předání dat a integrace do DMSL jsou popsány v podkapitolách 7.2.5 a 7.2.6.

7.2.5 Modifikace třídy `MiningPiece`

Komponenty mohou převzít odkaz na vstupní tabulku (výstup předchozí komponenty) více způsoby. Prvním způsobem je za pomoci DMSL získat z komponentě náležejícího elementu `DataMiningMatrix` příslušný odkaz na tabulku zdrojových dat uchovaných v synovském elementu `UseMatrix`.

Abstraktní třída `MiningPiece`, kterou musí všechny komponenty povinně rozšiřovat poskytuje ale ještě druhý mechanizmus, kdy instanci třídy `MiningPiece` je z jádra systému předán odkaz na výstup předka. Tato vlastnost je potom získatelná metodou instance `getSourceKernelBinding()`.

Implementace dvou výstupů komponenty musí být tedy taková, že odkaz na sekundární

výstup bude získatelný oběma způsoby. Integrací do DMSL se podrobně zabývá následující podkapitola 7.2.6. Pro získání odkazu na zdroj dat přímo pomocí metody byla třída `MiningPiece` upravena následujícím způsobem.

- **Metodě** `getSourceKernelBinding()` – byla pozměněna pouze sémantika. Nyní tato metoda vrací odkaz na primární výstup předchozí komponenty. Tím zůstává i kompatibilita s předchozími implementacemi zachována.
- **Metoda** `getSourceKernelBindingSecondary()` – nová metoda, která vrací v každé instanci textový odkaz na sekundární výstup předchozí komponenty. Pokud výstup nebyl definován, odkazuje `null`. Komponenty používající tento výstup by tedy *vždy měly kontrolovat, zda byl výstup inicializován.*

7.2.6 Úpravy v DMSL

Při implementaci této komponenty bylo třeba ještě vhodným způsobem navrhnout uložení parametrů redukce dat do DMSL. Zde je třeba říci, že DMSL popisuje spíše metadata dat v dolovacím procesu (tabulky, sloupce a jejich vlastnosti). Nezabývá se ale výskyty těchto dat. Jedinou možností, jak ovlivnit samotné hodnoty, je odkazování definovaných funkcí v elementu `FunctionPool`. Ty jsou určeny přímo pro práci s hodnotami. Avšak tento způsob se nejevil příliš vhodný, neboť zde jsem řešil spíše vlastnost náležející odkazu mezi předchozí dolovací tabulkou a tabulkou následující (konkrétně aplikováno: *z tabulky A zkopírovat 20 % hodnot do tabulky B*) při redukci dat. Podobně i rozdělení výstupní tabulky na 2 kanály při rozdělení dat.

Pro **redukci dat** byl z těchto důvodů rozšířen element `UseMatrix`, který obsahuje odkaz na datovou matici předka v dolovacím procesu, o nové atributy:

- `dataUseRatio` – atribut nabývající číselných hodnot z intervalu $\langle 0; 100 \rangle$, kde hodnota vyjadřuje procentuálně poměr dat, které se mají z původní matice zkopírovat do nové, výstupní. Atribut není povinný. Pokud není zadán, nedochází k redukci dat.
- `dataUseMethod` – atribut nabývající hodnot definovaných výčtem pomocí nové entity `%REDUCTION-METHOD`. Jedná se o definici metody, kterou bude vybrán redukovaný datový vzorek (SRSWOR, stratifikovaný vzorek, ...). Pokud atribut není zadán, zvolí komponenta svůj výchozí typ redukce dat.
- `dataUseStratifiedColumn` – pokud je hodnota atributu `dataUseMethod` rovna hodnotě `STRATIFIED` (stratifikovaný vzorek), pak se očekává, že tento atribut obsahuje název sloupce, dle kterého mají být data klasifikována do tříd při stratifikovaném výběru vzorku. Atribut je nepovinný.

Po aplikaci výše uvedených navržených změn DMSL se změní definice v DTD elementu `UseMatrix` následujícím způsobem:

```
<!ENTITY % REDUCTION-METHOD "SRSWOR | SRSWR | STRATIFIED | default" >

<!ELEMENT UseMatrix (UseField*) >
<!ATTLIST UseMatrix    matrixRef           CDATA              #REQUIRED
                       matrixTreatmentRef  CDATA              #IMPLIED
```

```
                    dataUseRatio    (%INT-NUMBER;)          #IMPLIED
                    dataUseMethod   (%REDUCTION-METHOD;)    #IMPLIED
                    dataUseStratifiedColumn  CDATA          #IMPLIED >
```

Rozdělení dat se týká přímo elementu `DataMiningMatrix`, který odpovídá výstupní dolovací matici. Pro parametrizování rozdělení výstupu do 2 kanálů byl upraven přidáním atributů:

- `partitionName` – atribut definující jméno výstupního sekundárního kanálu. Primární výstup je nadále pojmenován atributem `name`. Atribut je volitelný a nemusí být uveden, pokud sekundární výstup není definován.
- `partitionRatio` – atribut nabývající číselných hodnot z intervalu $\langle 0; 100 \rangle$, kde hodnota vyjadřuje procentuálně poměr rozdělení dat do sekundárního toku. Při neuvedení tohoto atributu není sekundární výstup inicializován.

Výsledná definice DTD elementu `DataMiningMatrix` po aplikaci změn je následující:

```
<!ELEMENT DataMiningMatrix ((UseMatrix | (Query, UseMatrix*)?),
              DataMiningField*, MatrixTreatment*, Annotation?) >
<!ATTLIST DataMiningMatrix    name            CDATA           #REQUIRED
                              partitionName   CDATA           #IMPLIED
                              partitionRatio  (%INT-NUMBER;)  #IMPLIED >
```

Konkrétní dokument ve formátu DMSL dle nové definice může vypadat takto:

```
<DataMiningModel name="DataMiningModel1" functionPoolRef="FunctionPool1">
  <DataMiningMatrix name = "DMt_PartitionReduceData1"
                    partitionName="DMt_PartitionReduceData1A"
                    partitionRatio="20" >
    <UseMatrix matrixRef="DMt_SelectData1"
               dataUseRatio="70" dataUseMethod="STRATIFIED"
               dataUseStratifiedColumn="VZDELANI"  />
    ...
  </DataMiningMatrix>
</DataMiningModel>
```

Kapitola 8

Návrh a implementace změn v jádře systému

V této kapitole jsou popsány změny v kódu jádra, které jsou obecné povahy a nevztahují se ke konkrétní komponentě nebo fázi předzpracování. Jedná se především o řízení spojení s databází, provádění SQL dotazů, implementace vícevláknového běhu výpočtu, apod.

8.1 Abstrakce přístupu k databázi z jádra

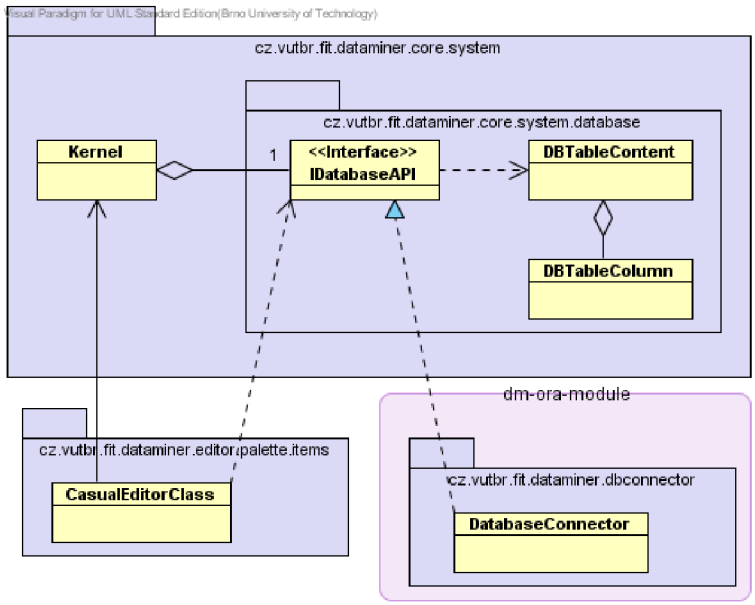
Pokud v předchozích verzích docházelo v rámci nějaké komponenty či samotného jádra k nějaké interakci s databází (načítání hodnot z databáze, její modifikace, apod.), byl kód řešící komunikaci s databází vždy součástí příslušné vykonávající metody. Tím se vytvářela velká a nepříliš vhodná závislost daného úseku kódu (a tím i celé komponenty a jádra) na konkrétním databázovém serveru. A to i přesto, že používané rozhraní JDBC pro připojení sice sjednocuje API nad různými ovladači databází, ale již nezaručuje plnou kompatibilitu konkrétních dotazů jazyka SQL. Syntaxe jazyka se totiž u různých databází (v tomto uvažovaném případě systému Oracle a např. systému MySQL) mírně odlišuje (a to jak DDL, tak i DML).

Rovněž stávající řešení v podstatě vynucuje reimplementaci různých částí jádra a komponent při zásahu do modelu persistence dat, čímž se značným způsobem zhoršuje udržovatelnost celé aplikace. Pokud bychom chtěli mít možnost používat systém s různými implementacemi algoritmů provádějící manipulaci s daty v databázi (například z důvodu různé efektivity), bylo by třeba nahrazovat funkce přímo v kódu a rekompilovat jádro či komponenty jádra.

Je nezbytné dodat, že v tomto případě se jedná pouze o komponenty předzpracování dat náležejících těsným způsobem k jádru systému. Nejedná se o závislost samostatných připojitelných dolovacích modulů, kde závislost na databázovém systému je naopak výhodná a modul lze snadno zaměnit za jiný v případě změny databázového systému. To by byl u komponent v jádře systému zásadní problém.

Z uvedených důvodu plyne, že praktičtější řešení je řešení s co největší možnou **nezávislostí komponent a především jádra systému** samotného na jakémkoliv databázovém systému. Jelikož původní implementace s podobným řešením nepočítaly, bylo třeba vytvořit v systému novou vrstvu, která zajišťuje komunikaci s databází. Rovněž ale je třeba, aby třídy jádra nebyly na této vrstvě přímo závislé, neboť jádro je v systému modul, který má závislosti pouze na standardní knihovny, ne na ostatní moduly systému.

Z tohoto důvodu bylo navrženo a implementováno řešení vyobrazené na obrázku 8.1 s použitím rozhraní `IDatabaseAPI`, které je součástí nového balíčku jádra pro manipulaci s databází `cz.fit.vutbr.dataminer.core.system.database` , abstrahujícího operace s databází. Rozhraní `IDatabaseAPI` obsahuje pouze hlavičky metod, které musí být implementovány. Očekává se, že po doimplementování všech nutných metod volaných z jádra se stane toto rozhraní neměnným. Pro abstrakci databázových tabulek byly vytvořeny třídy pro jejich reprezentaci na úrovni aplikace – třída `DBTableContent` pro reprezentaci především metadat databázových tabulek a `DBTableColumn` volitelně používaných k předávání obsahu tabulek. Jádro a příslušné komponenty jsou tedy závislé pouze na tomto rozhraní `IDatabaseAPI` a ne na konkrétní implementaci.



Obrázek 8.1: Diagram tříd pro abstrakci databáze

Aby bylo řešení funkční, je ještě třeba realizovat implementaci tohoto rozhraní třídou. Proto byl vytvořen balíček `cz.fit.vutbr.dataminer.dbconnector` s třídou implementující zmíněné databázové rozhraní `DatabaseConnector`. Posledním nutným krokem je vytvoření instance této třídy a její dodání do jádra, které ji bude poskytovat závislým komponentám. Pro toto je vhodný moment vytváření samotného jádra, kdy jsou nastavovány připojení k databázi v jádře při otevírání nového projektu. Tato třída (a balíček) je jediný, který má závislost na balíčku s připojením k databázi, což ale už není problémem.

Uvedené řešení tak poskytuje kompletní abstrakci manipulace s databází v celém systému, nezavádí nepříjemné závislosti do jádra a soustředí implementaci databázové vrstvy do jednoho místa, čímž značně zvyšuje udržovatelnost systému. Konkrétní implementaci lze následně jednoduše zaměnit výměnou implementace příslušného balíčku.

8.2 Řešení vícevláknového běhu výpočtu a obsluhy GUI

Závažným nedostatkem aplikace bylo jednovláknové zpracovávání jakékoliv části dolovacího procesu. Prakticky to znamenalo, že GUI aplikace i výpočet (např. asociační analýza) probíhaly v jednom vlákně. Výsledkem bylo, že po dobu provádění výpočtu aplikace přestala

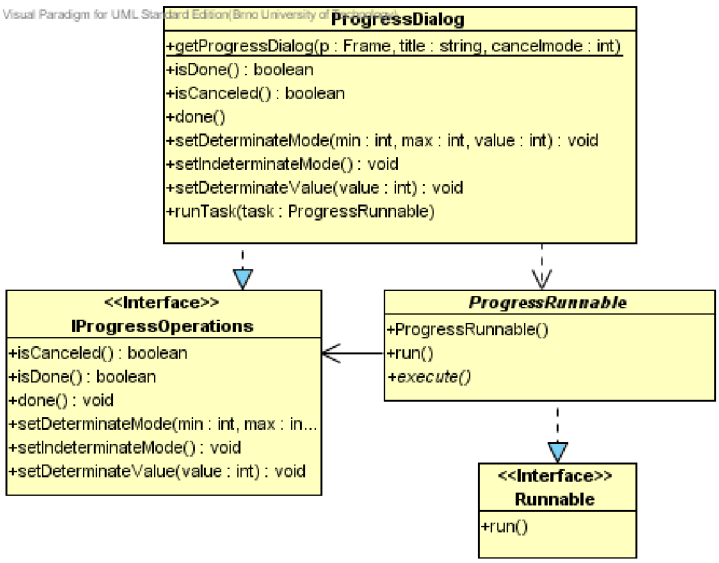
zpracovávat události uživatelského rozhraní až do ukončení výpočtu. Nebyla zde tedy ani možnost zobrazit stav výpočtu, ani výpočet přerušit.

Proto bylo nezbytné navrhnout řešení, které by umožňovalo výpočet a zpracovávání událostí aplikace rozdělit do různých vláken. Standardně by se situace řešila zobrazením modálního dialogu o stavu úlohy v rámci vlákna obsluhujícího GUI a vlastní výpočet by byl spuštěn v samostatném vlákně.

Toto představuje z pohledu současného stavu aplikace zásadní problém, neboť již implementované části se separací do samostatných vláken nepočítají a nejsou k tomu vhodně rozčleněny do tříd implementujících `Runnable` rozhraní podporující spuštění samostatného vlákna. Bylo by tedy třeba významnou část implementovaných částí systému upravit tak, aby třídy s výpočty toto rozhraní implementovaly.

Výsledné řešení, které jsem navrhl a implementoval, není proto řešeno takto standardním způsobem, které platforma Java podporuje, ale pomocí třídy dialogu `ProgressDialog` a pomocné abstraktní třídy `ProgressRunnable` v balíčku `cz.vutbr.fit.dataminer.api`. Rozhraní `IProgressOperations` slouží k zajištění definice metod, které jsou potřebné k manipulaci se stavem průběhu vykonávání a je v balíčku jádra, aby mohly i operace jádra reagovat na stavový dialog. Toto rozhraní tedy musí implementovat třída `ProgressDialog`, která stav úlohy zapouzdřuje.

Princip implementace spočívá v tom, že třída `ProgressDialog` řeší zobrazení příslušného dialogu, zobrazování stavových informaci uživateli a reagování na akci uživatele, v tomto případě reakci na stisk tlačítka přerušujícího úlohu. Abstraktní třída `ProgressRunnable` představuje právě třídu, která implementuje `Runnable` rozhraní. Uživatel musí implementovat jedinou její abstraktní metodu `execute()`, kde bude kód vykonávající výpočet. Implementace sama zajistí správnou práci s vlákny a jejich synchronizaci. Diagram tříd je naznačen na obrázku 8.2.



Obrázek 8.2: Diagram tříd pro `ProgressDialog`

Tento způsob byl zvolen, neboť lze poměrně jednoduše začlenit do již existujícího kódu a podporuje všechny nutné funkce. Zde jde především o možnost stornování úlohy a zobrazení stavu (postupu) úlohy. Stav úlohy lze zobrazit buď v nedeterministickém režimu, kdy

uživateli není zobrazován procentuální stav, nebo deterministickém, kdy je třeba z vlákna výpočtu aktualizovat zobrazovaný údaj pomocí metody `setDeterminateValue()`. Pro stornování bylo definováno více režimů:

- zobrazení stavu **bez podpory stornování**,
- **vynucené okamžité stornování** při stisku tlačítka (vlákno s výpočtem je přerušeno okamžitě v jeho vykonávání),
- **nevynucené stornování úlohy**, kdy je při stisku událost zaznamenána, ale ukončení musí být realizováno přímo z vlákna výpočtu jako reakce na testování stavu dialogu metodou `isCanceled()`. Tento postup je doporučený, neboť umožňuje před ukončením vlákna uvést systém do konzistentního stavu a teprve následně úlohu ukončit.

Příklad kódu pro otevření dialogu se zobrazením stavu může vypadat následovně (více k významu metod lze nalézt v API dokumentaci na přiloženém CD):

```
// Creates new progress dialog
final ProgressDialog dlg = ProgressDialog.getProgressDialog(null,
    "Process Desc", ProgressDialog.CANCEL_MODE_NONE);
// Defines task to be executed
ProgressRunnable dlgTask = new ProgressRunnable(){
  public void execute() {
     callTaskFunction(parameter);
     ...
  }
};
// Executes specified task with progress message
dlg.runTask(dlgTask);
```

8.3 Odlišení databázových tabulek pro více projektů

V projektu se od první verze předpokládalo, že k databázovému serveru bude pod jedním uživatelem zpracováván pouze jeden projekt. Z reimplementace grafického rozhraní Ing. Galétem se však možnosti systému rozšířily a je možnost pracovat v jedné aplikaci i s více projekty zároveň. Rovněž je zde možnost, že jeden uživatel bude používat aplikace ve více instancích a v těchto okamžicích by při současném přístupu docházelo k přepisování jednotlivých přechodných databázových tabulek vzniklých pro komponenty v rámci dolovacího procesu.

Proto je nezbytné vyřešit (nebo alespoň zmenšit pravděpodobnost) kolizí při přístupech projektu na server. Unikátnost projektu nemusí být zajištěna striktně, neboť přechodné tabulky jsou používány pouze při aktivní práci a po opakovaném spuštění aplikace jsou znovu vytvořeny, ale jde o zmenšení pravděpodobnosti kolize při práci s více projekty pod jedním účtem ve stejném čase.

Proto jsem jako vyhovující implementoval způsob, kdy je projektu při vytvoření přiřazen náhodný tříčíslicový identifikátor projektu. Při implementaci bylo zvažováno, zdali identifikátor vygenerovat při každém otevření projektu, nebo pouze při vytvoření a následně by jeden projekt měl identifikaci vždy stejnou. Byl zvolen způsob, kdy identifikace projektu je po dobu jeho existence stejná, neboť pokud by uživatel pracoval sice i na jednom projektu, ale vícekrát, vytvářely by se stále nové tabulky s různými označeními. Takto se pro

jeden projekt používají tabulky se stejným názvem a jejich počet se tak rapidně nezvyšuje (přestože byla implementována i funkce, kdy se při řádném uzavření projektu tabulky automaticky odstraní).

Za tímto účelem musel být dokument DMSL rozšířen o možnost uložení identifikace projektu. Konkrétně se informace ukládá do hlavičky `Header`, která byla rozšířena o synovský element `ProjectId` obsahující identifikátor projektu, neboť právě element hlavičky má obsahovat informace o dokumentu. Nová definice elementu `Header` je následující:

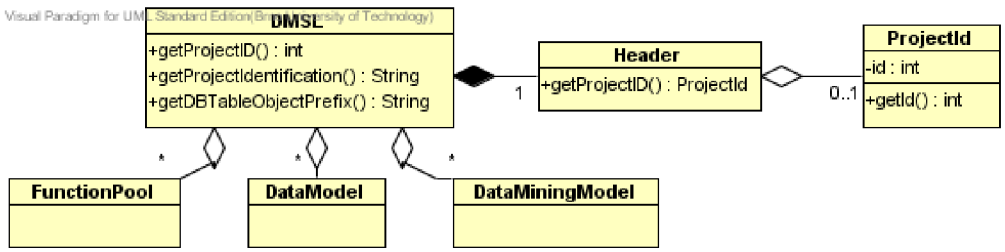
```
<!ELEMENT Header ((DMSLProducer | Annotation)+,ProjectId?) >
<!ELEMENT ProjectId   %INT-NUMBER; >
```

Jelikož doposud nebyl nepovinný element `Header` vůbec aplikací používán, muselo být jádro rozšířeno o zpracovávání elementu hlavičky.

Takto definovaná identifikace projektu se využívá pro generování jmen objektů v databázi, především databázových tabulek, a *pro navazující projekty se doporučuje tuto identifikaci používat.* Přístup ke získání identifikace je buď standardním způsobem ve struktuře DMSL, nebo (jelikož se jedná o vlastnost se silnou vazbou na konkrétní projekt) byl definován zkrácený přístup k vlastnosti přímo pomocí objektu třídy `DMSL`, konkrétně pomocí následujících metod:

- `getProjectID()` – získá číslo identifikace projektu;
- `getDBTableObjectPrefix()` – získá kompletní prefix databázových objektů;
- `setProjectID(int projectID)` – nastaví novou projektovou identifikaci. (Dříve vytvořeným komponentám však zůstává identifikace původní, neboť již vytvořená vazba `kernelBinding` na název tabulky je dle dřívější koncepce už neměnná!)

Odpovídajícím způsobem jsem doplnil DMSL datovou strukturu jádra tak, aby reflektovala aktuální stav definice DMSL dokumentu. Diagram tříd struktury DMSL obsahující identifikaci projektu je na obrázku 8.3.



Obrázek 8.3: Diagram tříd DMSL s dodefinovanou identifikací projektu

8.4 Úprava procesu připojování k databázi

Doposud v žádné práci nebyla dořešena problematika připojování k databázi, přestože se jedná o důležitou součást systému, bez jejíž dokončení nebude systém řádně fungovat. Já jsem se zaměřil v práci především na problém (popsaný i v [4]), který vychází z toho, že dříve bylo připojení zadáno pevně v kódu. I přes následná vylepšení nebylo například možné

bez manipulace s konfiguračními soubory použít heslo odlišné od jména. Proces připojování je navíc komplikován existencí 2 spojení – jedno klasické pro dotazy a druhé pro použití ODM.

Celou problematiku připojování k databázi jsem tedy uvedl do funkčního řešení opravou všech vzniklých chyb. Jednalo se především o reimplementaci procesu připojení v rámci třídy `DMSLDataObject`. Rovněž jsem přidal možnost obnovení připojení (tzv. **Database Reconnect**) pro případ, kdy dojde k odpojení ze strany serveru – dosud nebyla jiná možnost, než zavřít celou aplikaci.

8.5 Konvence maximální délky názvu komponenty jádra

Při ověřování funkčnosti aplikace jsem zjistil problém, že docházelo k přesažení maximální možné délky názvu tabulky na databázovém serveru určené databázovým serverem Oracle na *30 znaků*. To způsobilo nefunkčnost v podstatě celé komponenty. Navíc například modul dolování asociačních pravidel používá jako prefix název předchozí komponenty, ze které čerpá data. Vzhledem k přidání prefixu databázovým tabulkám uvedeným v kapitole 8.3 se tento modul stal nefunkčním, neboť jeho výsledné tabulky maximální délku názvu přesahovaly.

Jelikož se název tabulky vytvářené komponentou skládá z prefixu, jejího názvu a identifikace instance, zavedl jsem konvenci, že maximální možná délka názvu komponenty jádra vytvářející tabulky je **maximálně 6 znaků**. Společně s prefixem projektu o délce 6 znaků a číselným identifikátorem instance komponenty si tedy nyní **jádro rezervuje prefix názvu tabulky o délce 15 znaků**.

Proto musely být upraveny názvy komponent jádra na následující zkratky:

- `SelectData` na `Select`,
- `Transformations` na `Transf`,
- `Vimeo` na `VimeoF`,
- `ReduceData` na `Reduce`.

Kapitola 9

Návrh dalších rozšíření projektu

Širší podpora importovatelných typů souborů

V systému jsem implementoval import často používaného typu souboru CSV. Do budoucna by bylo vhodné rozšířit import i na jiné formáty dat – například XLS. Dále by bylo možné definovat i XML formát pro výměnu dat mezi více instancemi této aplikace a implementovat jeho export a import.

Doimplementování dolovacích modulů

Dořešení všech potřebných dolovacích modulů je nezbytné pro reálnou použitelnost systému. Paralelně s touto prací jsou vyvíjeny některé klasifikační a predikční dolovací moduly. Mezi moduly doposud neřešené se řadí především klasifikátor pomocí neuronové sítě. Dále by mohly být pro školní účely analyzovány a implementovány některé méně typické klasifikátory založené na genetických algoritmech či fuzzy množinách.

Podpora jiných databázových prostředí

Pokud by měla být aplikace použitelná i nad jinými databázovými servery – např. MySQL, bylo by třeba vhodně upravit i správu připojení, která je nyní řešena výhradně pro systém Oracle. Rovněž by bylo třeba ekvivalentním způsobem reimplementovat i databázovou vrstvu, jak byla v této práci zavedena, všechny její algoritmy a následně i dolovací moduly.

Komponenta sloučení výsledků dolování a jejich porovnání

Po implementaci jednotlivých dolovacích modulů by bylo vhodné navrhnout novou komponentu, která by umožnila sumarizovat a porovnat výsledky připojených dolovacích modulů. Odpovídajícím způsobem by muselo být upravené jádro, které prozatím nepodporuje připojení více než jednoho vstupu komponenty.

Důkladné odladění systému při reálném používání

Přestože je při implementaci prováděno testování, bude třeba systém vyladit i při reálném používání, kdy se mohou projevit skryté neodladěné chyby.

Kapitola 10

Závěr

S rostoucím objemem uchovávaných dat se zvyšuje potřeba jejich analýzy procesem získávání znalostí z databází. Pro podporu tohoto procesu vznikají komplexní nástroje. Právě rozšíření funkčnosti jednoho z těchto systémů, který bude používán především pro potřeby výuky na Fakultě informačních technologií, se zabývala tato práce.

Diplomová práce přímo navazuje na práci Ing. Krásného a práci Ing. Madera [4] [5]. Jejím cílem bylo shrnout problematiku předzpracování dat a následně vyvíjený systém v tomto směru vylepšit tak, aby systém mohl být plnohodnotně používán. Tohoto cíle bylo v práci dosaženo tím, že byly rozšířeny existující a navrženy nové komponenty procesu předzpracování dat. Z tohoto návrhu vychází i článek publikovaný na konferenci EEICT na téma *Data Preprocessing for Data Mining System* [12] oceněný prvním místem v magisterské kategorii Informačních systémů.

Rozšíření stávajících komponent spočívalo v *rozsáhlých úpravách GUI* a především v *modifikaci stávajících algoritmů*. Přepracována byla komponenta výběru dat, která byla rozšířena o import datových souborů, a komponenty VIMEO a transformací dat. U nich byly doplněny chybějící funkce předzpracování jako například *diskretizace atributů a funkce čištění dat*. Za hlavní přínos zde však považuji přepracování většiny algoritmů tak, aby *zpracování dat proběhlo na straně databázového serveru* pomocí SQL dotazů a uložených PL/SQL procedur, čímž došlo k **značné časové optimalizaci** celého procesu předzpracování.

Hlavní těžiště práce ale bylo v *návrhu a implementaci komponenty pro statistickou analýzu a vizualizaci dat*. V rámci komponenty jsem implementoval třídu statistických výpočtů, vizualizaci dat pomocí histogramů, krabicových grafů, bodových grafů, atp. Pro ověřování závislosti atributů jsem zpracoval i podporu korelační analýzy pro numerické i kategorické atributy.

Systém jsem dále rozšířil o *podporu redukce dat* různými metodami, která v systému doposud chyběla. Rovněž byly řešeny další dílčí problémy jako například rozdělení dat na trénovací a testovací množinu, řešení běhu aplikace ve více vláknech, či vymezení databázové vrstvy v projektu.

Zároveň s tímto projektem jsou řešeny dolovací moduly, konkrétně jde o klasifikační metody *rozhodovací strom*, *bayesovská klasifikace* a predikční metodu *regrese*. Dalším rozvojem projektu a rozšířeními, které bude třeba vyřešit, jsem se podrobně zabýval v kapitole 9. Nicméně po dokončení rozpracovaných dolovacích modulů by se systém mohl začít zkušebně používat ve výuce.

Literatura

[1] Fajmon, B.; Růžičková, I.: *Matematika 3*. FEKT VUT v Brně, Brno, 2005.

[2] Han, J.; Kamber, M.: *Data Mining: Concepts and Techniques*. Elsevier Inc., druhé vydání, 2006, ISBN 978-1-55860-901-3, 770 s.

[3] Kotásek, P.: *DMSL: The Data Mining Specification Language*. Dizertační práce, FIT VUT v Brně, Brno, 2003, [Online; navštíveno 2. 12. 2008].
URL http://www.fit.vutbr.cz/~kotasekp/dmsl/

[4] Krásný, M.: *Systém pro dolování z dat v prostředí Oracle*. Diplomová práce, FIT VUT v Brně, Brno, 2008.

[5] Mader, P.: *Dolovací moduly systému pro dolování z dat v prostředí Oracle*. Diplomová práce, FIT VUT v Brně, Brno, 2009.

[6] NetBeans: Co je NetBeans? 2008, [Online; navštíveno 29. 12. 2008].
URL http://www.netbeans.org/index_cs.html

[7] NetBeans: NetBeans Visual Library in NetBeans Platform 6.0. 2008, [Online; navštíveno 29. 12. 2008].
URL http://graph.netbeans.org/

[8] Object Refinery Limited: JFreeChart. 2008, department of Statistics, Tung Hai University, [Online; navštíveno 20. 2. 2009].
URL http://www.jfree.org/jfreechart/

[9] Oracle: Oracle Data Mining. 2005, [Online; navštíveno 29. 12. 2008].
URL http://download.oracle.com/docs/pdf/B14340_01.pdf

[10] Pyle, D.: *Data Preparation for Data Mining*. Academic Press, 1999, ISBN 1-55860-529-0.

[11] Rektorys, K.: *Přehled užité matematiky.II*. Prometheus, šesté vydání, 1995, ISBN 80-85849-62-3.

[12] Šebek, M.: Data Preprocessing for Data Mining System. In *Proceedings of the 15th Conference Student EEICT 2009: Volume 2*, Brno: Vysoké učení technické v Brně, Duben 2009, ISBN 978-80-214-3868-2, s. 208–210.

[13] Urman, S.; Hardman, R.; McLaughlin, M.: *Oracle: Programování v PL/SQL*. Computer Press, první vydání, 2007, ISBN 978-80-251-1870-2, 720 s.

[14] Wikipedia: Comma-separated values – Wikipedia, The Free Encyclopedia. 2008, [Online; navštíveno 5. 3. 2009].
URL http://en.wikipedia.org/wiki/Comma-separated_values

[15] Wikipedia: NetBeans – Wikipedia, The Free Encyclopedia. 2008, [Online; navštíveno 29. 12. 2008].
URL http://en.wikipedia.org/w/index.php?title=NetBeans&oldid=260219076

[16] Wikipedia: Quantile – Wikipedia, The Free Encyclopedia. 2008, [Online; navštíveno 3. 12. 2008].
URL http://en.wikipedia.org/w/index.php?title=Quantile&oldid=255600772

[17] Wikipedie: Charakteristika náhodné veličiny – Wikipedie: Otevřená encyklopedie. 2008, [Online; navštíveno 3. 12. 2008].
URL http://cs.wikipedia.org/w/index.php?title=Charakteristika_n%C3%A1hodn%C3%A9_veli%C4%8Diny&oldid=2129856

[18] Wikipedie: Rozptyl (statistika) – Wikipedie: Otevřená encyklopedie. 2008, [Online; navštíveno 3. 12. 2008].
URL http://cs.wikipedia.org/w/index.php?title=Rozptyl_(statistika)&oldid=3221727

[19] Wikipedie: Směrodatná odchylka – Wikipedie: Otevřená encyklopedie. 2008, [Online; navštíveno 3. 12. 2008].
URL http://cs.wikipedia.org/w/index.php?title=Sm%C4%9Brodatn%C3%A1_odchylka&oldid=3303962

[20] Zendulka, J.; Bartík, V.; Lukáš, R.; aj.: *Získávání znalostí z databází – studijní opora*. FIT VUT v Brně, Brno, 2006.

Seznam použitých zkratek a symbolů

API (*Application Programming Interface* – rozhraní pro programování aplikací) – seznam funkcí, metod tříd a rozhraní knihovny, které může programátor použít.

CSV (*Comma Separated Values* – čárkou oddělené hodnoty) – formát souboru, kde jsou sloupce hodnot odděleny zvoleným oddělovačem, typicky čárkou.

DDL (*Data Definition Language* – jazyk pro definici dat) – jde o rodinu jazyků, kterými lze měnit struktura databáze (vytvářet, modifikovat, odstraňovat, či jinak ovlivňovat strukturu databáze).

DML (*Data Manipulation Language* – jazyk pro manipulaci s daty) – jde o rodinu jazyků, pomocí kterých dochází k manipulaci s daty, tj. jejich vkládání, aktualizací či mazání z databáze. V současné době je velmi populární verze DML daná standardem SQL.

DMSL (*Data Mining Specification Language* – jazyk pro popis dolování z dat) – jde o jazyk ve formátu XML, pomocí něhož lze popsat jednotlivé kroky dolovacího procesu.

GUI (*Graphical User Interface* – grafické uživatelské rozhraní) – rozhraní pro interakci lidí s elektronickými zařízeními pomocí grafického výstupu.

JDBC (*Java Database Connectivity* – rozhraní připojení Java aplikace k databázi) – definice jednotného rozhraní pro přístup k relačním databázím z prostředí jazyku Java.

MySQL – je open-source řešení databázového serveru. Databáze je relační a obsahuje dotazovací jazyk založený na standardu SQL. Aktuální verzí je MySQL verze 5.

ODM (*Oracle Data Mining* – dolování dat v prostředí Oracle) – doplněk databázového systému Oracle podporující analýzu dat ve smyslu implementovaných algoritmů dolování z dat.

PL/SQL (*Procedural Language/Structured Query Language*) – procedurální nadstavba dotazovacího jazyka SQL na databázovém serveru Oracle.

SQL (*Structured Query Language* – strukturovaný dotazovací jazyk) – používaný deklarativní jazyk pro komunikaci s databázovými servery. Jeho historie sahá až do doku 1970 a poslední standard je z roku 2006.

VIMEO (*Valid, Invalid, Missing, Empty, Outlier*) – příznaky hodnot zavedené jazykem DMSL pro čištění dat.

XLS (*Microsoft Excel Spreadsheet*) – přípona tabulkového souboru vytvořeného aplikací Microsoft Excel.

XML (*Extensible Markup Language* – rozšiřitelný značkovací jazyk) – jde o univerzální značkovací jazyk, kde sémantika značek není přesně daná. Používá se ke sdílení informací nezávisle na použitém informačním systému.

Seznam příloh

Dodatek A – Uživatelský manuál k předzpracování dat

Dodatek B – Úpravy definice dokumentu DMSL

Dodatek C – Obsah přiloženého CD

Dodatek A

Uživatelský manuál k předzpracování dat

Tento uživatelský manuál se snaží představit základní možnosti systému v oblasti podpory předzpracování dat a popis použití dolovacích modulů je dodán ke konkrétním modulům.

A.1 Instalace systému

- Instalace aplikace pro používání (potřebné aplikace pro prostředí MS Windows jsou na přiloženém CD):
  1. nainstalovat distribuci *Java SDK 6* nebo *Java JRE 6*,
  2. *(doporučené)* na straně serveru spustit skript pro vytvoření používaných uložených procedur `procedures_sql.sql`,
  3. spustit aplikaci `/dataminer/dist/data_miner/bin/data_miner.exe`.
- Instalace systému pro další vývoj:
  1. nainstalovat distribuci *Java SDK 6*,
  2. *(nepovinné)* nainstalovat nástroj *SQL Developer 5*,
  3. nainstalovat vývojové prostředí *NetBeans 6.5*,
  4. *(doporučené)* na straně serveru spustit skript pro vytvoření používaných uložených procedur `procedures_sql.sql`,
  5. v prostředí NetBeans otevřít projekt `/dataminer/sources/`.

A.2 Práce s nástrojem a založení projektu

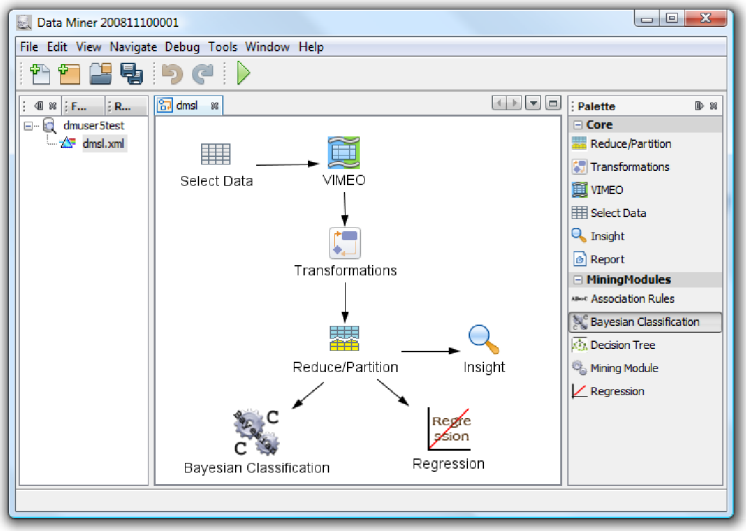
Základem práce je však založení projektu (vytvoření dokumentu `dmsl.xml`). Použijeme průvodce:

1. v menu **File** vybereme **New Project ...**,
2. v průvodci založíme **Data Mining Project**,
3. dále zadáme především název projektu,

4. a v posledním kroku vytvoříme nové spojení s databází Oracle **New Oracle database connection**,

5. po dokončení máme založený nový projekt; otevřeme jej vlevo v seznamu projektů poklikáním na `dmsl.xml` dokument. Systém nás ještě jednou vyzve pro zadání hesla k sekundárnímu připojení na server a tím máme vytvořený nový projekt. Pracovní plochu projektu vidíme podobně jako na obrázku A.1.

Základní princip systému je ve spojování uzlů představující jednotlivé komponenty. Mezi uzly se předávají jednotlivé databázové tabulky tak, jak jsou nad komponentami definované jednotlivé operace. Uzly přetahujeme z nástrojového panelu (napravo) a spojujeme tažením myši spolu se stisknutou klávesou `CTRL`. Vstup je povolen jeden, výstupy lze připojovat neomezeně.



Obrázek A.1: Pracovní plocha aplikace

A.3 Komponenta výběru dat *Select Data*

Pro každý dolovací projekt potřebujeme vstupní data. Ty do projektu přidáme komponentou výběru dat – tzv. *Select Data*.

Import dat z CSV

Komponenta umožňuje data importovat z lokálního souboru ve formátu CSV (obrázek A.2):

1. vybereme záložku **Import Data CSV** v komponentě výběr dat,

2. zvolíme odpovídající separátor importovaného souboru **Select file separator** a vybereme soubor k importu pomocí **Choose file...**,

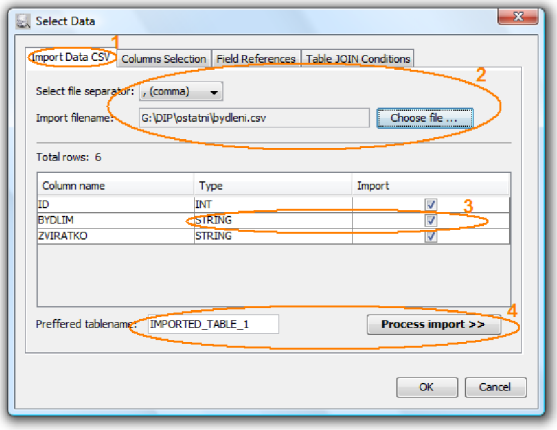
3. v zobrazené nabídce sloupců k importu vybereme ty, které chceme mít ve výsledné tabulce pomocí parametru **Import** a zkontrolujeme navržené datové typy sloupců (případně změníme) pomocí parametru **Type**,

4. zadáme název tabulky **Prefered tablename** (nesmí se na serveru vyskytovat) a importujeme pomocí **Process import** ». Průvodce nás přesměruje na další bod s předvybranou tabulkou.

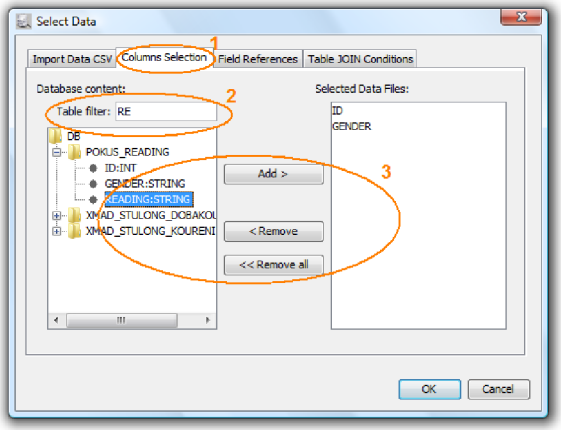
Výběr dat

Stěžejní je výběr dat vstupujících do aplikace. Ten provedeme následovně (obrázek A.3):

1. vybereme záložku **Columns Selection**,
2. pokud máme v databázi více tabulek, je možné předvyplnit filtr názvu,
3. vybereme sloupce k importu, vybrat lze rovněž celé tabulky či množiny sloupců a tabulek pomocí standardních označovacích klávesových zkratek `SHIFT` a `CTRL`. Názvy sloupců musí být unikátní, v případě kolize systém nabízí jejich přejmenování.



Obrázek A.2: Výběr dat – import CSV



Obrázek A.3: Výběr dat – výběr sloupců

Připojení referencí do číselníků

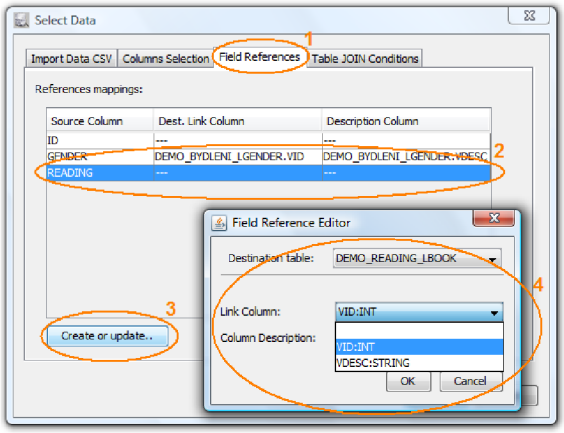
Pokud máme ve vstupních datech k dispozici pouze odkazy do číselníků v podobě cizích klíčů, nadefinujeme spojení s číselníky následovně (obrázek A.4):

1. vybereme záložku **Field References**,
2. zvolíme sloupec, který využívá reference pomocí seznamu,
3. a zvolíme navázání sloupce na číselník pomocí **Create or update...**,
4. kde pomocí dialogu naváženeme vybraný sloupec na tabulku **Destination table** pomocí klíče **Link Column** a vybereme sloupec číselníku s popisem dat **Column Description**.
5. Pokud chceme odstranit existující referenci, vybereme ji a zvolíme **Remove condition**.

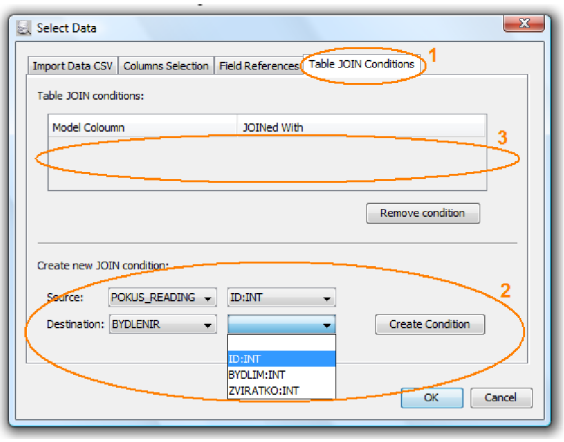
Definice spojovacích podmínek tabulek

Při výběru dat z více tabulek se implicitně provede kartézský součin záznamů v těchto tabulkách. Pro omezení spojování vytvoříme podmínku spojení na základě rovnosti hodnot sloupců (obrázek A.5):

1. vybereme záložku **Table JOIN Conditions**,
2. vybereme dvojici tabulek a příslušných sloupců na jejichž rovnosti proběhne spojení,
3. výsledné spojovací podmínky vidíme v tabulce. Pomocí **Remove condition** můžeme označené případně odebrat.



Obrázek A.4: Výběr dat – číselníky



Obrázek A.5: Výběr dat – JOIN podmínky

A.4 Komponenta analýzy dat *Insight*

Komponenta *Insight* podporuje kromě náhledu do samotných dat i řadu možností statistické a korelační analýzy dat.

První co je nutné provést před každou akcí je povinný výběr dat (sloupců) k analýze. Pro výběr stačí kliknout do dat daného sloupce nebo při výběru více sloupců použijeme kláves `SHIFT` nebo `CTRL`. Při aplikaci nástrojů je nutné respektovat typy sloupců a počet vybraných sloupců (např. *korelace dvou pouze kategorických* sloupců).

Vybrané sloupce se zobrazují ve výčtu **Selected columns to analyze**. Pokud máme vybrané právě dva sloupce, zobrazí se jejich závislost v podobě veličiny X a Y pomocí položek **Column X** a **Column Y**. Níže je charakteristika možností komponenty (ukázka na obrázku A.6).

Show statistics – zobrazí sumarizační tabulku statistických veličin pro vybrané numerické atributy.

Export data to CSV/table – export dat do CSV souboru nebo tabulky na serveru.

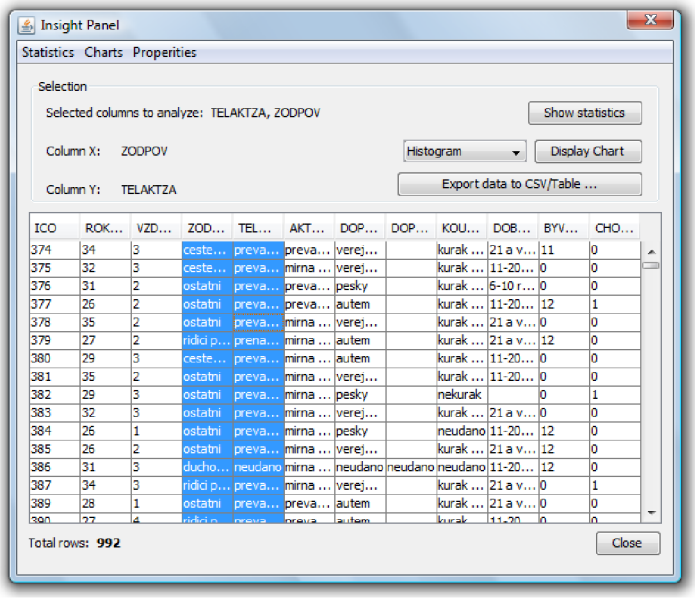
Display Chart – zobrazí graf, jehož typ je uveden v roletkovém menu. Musí být respektován počet vybraných sloupců a jejich typ (např. na kategorický atribut lze aplikovat pouze histogram).

Statistics menu / Calculate Correlation – spočítá korelační koeficient (χ^2-test) pro dva vybrané numerické (kategorické) atributy. U χ^2-testu bude uživatel vyzván k zadání hladiny významnosti testu.

Charts menu – zobrazí příslušné statistické grafy k vybraným sloupcům.

Properities menu / Apply Field References – zapne mód s nahrazením hodnot z číselníků.

Properities menu / Show Primary/Secondary Partition – přepíná mezi zobrazením primárního a sekundárního výstupu předcházející komponenty.



Obrázek A.6: Komponenta analýzy dat

A.5 Definice uživatelských funkcí

Definice uživatelských funkcí je nezbytná pro VIMEO a transformační komponentu. Dle toho se také liší možné návratové hodnoty, buď jde o VIMEO hodnotu, nebo klasickou hodnoty standardních datových typů. Definice se provádí ve 2 krocích.

První krok – definice hlavičky funkce (obrázek A.7):

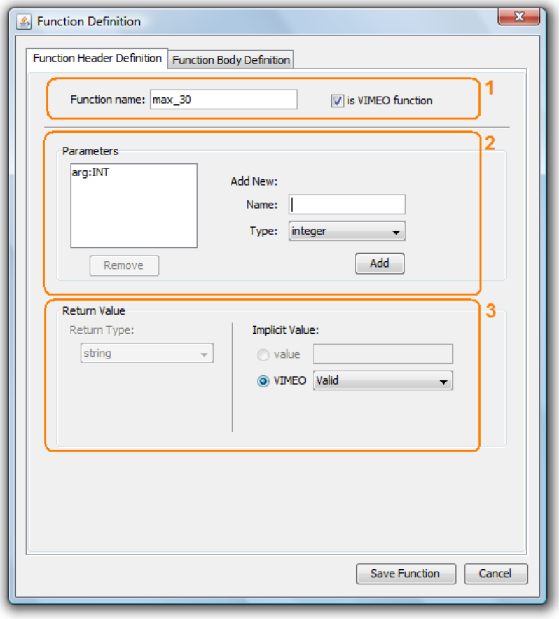
1. nadefinujeme unikátní jméno funkce **Function name** a zvolíme, zda se bude jednat o VIMEO, nebo standardní funkci vlastností **is VIMEO function**.

2. Nadefinujeme seznam parametrů s jejich konkrétními typy. Jména parametrů jsou používána při definici těla funkce.

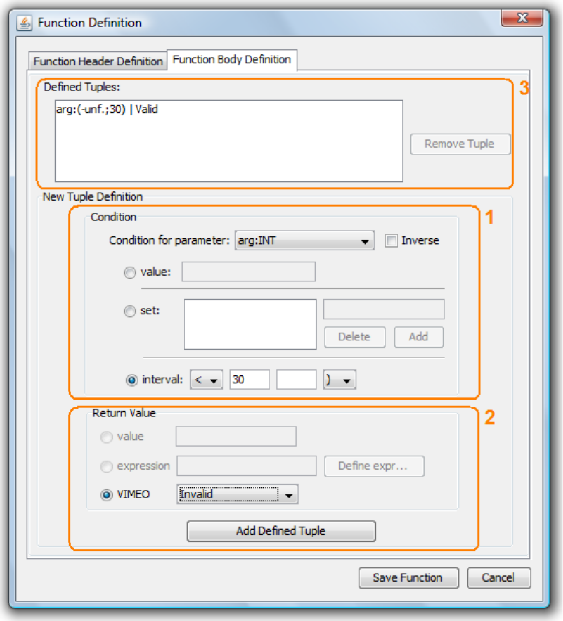
3. Vybereme návratový typ funkce **Return Type** (v případě, že nejde o VIMEO funkci) a zvolíme výchozí návratovou hodnotu (použije se v případě, že v těle funkce neodpovídá žádná `TUPLE`).

Druhý krok – definice těla interní funkce (obrázek A.8):

1. nadefinujeme podmínky postupně pro všechny parametry postupným výběrem **Condition for parameter**. Podmínka může být na základě rovnosti hodnoty, náležitosti množině nebo intervalu;
2. zvolíme návratovou hodnotu pro podmínku specifikovanou prvním bodem. V případě ne-VIMEO funkce můžeme nadefinovat nad numerickými hodnotami i *výraz* (**expression**), který bude výsledkem funkce. Po nadefinování přidáme `TUPLE` tlačítkem **Add Defined Tuple**.
3. V **Defined Tuples** vidíme kompletní definované tělo funkce.



Obrázek A.7: Definice hlavičky funkce



Obrázek A.8: Definice těla funkce

A.6 Komponenta čištění dat *VIMEO*

Komponenta VIMEO umožňuje primárně čištění dat ve zobecněném principu VIMEO. Princip spočívá v nadefinování funkce vracející jednu z hodnot VIMEO a následně jsou definovány reakce na tyto hodnoty nad vybranými sloupci. Možnými reakcemi jsou:

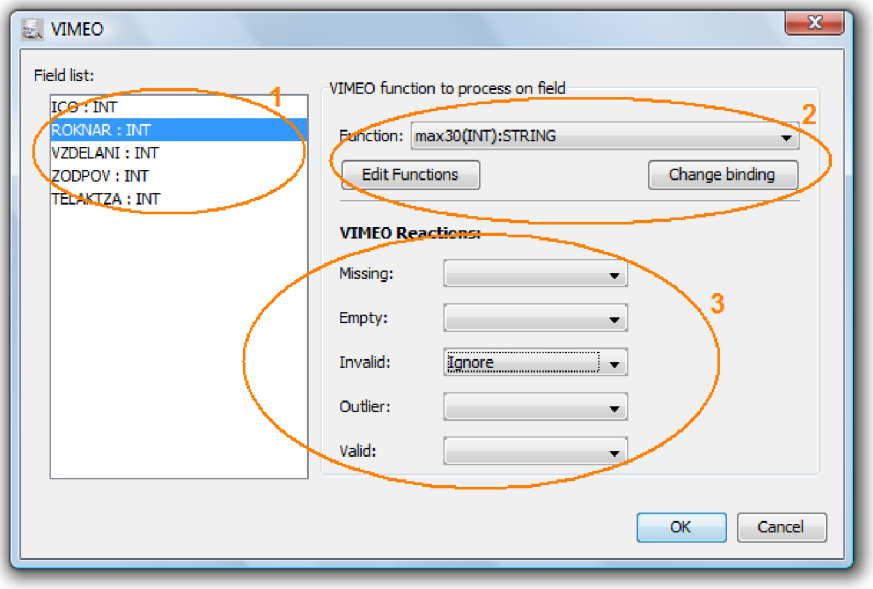
- ignorování záznamu,
- nahrazení konstantou,
- nahrazení průměrem,
- nahrazení průměrem hodnot z konkrétní třídy.

Postup definice VIMEO reakce je následující (obrázek A.9):

1. vybereme sloupec, nad kterým chceme definovat VIMEO funkci **Field list**,

2. nadefinujeme (více v kapitole A.5) a vybereme vyhodnocovací funkci. Té přiřadíme parametry, se kterými je volána, a tím máme nachystané volání této funkce.

3. Pro zvolené VIMEO hodnoty nadefinujeme příslušné reakce, které budou provedeny v závislosti na návratové hodnotě volané funkce.



Obrázek A.9: Nastavení komponenty VIMEO

A.7 Komponenta transformací *Transformations*

V komponentě transformací dat definujeme operace, které se provedou nad celými sloupci vstupní množiny dat následovně (obrázek A.10):

1. vybereme sloupec, který chceme převést na výstup v **Input Fields**,

2. zvolíme příslušnou transformaci sloupce na výstup. V případě vytvoření nového odvozeného sloupce není třeba označovat žádný vstupní sloupec.

3. Vpravo vidíme definované sloupce na výstupu a způsob jejich odvození.

Možné akce transformací jsou:

Copy, Rename, Copy All – zkopíruje sloupec na výstup bez modifikace.

Discretize – znormalizuje vybraný sloupec.

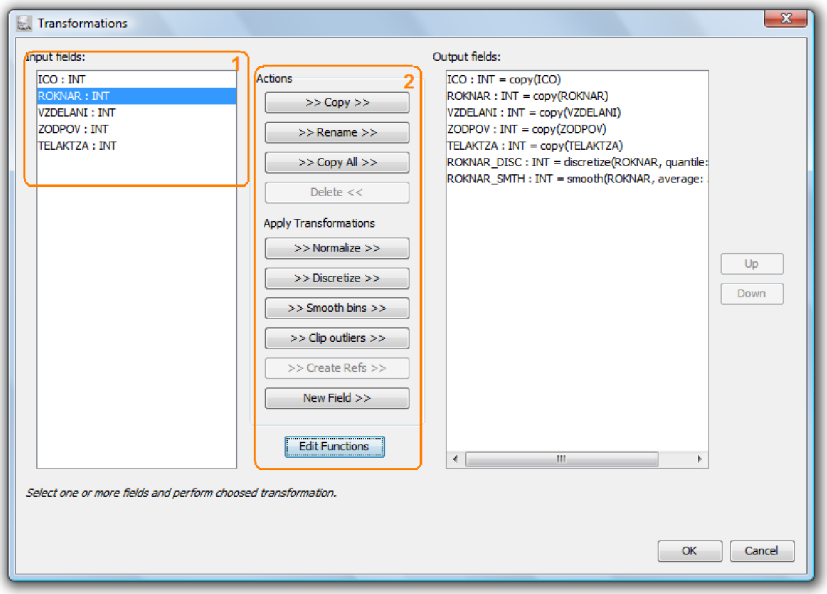
Display Chart – diskretizuje sloupec do zvoleného počtu množin zvolenou metodou.

Smooth Bins – vyhladí metodou plnění pomocí zadaného počtu košů vybraným způsobem (průměrem, mediánem).

Clip Outliers – ořízne odlehlé hodnoty.

Create Refs – pro atributy typu řetězec vytvoří číselníky a převede hodnoty na odkazy.

New Field – odvodí nový sloupec definovanou funkcí (definice pomocí **Edit Functions**).



Obrázek A.10: Nastavení komponenty transformací dat

A.8 Komponenta redukce a rozdělení dat *Reduce/Partition*

Komponenta redukce a rozdělení dat podporuje redukci dat na svém výstupu pomocí různých metod a následně zvolit rozdělení dat na trénovací a testovací množiny dat na výstupu:

1. zvolíme poměr redukce dat **Data Reduction** a metodu, pomocí které bude redukce prováděna **Reduction method**.
2. Pokud požadujeme rozdělení dat na trénovací a testovací disjunktní množiny, aktivujeme funkci **Enable data partitioning**
3. a upravíme poměr rozdělení množin **Partition ratio**.

A.9 Dolování z dat

Práce s dolovacími moduly se liší v závislosti na konkrétním modulu. Obecně lze pouze říct, že dolovací moduly klasifikace a predikce **je nutné** připojit na komponentu *Reduce/Partition* se zapnutou podporou rozdělení dat na trénovací a testovací množiny.

Dodatek B

Úpravy definice dokumentu DMSL

Změna definice je chápána inkrementálně vůči definici v [4] a ji upravující definici v [5].

```
<!-- Value interpretation and treatment -->

<!ENTITY % REDUCTION-METHOD "SRSWOR | SRSWR | STRATIFIED | default" >

<!-- Header -->
<!-- ###### -->

<!ELEMENT Header ((DMSLProducer | Annotation)+,ProjectId?) >
<!ELEMENT ProjectId   %INT-NUMBER; >

<!-- Data Mining Matrix -->
<!-- ################## -->

<!ELEMENT DataMiningMatrix ((UseMatrix | (Query, UseMatrix*)?),
              DataMiningField*, MatrixTreatment*, Annotation?) >
<!ATTLIST DataMiningMatrix    name            CDATA           #REQUIRED
                              partitionName   CDATA           #IMPLIED
                              partitionRatio  (%INT-NUMBER;)  #IMPLIED >

<!ELEMENT UseMatrix (UseField*) >
<!ATTLIST UseMatrix    matrixRef           CDATA              #REQUIRED
                       matrixTreatmentRef  CDATA              #IMPLIED
                       dataUseRatio    (%INT-NUMBER;)         #IMPLIED
                       dataUseMethod   (%REDUCTION-METHOD;)   #IMPLIED
                       dataUseStratifiedColumn  CDATA         #IMPLIED >
```

Dodatek C

Obsah přiloženého CD

Adresářová struktura přiloženého datového média:

- `apps` – aplikace doporučené pro další vývoj/běh projektu
- `dataminer` – hlavní složka programové realizace
  - `dist` – distribuční verze systému pro dolování
  - `examp-data` – ukázková data do databáze (CSV formát)
  - `sources` – **zdrojové kódy systému pro dolování**
- `docs` – dokumentace k projektu
  - `api` – programová dokumentace Javadoc
  - `tutorial` – manuál k používání systému v oblasti předzpracování
- `thesis` – text diplomové práce
  - `src` – zdrojový kód práce pro systém LaTeX