TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Vytváření rozšiřujících balíčků pro data miningové nástroje

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Lukáš Mázl**
Vedoucí práce: Ing. Marián Lamr, Ph.D.

Liberec 2019

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Creating extensions for data mining tools

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Lukáš Mázl**
Supervisor: Ing. Marián Lamr, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2018/2019**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení:	**Lukáš Mázl**
Název práce:	**Vytváření rozšiřujících balíčků pro data miningové nástroje**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**
Vedoucí práce:	**Ing. Bc. Marián Lamr**
Rozsah práce:	**30—40 stran**
Konzultant:	**RNDr. Klára Císařová, Ph.D.**

Zásady pro vypracování:

1. Seznamte se s vybranými data miningovými postupy, nástroji a algoritmy.
2. Prostudujte možnosti rozšiřitelnosti vybraných data miningových nástrojů o další algoritmy.
3. Naprogramujte vlastní sadu algoritmů a analytických rozšíření pro vybraný data miningový nástroj.
4. Vytvořte případovou studii, ve které budete ilustrovat využití naprogramovaných rozšíření.

Seznam odborné literatury:

[1] BERKA, Petr, 2005. Dobývání znalostí z databází. Praha: Academia. ISBN 8020010629.

[2] HAN, Jiawei a Micheline KAMBER. Data mining: concepts and techniques. San Francisco: Morgan Kaufmann Publishers, c2001. Morgan Kaufmann series in data management systems. ISBN 1558604898.

[3] WITTEN, I. H. a Frank EIBE, 2017. Data mining: practical machine learning tools and techniques. Fourth Edition. Cambrige: Morgan Kaufmann. ISBN 9780128042

[4] ŘEZANKOVÁ, Hana, Dušan HÚSEK a Václav SNÁŠEL. Shluková analýza dat. 2., rozš. vyd. Praha: Professional Publishing, 2009. ISBN 978-80-86946-81-8.

[5] SCHILDT, Herbert. Mistrovství - Java. 1. vyd. Brno: Computer Press, 2014. Mistrovství. ISBN 978-80-2-1-4145-8.

V Liberci dne .....

.....
Ing. Bc. Marián Lamr

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

29. 4. 2019 Lukáš Mázl

Abstrakt

Tato bakalářská práce demonstruje možnosti rozšiřitelnosti data miningových nástrojů. Práce zahrnuje základní popis struktury projektu KNIME a znázorňuje kroky které je nutné splnit pro vytvoření vlastního uzlu. V průběhu této práce byly vytvořeny uzly pro shlukovou analýzu tj. DBSCAN a OPTICS. Uzel OPTICS zároveň umožňuje generování asociačních pravidel pro vybrané skupiny. Mezi další uzly vytvořené v rámci této práce patří uzel pro vizualizaci dat (Data Audit). V neposlední řade byl také vytvořen uzel, který umožňuje stahování dat o hustotě dopravy území České republiky.

Klíčová slova: Data mining, KNIME, rozšiřující balíčky, DBSCAN, OPTICS, APRIORI, Data Audit, Hustota dopravy

Abstract

This bachelor thesis demonstrates the possibilities of extensibility of data mining tools. The work includes a basic description of the KNIME project and shows the steps you must take to create your own node. During this work, nodes for cluster analysis were created, ie DBSCAN and OPTICS. At the same time, the OPTICS node allows the generation of association rules for selected groups. Other nodes created within this work include a Data Audit node. Last but not least, a node was created that allows downloading of traffic density data for the territory of the Czech Republic.

Key words: Data mining, KNIME, expansion packs, DBSCAN, OPTICS, APRIORI, Data Audit, Traffic density

Poděkování

Rád bych poděkoval Ing. Mariánu Lamrovi, Ph.D. za cenné rady, věcné připomínky, vstřícnost při konzultacích a při sepisování této bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

GPS	Global Positioning System
API	Application Programing Interface
CSV	Comma-separated values
XLS	Excel Spreadsheet
MLP	Multiple Layer Perceptron
PNN	Probabilistic neural network
SVM	Support Vector machines
PCA	Principal Component Analysis
MDS	Multidimensional scaling
HTML	HyperText Markup Language
PDF	Portable Document Format
JAR	Java Archive
SDK	Software Development Kit
IDE	Integrated Development Enviroment
JVM	Java Virtual Machine
XML	Extensible Markup Language
JTSK	Jednotná trigonometrická síť katastrální
PNG	Portable Network Graphics
SQL	Structured Query Language

1 Úvod

Tato bakalářská práce se zabývá vytvořením data miningových uzlů pro prostředí KNIME, které nadále budou složit k analýze dopravních nehod na území České republiky. Jedním ze základních úkolů této bakalářské práce je seznámení se podrobněji s vybranými data miningovými nástroji a algoritmy. S ohledem na potřeby vyhledávání specifický shluků dopravních nehod byly vybrány pro implementaci algoritmy DBSCAN, OPTICS a APRIORI.
Úkolem této bakalářské práce je vytvořit balíček uzlů, které budou schopny vizualizovat, analyzovat a upravovat dostupná data. Algoritmus DBSCAN a OPTICS umožňují detekovat shluky častých dopravních nehod na základě GPS souřadnic. Tyto jednotlivé shluky by mělo být následně podrobit další analýze. K analýze shluků, které budou vyhledány za pomoci uzlu DBSCAN nebo OPTICS, bude sloužit algoritmus APRIORI umožnující vyhledávat asociační pravidla na základě uživatelem zadaných vstupních parametrů. Součástí baličku jsou i další uzly umožnující data analyzovat (DataAudit) a dále s nimi jednoduše manipulovat (MyFilter).
Tyto vytvořené uzly jsou testované na reálných datech, kterými jsou záznamy o dopravních nehodách v České republice. Tato databáze uchovává až 700 000 záznamů o dopravních nehodách.
Data miningové prostředí KNIME nabízí uživateli před-připravené uzly DBSCAN a APRIORI, ale nejsou použitelné pro analýzu dat, na příklad z důvodu složitosti vstupných parametrů, které vstupují do shlukovacích algoritmů. Tyto parametry musí být uživatelem před-počítat do reálného prostoru.
Algoritmus APRIORI, který se nachází v KNIME nedovoluje vytvářet asociační pravidla na základě zvolené skupiny. Jedním z úkolů je umožnit tomuto uzlu vytváření asociačních pravidel na základě zadané skupiny. V případě dopravních nehod se jedná o vytvoření asociačního pravidla na základě shluku, získaného z jednoho z výše zmíněných shlukovacích algoritmů.
Pro přípravu dat byly vytvořeny data miningové uzly, které umožňují filtrování dat nebo rozšiřování datové matice o řadu dalších atributů popisujících hustotu dopravy (DensityFinder).

2 Data mining, data miningové nástroje a jejich rozšiřitelnost

2.1 Data mining

Data mining je proces pro získání netriviálně skrytých nebo potenciálně užitečných dat. Někdy se tomuto procesu říká dobývání znalostí z databází (Knowledge Discorevy in Databases). S data miningem je možné se setkat v oblasti marketingu pro doporučení zboží zákazníkům, politiky, ve vědě nebo na sociálních stránkách. V oblasti marketingu je data mining využíván k rozpoznání potřeb jednotlivých zákazníků, aby bylo možné zjistit co zákazníkovi nejlépe nabídnout. Data mining je také využíván například na sociálních sítích pro doporučování obsahu uživatelům na základě jejich aktivit na internetu. Další oblastí je monitorování aktivit na internetu s cílem odhalit činnost potenciálních škůdců.[1] V dnešní době je kladen především důraz na rychlost algoritmů, aby tyto algoritmy byly schopny zpracovávat data v reálném čase.
První náznaky data miningu se objevili již v 60. letech 20. století, kde vznikaly první rozhodovací stromy. Následný rozvoj statistických metod, databázových aplikací a umělé inteligence umožnil velký vzestup data miningu.[2]

2.2 Data miningové nástroje

Data miningovými nástroji je chápán jakýkoliv nástroj, který analyzuje data a snaží se v nich najít skryté souvislosti. Nemusí jít vždy o software s propracovaným uživatelským prostředím ale může se jednat o programovací jazyk.

2.2.1 Programovací jazyky

Mezi nejpoužívanější data miningové nástroje využívající programovací jazyk patří jazyk R a Python. Oba poskytují uživatelům možnost manipulace s daty, vizualizaci pomocí grafů i analýzu pomocí připravených nástrojů.
V jazyce R uživatel může nalézt přes 4 000 balíčků, které umožňují statistickou analýzu. Mimo jiné tyto balíčky mohou být také použity v programovacím jazyku Python.
Nevýhodou jazyku R je jeho složitost, kvůli velikému množství funkcí je velice obtížné ze začátku být s tímto nástrojem efektivní. Mnoho programátorů využívá Py-

thon pro přípravu dat a vytváření komplexních analýz které mohou zabrat několik minut. Toto dělá z Pythonu efektní data miningový nástroj.

2.2.2 Data miningové nástroje s uživatelským prostředím

Častěji používanými nástroji jsou nástroje s uživatelským rozhraním, které umožňují vizualizaci práce s daty. Uživateli jsou zobrazena data v jednotlivým fázích (při načtení, během analýzy a zpracované výsledky). Nástroje s uživatelským rozhraním jsou omezeny tím, že uživatel může využít jen předem naprogramovaný set funkcí. Často je pro rozšíření funkcionality daného prostředí o další možnosti potřeba napsat vlastní skript, který je spuštěn v před-připraveném uzlu. Další možností je vytvoření vlastního uzlu s logikou. V takovém případě není programátor limitován uzlem nebo příkazy, které jsou použitelné při psaní skriptu.
Nejvíce používanými nástroji s uživatelským rozhraním jsou nástroje IBM SPSS Modeler, KNIME, Orange a Weka. Tyto nástroje a jejich možnosti rozšíření budou dále popsány.[3]

2.2.3 IBM SPSS Modeler

Data miningové prostředí IBM SPSS Modeler vyvinuté firmou IBM umožňuje svým uživatelům využívat množinu data miningových nástrojů bez jakékoli nutnosti programovat. Nejčastěji je tento nástroj využíván při práci na velkých projektech, které umožňují analyzovat nejrůznější typy dat. Především kvůli možnosti vizualizace dat je Modeler velice užitečným nástrojem.
Nevýhodou IBM SPSS Modeleru je jeho cena, na rozdíl od nástrojů KNIME, Weka nebo Orange není IBM SPSS Modeler volně dostupný.
IBM SPSS Modeler umožňuje vytvářet vlastní uzly, kterými může programátor rozšířit toto prostředí. V Modeleru lze také spouštět vlastní skripty v jazyce R a Python. Tento nástroj poskytuje programátorů API (Aplication Programming Interface), které umožňuje komunikovat s daným uzlem, číst, spouštět nebo přerušit jeho životní cyklus.[4]

2.2.4 Orange

Orange je data minigový nástroj napsaný v jazyce Python vyvinutý v laboratoři bioinformatiky na fakultě Informační a počítačové vědy na univerzitě v Ljublajany, který je volně dostupný a vhodný pro uživatele, kteří začínají v oblasti data miningu. Časté využití tohoto prostředí je pro text mining (podobně jako u IBM SPSS Modeleru se jedná o mining dat z textových dat), teplotní mapy či dendrogramy.[5]

2.2.5 WEKA

Slovo WEKA je akronymem pro Waikato Environment for Knowledge Analysis. Jedná se o data miningové prostředí vyvinuté univerzitou Waikato na Novém Zélandu. Tento program je napsaný v jazyce Java. Obsahuje kolekci vizualizujících nástrojů

a algoritmů pro analýzu dat. WEKA podporuje několik standardních data miningových úloh, kterými jsou preprocessing, shluková analýza, klasifikace dat, regrese a vizualizačních.
WEKA je nástroj volně dostupný. Umožňuje programátorovi spouštění skriptů, které jsou napsané v jazyce R nebo Python. [6]

2.2.6 KNIME Analytics Platform

KNIME Analytics Platform je open source nástroj pro data mining vyvinutý v jazyce Java, který v roce 2004 začal vytvářet tým z Univerzity v Kostnici (University of Konstanz). Od roku 2006 je používán ve farmakologii, k analýze financí nebo v oblatech CRM (customer relation managent – nástroje pro řízení komunikace se zákazníkem). V základní verzi KNIME nabízí uživatelům přes 3 000 uzlů pro práci s daty. Toto data miningové prostředí umožňuje uživatelům seznámit se s jeho uzly pomocí velikého množstvím před-připravených příkladů, na kterých je možné vidět využitelnost uzlu. Prostředí KNIME umožňuje uživateli vytvořit další rozšíření o nově naprogramované uzly, které jsou považovány za pluginy. Druhou možností jak tento nástroj rozšířit o před připravené uzly je stažení již připravených pluginu z oficiálních stránek KNIME.
Tento data miningový nástroj je založen na myšlence modularity a propojování uzlů pomocí pipe line. Software je vyvinut v jazyce Java. Nevýhodou je špatná dokumentace pro programátory, kteří by chtěli pro tento nástroj vytvářet uzly. Pro vytvoření uzlu programátorovi postačí implementace základních abstraktních metod. Metody execute zprostředkující spuštění logiky uzlu a dialogu pro možnost konfigurace uzlu. KNIME Analytic Platfom byl vybrán pro tuto bakalářskou práci z důvodu možnosti vytváření rozšiřujících uzlů bez nutnosti spouštění skriptů v připravených uzlech. Po sestavení je možné nově vytvořený uzel naimportovat jako plugin do KNIME prostředí. [7]

2.3 Data miningové prostředí KNIME

Tato kapitola popisuje základní strukturu prostředí KNIME a základní uzly které jsou zde dostupné. Pro seznámení s tímto nástrojem je možné využít příklady vytvořené od vývojářů KNIME.
Základní kostrou každého workflow (schéma zapojení jednotlivých uzlů za sebou pro daný výsledek) je načtení dat, na kterých se bude provádět analýza, následná manipulace (například spuštění DBSCANu na množině právě načtených dat) a výpis/vizualizace výsledků.

2.3.1 Nástroje pro načtení dat

Tato kapitola se zabývá možnostmi načítání vstupních dat, se kterými můžeme nadále pracovat a analyzovat je. KNIME umožnuje načítání dat z několika různých zdrojů. Data mohou být příklad načítána ve formátu CSV či XLS. Pokud uživatel

nechce využívat specifický uzel pro daný typ formátu může použít uzel File Reader, který dokáže sám rozpoznat formát vstupního souboru a hned ho rozpársovat. Příklad vstupných uzlů:

- File reader - Načtení a rozpoznání formátu vstupných dat a následné rozpársování.
- CSV reader - Načtení dat ve formátu CSV, kterém lze specifikovat oddělovacím znakem.
- XLS reader - Načtení dat, která jsou uložena v XLS formátu využívaného programem Microsoft Excel.
- Table reader - Načtení KNIME tabulky ze souboru.
- List files - Načtení seznamu vstupných souborů.
- Database reader - Načítání dat z databáze.

2.3.2 Nástroje pro analýzu dat

Analytické uzly v KNIME lze rozdělit do dvou kategorií. Statistické uzly, které umožňují, jak již název napovídá, získávání statistických výsledků, jako je například počet záznamů vytvoření lineární, polynomiální nebo logistické regrese. Také zde můžeme najít statistické nástroje pro testování hypotéz. Příkladem nástroje pro testování hypotéz je párový T-test nebo test ANOVA.
KNIME dále poskytuje řadu data miningových algoritmů:

- Algoritmy shlukové analýzy - DBSCAN, SOTA a K-Means
- Neuronové sítě - Multiple Layer Perceptron (MLP) nebo Probabilistic neural network (PNN)
- Rozhodovací stromy
- Algoritmy pro hledání asociačních pravidel
- Support vector machine (SVM)
- Analýza hlavních komponent (PCA)
- Multidimensional scaling (MDS)

2.3.3 Nástroje pro exportování dat

Po provedené analýze mohou být data exportována z KNIME prostředí. KNIME umožňuje exportovat tyto data pomocí následujících uzlů.

- CSV Writer - Výpis dat do formátu CSV

- Table to HTML - Vygenerování dat do HTML stránky
- Table to PDF - Vyexportování dat do formátu PDF
- Excel Writer - Vyexportování dat do excel formátu

2.3.4 Nástroje pro vizualizaci dat

Vizualizace dat umožňuje data miningovému analytikovi zobrazit data pro základní odhad k nastavení správných vstupných parametrů pro analytické uzly nebo k otestování, že výsledek po analýze je správný či očekávaný. KNIME také umožňuje vygenerování základních grafů a exportování daných grafů do formátu JPG nebo PNG.
Seznam vybraných uzlů pro vizualizaci dat.

- Interactive Table - Zobrazuje data v tabulce
- Scatter plot - Umožňuje zobrazit data ze sloupci A v závislosti na sloupci B v kartézském systému souřadnic
- Pie Char - Zobrazení dat koláčovém grafu
- Line Plot - Zobrazení dat v lineárním grafu
- Histogram - Zobrazení histogramu v různých režimech
- Color manager - V případě když chce uživatel zvýraznit data, které mají stejný atribut může použít tento uzel. (Tento uzel data nevizualizuje)
- Shape manager - Jedná se o obdobu uzlu Color manger. Tento uzel mění tvar na místo barvy

2.4 Knime SDK

Vývojové prostředí Knime SDK je postaveno na jádře Eclipse IDE, rozšířené o plugin, který programátorům umožňuje snadnější tvorbu nových uzlů. Toto vývojové prostředí umožňuje vygenerování celé strukturu projektu nově vytvářeného uzlu a jeho potřebných Java tříd. Prostředí KNIME SDK, je volně dostupné na oficiálních stránkách KNIME. Toto prostředí dovoluje naprogramovat vlastní uzel, následně ho vyexportovat ve formátu JAR a přidat mezi již před-připravené uzly.[8] Toto prostředí využívá poupravenou verzi Java 7. To může způsobit problémy s nedostupností některých funkcí, na které jsou programátoři ve standardní Javě zvyklí.
Grafické rozhraní v prostředí KNIME je vytvořeno na technologii Java Swing. Nevýhodou Java Swing je špatná čitelnost kódu a horší hledání pozdějších chyb.[9]

3 Vybrané data miningové algoritmy

Uzly, které jsou v této práci vytvořeny slouží především k usnadnění práce data miningového analytika, který se zabývá analýzou dopravních nehod a možností predikce. Pro analýzu je potřeba veliký počet dat, v některých přepadech je potřeba i správná vizualizace těchto dat, aby bylo možné si data představit v hlubším kontextu. Pro tuto vizualizaci dat je v této práci vytvořen uzel DataAudit, který umožnuje analytikovi vizualizovat data, se kterými bude nadále pracovat.
Uzly které jsou vytvořeny v této bakalářské práci můžeme rozdělit na uzly které využívají shlukujících algoritmů, uzly pro hledání asociačních pravidel, pro vizualizující dat a práci s nimi.

3.1 Algoritmy pro shlukovou analýzu

Algoritmy pro shlukovou analýzu se používají pro vyhledání objektů, které si jsou podobné ve vybraných atributech. Tyto navzájem si podobné objekty jsou pak považovány za shluk. Většina algoritmů využívá k porovnání podobnosti těchto objektů metriku. Příkladem algoritmu využívající metriky může být algoritmus DBSCAN nebo OPTICS. V této bakalářské práci jsou využity algoritmy DBSCAN a OPTICS. Pomocí těchto algoritmů jsou vyhledány shluky dopravních nehod na základě GPS souřadnic.

3.1.1 Základní metriky

Tato sekce popisuje základní metriky, které jsou použity k výpočtu vzdálenosti mezi body. V této práci se počítá podobnost bodu na základě GPS souřadnic.Nejčastější metodou je metoda Euklidova. Tato metoda byla také použita v uzlu DBSCAN, který je cílem této bakalářské práce.
Řekneme, že funkce Pm je funkce metriky, která má dva vstupní parametry body A a B. Každý bod má stejnou dimenzi. Dále řekneme že A1 až An jsou hodnoty pro dimenzi 1 až N.

Euklidovská metrika

$$pe(A,B) = \sqrt{\sum_{i=1}^{p} (a_i - b_i)^2}$$

Manhattanská metrika

$pl(A, B) = \sum_{i=1}^{p} |a_i - b_i|$

Sokalova metrika

$ps(A, B) = \sqrt{\frac{p^2 e(A,B)}{p}}$

Sup metrika

$p_\infty(A, B) = \max_{i=1..p}(|a_i - b_i|)$

3.1.2 Algoritmus DBSCAN

DBSCAN (Density-based spatial clustering of applications with noise) je shlukující algoritmus vytvořený v roce 1994, který byl navrhnut Martin Ester, Hans-Peter Kriegel, Jörg Sander a Xiaowei Xu. Tento algoritmus vyhledává množinu bodů v prostoru, které seskupí na základě metriky a sousedských bodů.[10]
DBSCAN je v současné době jedním z nejběžnějších shlukujícím algoritmem. Jeho nevýhodou je parametr který může být obtížné správně zvolit.[11]

Popis algoritmu

Mějme bod P a vzdálenost ve které se musí nacházet minimální počet sousedských bodu minPts (hustota) k tomu, aby mohl být tento bod považovaný za shluk. Pro P pomocí vybrané metody metriky se vyhledají všechny sousedské body, které se nacházejí ve vzdálenosti nebo menší. Pak na základě počtu takto nalezených bodů můžeme říci o bodu P, zda-li se jedná o šum (noise) nebo zda-li patří do shluku (cluster). Pokud bod P nesplňuje podmínku minimálního počtu bodu v sousedství, je tento bod považován za šum. Pokud je podmínka splněna a v seznamu sousedských bodů je alespoň jeden bod, který již patří do shluku C pak se bod P stává součástí shluku C. V případě, že se nenachází v seznamu ani jeden bod který by byl součástí nějakého shluku vzniká nový shluk. Také může nastat případ, kdy v seznamu je více bodů, které patří do různých shluků, v tomto případě dojde k sjednocení shluků do jednoho shluku.[12]

Základni kroky algoritmu

- Zvolení libovolného (dosud nenavštívený) bod P.
- Bod P je označen jako navštívený.
- Jsou vyhledány okolní body, které jsou od bodu P ve vzdálenosti menší nebo rovné .

- Pokud je počet bodu v okolí bodů v sousedství bodu P menší než minPts, je nastaven bod P jako šum. Pokud je větší nebo rovno minPts, pak bod přidán do shluku. V případě kdy neexistuje ještě v jeho blízkosti žádný shluk je vytvořen nový.
- Celý proces se opakuje.

3.1.3 Algoritmus OPTICS

Ordering points to identify the clustering structure (OPTICS) je algoritmus pro vyhledávání shluků na základě hustoty. Tento algoritmus byl poprvé publikován Mihael Ankerst, Markusem M. Breunigem, Hans-Peter Kriegelem a Jorg Sander. Hlavní myšlenka algoritmu je podobná jako u algoritmu DBSCAN, ale řeší hlavní slabost DBSCANu. Problém detekce smysluplných shluků v datech ve kterých je rozdílná hustota. Výsledkem OPTICS jsou data uspořádány lineárně za sebou seřazeny tak, aby u sebe byly body, které si jsou nejblíže. Výsledek lze pak prezentovat jako dendrogram.[13]

Popis algoritmu

Jako DBSCAN tak i OPTICS vyžaduje dva parametry, které popisují maximální vzdálenost (rádius) a minPts (hustotu), která popisuje minimální požadovaný počet bodů k formování shluku.
Řekněme, že bod A má ve svém okolí R daný počet sousedských bodů N. Stav bodu A je nastaven na navštívený. Pro vybraný bod A jsou vyhledány sousedské body, které jsou buď vzdáleny od bodu A ve vzdálenosti R vypočtené pomocí metriky nebo blíže. Pokud pro bod A neexistuje žádný bod ve v okolí R je vyhledán nejbližší dosud nenavštívený bod B. Vzdálenost mezi body A a B se uloží do seznamu (pořadí v seznamu se nesmí prohazovat).
Pokud daný bod má ve svém sousedství takový počet bodu, který stále ještě nesplňuje podmínku minPts, v takovém případě je v jeho sousedství vyhledán nejbližší bod. Znovu je vypočtena tato vzdálenost a přidána do seznamu.
V případě, kdy bod A má ve svém sousedství více bodu nebo rovno počtu minPts, v tom případě je vypočten tzn. core distance (vzdálenost jádra). Core distance je poloměr okolí bodu, ve kterém se nachází právě minPts bodů. Tento core distance je uložen do seznamu a dále je vybrán další bod, který je k bodu A nejblíže.[14]
Výsledkem tohoto postupu je dendrogram na kterém je viditelné jakou hustotu mají nalezené shluky. Uživatel může pak na základě dendrogramu zvolit dodatečně hustotu, podle které jsou jednotlivé body následně přiřazeny do odpovídajícího shluku.

3.2 Algoritmy pro vyhledávání asociačních pravidel

3.2.1 Algoritmus APRIORI

Algoritmus APRIORI byl navržen roce 1994 pro generování asociačních pravidel nebo také pro generování frekventovaných množin. APRIORI iterativně využívá k-množiny k vytvoření k+1 množiny. Jeho vstupními parametry jsou support a confidence. Parametr support říká kolikrát item musí být minimálně ve vybrané množině. Confidence nebo také spolehlivost umožňuje filtrovat předpoklady, kteří mají menší spolehlivost (mohou nastat s menší pravděpodobnosti než je požadována). [15]

Popis algoritmu

Algorimus APRIORI se skládá ze 2 kroků, které se opakují do té doby dokud množiny mají dostatečný support.

- Spojovací krok
- Vylučovací krok

Spojovací krok

Pro vytvoření Lk je potřeba vygenerovat množinu kandidátu Vk spojením Lk-1 se sebou samým. Nech l1 a l2 jsou množiny z Lk-1. Pro spojeni l1 a l2 dojde pokud jejich předchozí prvky jsou shodné a poslední prvek l1 je menší jako poslední prvek l2. Algoritmus APRIORI předpokládá, že položky v k-množinách jsou lexikograficky uspořádané.

Vylučovací krok

Množina kandidátu je nad množinou Lk a to znamená, že jejich prvky můžou, ale nemusí být frekventovanými množinami. Aby bylo zajištěno pro každého kandidáta jestli splňuje minimální podporu, je nutné projít celou databází. Pokud je množina kandidátu velká, může tato operace trvat velice dlouho. Dokud existuje nějaká (k-1) množina, která je podmnožinou kandidátní k-množniny a je obsažena v Lk-1, potom je tato množina kandidátů odstraněna.

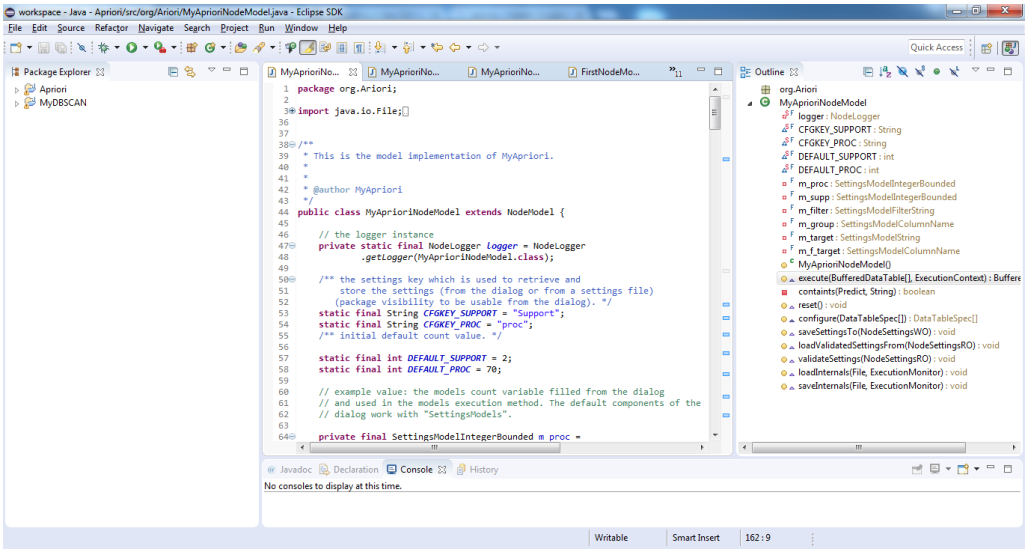
Využití nástroje APRIORI v KNIME

Nástroj, který využívá algoritmu APRIORI v KNIME prostředí již existuje, bohužel nedovoluje vygenerování asociačních pravidel pro více skupin najednou. Řekněme, že v datech o zákaznicích chceme vygenerovat asociační pravidla pro jednotlivé věkové skupiny. V takovém případě v prostředí KNIME nezbývá nic jiného než vyfiltrovat data pro děti a dospělé. Vytvořit 2 nové instance uzlu APRIORI, které vygenerují asociační pravidla a následně tyto data spojit (merge). Pro tento příklad se nejedná o obtížnou záležitost, ale v případě, kdy daných skupin je více nastává problém.
Tento uzel by měl být využit pro vygenerování asociační pravidel pro nalezené

shluky, které budou vyhledány pomocí již zmiňovaných shlukujících algoritmu DBSCANu a OPTICS. Pokud by musel uživatel pro každý nalezený shluk postupovat výše zmíněnou metodou, mohla by tato aktivita trvat velice dlouhou dobu. Z tohoto důvodu je vytvořen uzel nový uzel, který tuto možnost umožňuje.[16]

4 Příprava prostředí KNIME SDK pro vývoj

KNIME SDK je plug-in vytvořený pro prostředí Eclipse Neon, který pomáhá programátorovi s přípravou struktury nově vznikajícího uzlu. Výhoda tohoto plug-inu je jeho možnost spuštění nového uzlu bez nutnosti manuálního sestavování či instalace uzlu přímo do KNIME prostředí. Toto může být velice časově náročná činnost. V případě, že se jedná pouze o testování správné funkčnosti uzlu ušetří tento plug-in programátorovi velkou část času.
Plugin KNIME SDK pro testování uzlů využívá vlastní instanci KNIME a také vlastní workspace.



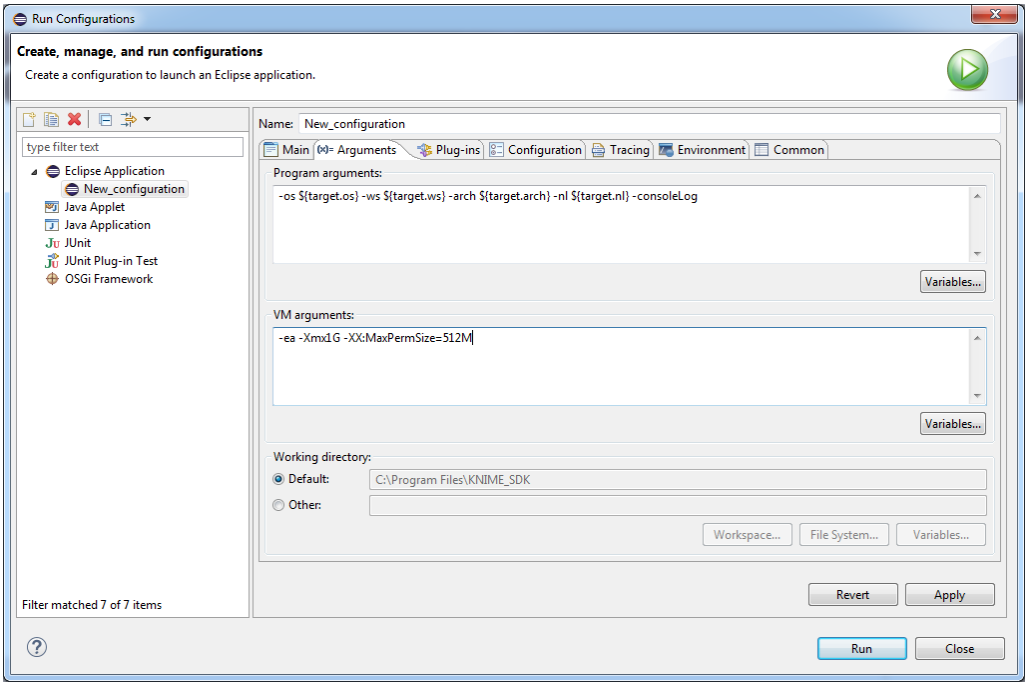
Obrázek 4.1: Ukázka workspace

4.1 Konfigurace KNIME SDK

Tato práce popisuje postup, jakým si každý programátor může vytvořit rozšiřující uzel do prostředí KNIME. Aby programátor nemusel vytvářet JAR po každé provedené změně v kódu, umožňuje KNIME SDK spuštění KNIME prostředí s automatickým sestavení uzlu do workspace. Tento uzel bez jakéhokoliv kopírování do složky s KNIME, nebo instalací tohoto uzlu může okamžitě použít a otestovat navržený uzel, zda-li funguje tak jak je po něm požadováno.

4.1.1 Příprava prostředí

První věcí, která je potřeba udělat je vytvoření spouštěcího konfigurátoru, který je zodpovědný za spuštění nové instance KNIME s vytvořeným uzlem. Run configuration, ve kterém se bude provádět celá konfigurace. Cesta k tomuto dialogu je **Run → Run configuration**. Po kliknutí na toto tlačítko se zobrazí daný dialog. Dvojklikem na **Eclipse Application**. Tímto se vytvoří první Eclipse application configuration, ve které můžeme nakonfigurovat jak chceme uzel spustit a jakým způsobem chceme uzel testovat.
První záložkou zde je **main**, zde je potřeba nastavit **run an application**, které nastavíme hodnotu **org.knime.product.KNIME_APPLICATION**. Dalším důležitým polem zde je **execution environment**, ve kterém se nastavuje java prostředí na kterém KNIME aplikace poběží. Zde by měla být nastavena hodnota JavaSE-1.8(KNIME_SDK). Jedná se o klasickou Javu, která je rozšířena o knihovny od KNIME.
V záložce **argument** je potřeba nastavit **VM argument** (Virtual machine arí guments). Tyto argumenty se používají k nakonfigurování prostředí Java Virtual Machine (dále jen JVM). Je možné nakonfigurovat například velikost haldy, nebo celkové paměti, která bude pro JVM vyhrazena. Pro spuštění je doporučena tato konfigurace: *-ea -Xmx1G -XX:MaxPermSize=512M*



Obrázek 4.2: Ukazka konfigurace

4.2 Vytvoření nového projektu s KNIME uzlem

K vytvoření nového projektu je zapotřebí zvolit následující cestu **File → New → Others → KNIME → Create a newKNIME Node-Extension**. Toto je možné se dostat do dialogu ve kterém se zadávají základní informace o nově vytvářeném uzlu. Mezi základní udaje patří název projektu, název uzle a také typ uzlu který chceme vytvořit (jedná se pouze o grafické odlišení uzlů, na funkčnost toto nemá vliv). Typ můžeme zvolit z níže vypsaných typu uzlu, které jsou v KNIME podporovány. Když jsou tyto údaje nastaveny, KNIME SDK vytvoří nový projekt s před-vygenerovanými třídami a konfiguračními soubory. Tento uzel je možné ihned spustit a vyzkoušet.

4.3 Základní struktura projektu

Nově vytvořený projekt se skládá z package, ve kterém je před generována kostra KNIME uzlu a dalších konfiguračních souborů, které jsou důležité pro správné sestavení JAR souboru.
V tomto package jsou mimo Java souborů i xml, ve kterém jsou uloženy základní konfigurační prvky. Název tohoto xml souboru je NAZEV_UZLU.xml. Zde jsou informace o typu uzlu, o způsobu zobrazení ve workspace (jakou ikonu má mít tento uzel v KNIME). Dále jsou zde krátké popisky o tom, jakou má tento uzel funkčnost. Popsání vstupních a výstupných portů.
Java soubory, které byly vygenerovány do tohoto package budou dále jednotlivě podrobněji popsány v této kapitole. Zde je stručný popisek jednotlivých tříd:

- NodeDialog - Uživatelské rozhraní pro nastavení vstupních parametrů uzlu.
- NodeFactory -Vytvoření nových instancí tohoto uzlu do workspace.
- NodeModel - Hlavní logika uzlu. Zpracování vstupných dat a vygenerování nových výstupných dat.
- NodePlugin - Umožňuje zavedení nového uzlu do prostředí KNIME.
- NodeView- Zobrazení dat po spuštění uzlu.

4.4 Třída NodeDialog

Tato třída spravuje grafické uživatelské rozhraní pro nakonfigurování vstupných parametrů. Umožňuje filtrovat sloupce se kterýma bude pracovat. KNIME má připraveno několik základních komponent, které může programátor využívat. Tyto komponenty jsou vkládány jednotlivě do dialogu. Také je tu možnost seskupení komponent, které mají na starost sobě blízké vstupní parametry.
Tato třída rozšiřuje abstraktní třídu *DefaultNodeSettingsPane*. Všechny komponenty

se zde vkládají do konstruktoru třídy. Zde jsou vkládány všechny komponenty pomocí metody *addDialogComponent();*. Argumentem této metody je kterákoliv třída, která dědí z abstraktní třídy *DialogComponent*. Zde je krátký seznam těch, které dále ještě budeme používat:

- ComponentNumber(); Vložení vstupných parametrů formátu number.
- ComponentColumnFilter(); filtrováni sloupců s kterými se bude po spuštění pracovat.
- ComponentStringListSelection(); Zvolení řetězce z nabídky

Každá komponenta má dva vstupní parametry. První vstupním parametrem je Settings. Settings je využíváno k nastavení vlastností vybrané komponenty. Součástí Setting je také klíč, pod kterým je možné tuto komponentu dohledat a propojit s modelem daného uzlu. Mimo jiné umožňuje nastavení defaultních hodnot v případě, kdy uživatel nevloží žádnou hodnotu do komponenty. Každá komponenta zde, musí mít tento Settings. KNIME využívá Setting jako interface, přes který je možné načítat hodnotu do workspace a uložit je, nebo naopak při spuštění workspace tuto hodnotu zpět načíst.
Druhým atributem této metody je label (popisek). Viditelné pojmenování v dialogu komponenty. Na příklad pokud máme komponentu pro načtení rozsahu, v tomto případě je label nastaven na hodnotu „Zadejte rozsah:“
Poslední věcí o která zde bude zmíněna je metoda *createNewGroupe(String nameGroup)*. Tato metoda umožňuje se seskupit komponenty v celém dialogu. Příkladem využití této metody je v případě kdy programátor chce v jedné části dialogu filtrovat hodnoty se kterými bude pracovat. K této komponentě chce přidat komponentu, která bude uchovávat informaci o maximální a minimální hodnotě vstupní proměnné.
V druhé části budou seskupeny komponenty které jsou součástí vstupních parametrů pro algoritmus DBSCANu. Aby programátor mohl ukončit tuto skupinu musí využít metody *closeCurrentGroup();*. Tato metoda je definována v rodičovské třídě. Z důvodu přehlednosti by nebylo vhodné tyto informace dávat do stejné skupiny.

4.5 Třída NodeModel

Všechna aplikační logika je obsažena v této třídě. NodeModel dědí z abstraktní třídy NodeModel, která má několik abstraktní metod, které jsou zapotřebí implementovat. V konstruktoru této třídy se nachází pouze zavolání konstruktoru rodiče, tedy *super(a,b)*. Atribut *a* je počet vstupních portů pro tabulku a Atribut *b* je počet výstupních portů pro výstup. Jeden uzel může mít několik vstupů a výstupů. Defaultní nastavení je *super(1,1)*. Tedy jeden vstupní port a jeden výstupní port.
Mezi abstraktní metody, které je potřeba implementovat se řadí metoda *execute()*. Tato abstraktní metoda umožňuje spustit logiku celého uzlu, který bude vykonávat operace a předávat je na výstup vytvářeného uzlu. Metoda *execute()* má dva vstupní parametry. První parametrem je pole *BufferedDataTable[]*, pomocí kterého

jsou předány všechny vstupní data do vytvářeného uzlu. Na základě indexu je pak vybráno z jakého vstupního portu mají data být zpracovány.
Druhým parametrem této metody je třída ExecutionContext. Tato třída nese v sobě všechny informace o spuštění tohoto uzlu. Mimo tyto informace umožňuje vytvářet předběžný přehled o tom, v jakém stavu se daný uzel nachází. Umožňuje nastavit status bar na hodnotu od 0 do 1, tato hodnota je nadále prezentována jako procento a vykreslena na uzlu. Ve většině data miningových algortimů je velice obtížné zaručit že daný progres bar bude růst konstantní rychlostí kvůli protože na začátku běhu uzlu není jasný přesný počet kroků, které bude algortimus vykonávat.
Metoda *execute()* má návratový typ *BufferedDataTable[]*, který reprezentuje výstupní data tohoto uzlu. V tomto případě se znovu jedná o pole kvůli protože může být na výstupu tohoto uzlu více portů. Je důležité vrátit stejný počet tabulek který byl definován v konstruktoru této třídy, jinak by tato ne-konzistence způsobila selhání spuštění tohoto uzlu.
V seznamu níže jsou vidět abstraktní metody, které by měl každý programátor vytvářející nový uzel v KNIME implementovat.

- reset();
- saveSettings();
- loadValidatedSettingsFrom();
- validateSettings();
- loadInternals();
- saveInternals();

4.5.1 Metoda reset

Metoda Reset(); by měla vrátit stav uzlu po spuštění do výchozího základního stavu. Životní cyklu uzlu v KNIME se skládá ze 3 částí. Uzel může být buď právě vložen do workspace, spuštěný a dokončený.
Tato metoda může být provolána přímo z workspace, při kliknutí na daný uzel pravým tlačítkem myši se zobrazí kontextové menu, ve kterém je položka restart. Při provolání této metody však nedochází k nastavení výchozích hodnot v dialogu daného uzlu.

4.5.2 Metoda configure

Metoda configue je využita ke kontrole, zda-li je k dispozici uživatelské nastavení, zda-li je uzel vhodný pro strukturu příchozích data a také zda-li je možné tento uzel spustit. Pokud uzel lze být spuštěn, pak tato metoda vráti specTable výstupních dat. Pokud tato kontrola nemá být provedena je vráceno pole s null hodnotou.

4.5.3 Metoda saveSettings

Tato metoda je provolána v okamžiku kdy uživatel v dialogu uzlu, ve kterém jsou vyplněny vstupní parametry klikne na tlačítko Save nebo Apply. Pak tyto data jsou předány do metody *saveSettings()*, která zpracuje vstupní nastavení a uloží je do Settings instancím třídy NodeModel. Takto uložené hodnoty mohou být jednoduše získány pro přečtení vstupních parametrů vytvářeného uzlu.

4.5.4 Metoda loadValidatedSetting

Podobně jako předchozí metody, je i tato metoda využit a propojena s dialogem uzlu. V okamžiku kdy je spuštěn tento uzel je uložena konfigurace. Po ukončení programu KNIME jsou tyto data uložena ve workspace. Při spuštění programu jsou tyto data znovu načtena pro každý uzel jednotlivě. Pro validaci těchto dat je využita tato metoda.

4.5.5 Metoda validateSetting

Po vyplnění vstupných parametrů v dialogu uzlu je zavolána tato metoda pro validování dat. Kontroluje zdali vstupní parametry jsou v hodné pro tento model a vyhovují omezení, které programátor při vytváření dialogu vytvořil. Pro příklad mějme komponentu, která má vstupní parametr typu Integer a reprezentuje počet sloupců. Uživatele nastaví hodnotu na -1 a uzavře dialog. Všechna data jsou předána do modelu tohoto uzlu který začne validovat vstupní data. V Settings je nastavena minimální povolená hodnota 1 a maximální hodnota 10. Když do této validace přijde -1 je vyhozena vyjímka InvalidSettingsException, protože logika vyhodnotí špatnou vstupní proměnou. Pokud by bylo číslo nastaveno na hodnotu mezi 1 až 10 hodnota by prošla.

4.6 Třída NodePlugin

Prostředí KNIME je postaveno na jádře Eclipse, pro přidání nového pluginu do Eclipse je potřeba, aby třída dědila ze třídy org.eclipse.core.runtime.Plugin. V tomto případě je za nový plugin považován nový uzel, který bude programátorem vytvořen. Tato třída, která je součástí hlavní vygenerované struktury je zodpovědná za to, že bude nový uzel správně zaveden jako plugin do KNIME a bude ho možné spustit. Je doporučované, aby programátor tuto třídu nijak nemodifikoval z důvodu možného výskytu chyby a ne-konzistence. Tato třída je pak navázaná na plugin.xml. V okamžiku změny např. názvu třídy nesmí být zapomenuto změnit cestu k této třídě i v plugin.xml.

4.7 Třída NodeView

Pro každý uzel, který je vložen do workspace je možné zobrazit jeho ViewDialog ve kterém jsou zobrazena data, která logika tohoto uzlu právě zpracovala nebo se jedná o pouhou funkčnost vizualizace dat.
Tento NodeView dialog lze vyvolat pravým kliknutím na tento uzle ve workspace a vybráním možnosti Show dialog View.
Programátor v této třídě, která dědí z abstraktní třídy NodeView, může využít tří událostí, na které může po té reagovat:

- modelChange() - Tato metoda je zavolána v okamžiku, kdy se zavolá logika uzlu a jsou změněna data na vstupu daného uzlu.
- onOpen() - Metoda provolaná na událost otevření dialogu s View. Právě do této metody může programátor vložit své View.
- onClose() - Metoda zavolaná v okamžiku kdy View bylo zavřeno.

Všechno grafické rozhraní je vytvořené pomocí Java Swing. Nástupcem Java Swing je JavaFX, která dovoluje programátorovi využívat modernějšího vzhledu, zobrazování dialogu s podporou HTML nebo CSS zde není dostupná. Existují možnosti, jak použít dialog, který je naprogramovaný za pomoci Java FX a spuštěný v threadu, který obsluhuje java Swing. V takovém případě si ale musí programátor pohlídat všechny závislosti na balíčky. Častým případem špatných závislostí a špatného sestavení balíčku do uzelu je spuštěný sestaveného uzlu v KNIME workspace, kdy tento uzel padá na chybu a kdy KNIME nemůže najít závislosti na další balíčky. V takovém případě je vyhozena vyjímka NoClassFoundException, tzn. vyžadovaná třída nebyla nalezena.

4.8 Třída NodeFactory

Tato třída využívá návrhového vzoru faktory. Pomáhá při vytváření nových instancí uzlu do workspace. Mimo to umožňuje nastavovat dodatečné konfigurační parametry. Příkladem takového parametru je v metodě hasDialog(). Metoda hasDialog() pak může být programátorem využita v okamžiku, kdy programátor vytváří nový uzel, který nepotřebuje mít dialog pro vkládání vstupních proměnných.

4.8.1 Návrhový vzor Factory

Návrhový vzor Factory je v programování používán k vytváření instancí třídy, pro které ovšem platí že nestačí pro správnou konfiguraci této třídy zavolat pouze konstruktor, ale musí být do settovány další hodnoty. Například mějme třídu Car, která má atribut brand. Pro vytvoření nové instance auta je však potřeba donastavovat hodnotu brand. Třída Faktory pak pomáhá programátorů připravovat instance tříd. Další výhodou návrhového vzoru je především přehlednost a čistota kódu.

4.9 Základní práce s daty

4.9.1 Čtení ze vstupu

Čtení z tabulky, která je přivedena na vstup uzlu je velice podobný práci s daty v poli. I zde je potřeba znát na jakém indexu se hodnota - kterou se programátor pokouší načíst - nachází. Každá vstupní tabulka je uložena v meta souborech workspace. V okamžiku kdy programátor chce data začít číst ze vstupu je tento soubor načten do paměti.
Programátor, který chce pracovat se vstupní tabulkou na vstupu bude pracovat s třídou BufferedDataTable. Z této třídy se dají zjistit informace jaké sloupce vstupní tabulka má a o jaký typ se jedná. Tato třída implementuje rozhraní Iterable<>. Toto rozhraní v jazyce Java umožňuje využívat třídu, která toto rozhraní implementuje k tomu, aby byla zdrojem dat v cyklu foreach.
Obecný pattern Iterable je využíván ve více programátorských jazycích. Skládá se ze dvou základních metod.

- hasNext() - Metoda která vrací True v případě, že v kolekci je další objekt, který může být vrácen. False když žádný další objekt už není.
- next() - Vrací další objekt z kolekce, který je další na řadě.

Obecně platí pravidlo kdy během iterace nesmí být změněn počet iterovaných objektů. Každý řádek v tabulce je reprezentován třídou DataRow, která v sobě dále uchovává informace o jednotlivých buňkách. Pro získání každého řádku je využíván právě cyklus foreach.
Třída DataRow,zapouzdře všechny buňky (cell) řádku. Pro přístup k datům uložených například v sloupci GPS_X je potřeba znát index tohoto sloupce ve specTable. Pak na základě znalosti této informace může být získaná buňka na daném index z řádku převedena na požadovaný formát.

```
BufferedDataTable data = inData[0];
int index = data.getDataTableSpec().findColumnIndex("GPS_X");
```

Obrázek 4.3: Získání tabulky a vyhledání indexu názvu sloupce.

Na obrázku je znázorněn příklad, vyhledání index podle názvu sloupce. Pokud je známý index sloupce, pak je možné přečíst hodnotu z buňky řádku. Nyní můžeme vzít proměnou z obrázku 2 dataRow.getCell(index).toString(); Takto lze získat String hodnotu z vybrané buňky. Když programátor ví o který se jedná typ, je možné tento Cell přetypovat na DoubleCell, IntCell nebo StringCell. Následně je možné získat hodnotu cell.

```
DBSCAN scanner = new DBSCAN(atribu.size());

BufferedDataTable data = inData[0]; //Vybraní tabulky
for (DataRow dataRow : data) {
    double[] array = getPoint(dataRow,atribu); //funkce, která vrátí GPS_X a GPS_Y v poli typu Double
    scanner.addNewNpoint(getNpoint(array,dataRow));//Přidá tento bod do seznamu v DBSCANU
}
```

Obrázek 4.4: Ukázka načítaní záznamů a vkládání do DBSCANu

4.9.2 Zápis dat na výstup

Výpis dat na výstupu je rozdělen do tří částí':

- Vytvoření spectTable - Vytvoření sloupců a definování daného typu sloupce pro buňku.
- Vytvoření řádku - Řádek je vytvořen v okamžiku kdy jsou přiřazené dané buňky do řádku.
- Vložení řádku do tabulky - Při vložení řádku do tabulky je potřebné vytvořit i klíč řádku. Nejčastěji se jedná o index v rozsahu od 0 do N, kde N je počet řádků výstupní tabulky.

Definice sloupců tabulky

Před vytvořením sloupců, by si měl každý programátor rozmyslet, jak by měla vypadat výstupní tabulka a jaké mám mít daný sloupec proměnné. Prvním krokem pro vytvoření sloupců je definování jednorozměrného pole DataSpectColumn[] o velikost N, kde N je počet sloupců. Dále vyplnit pole instancemi DataColumnSpecCreator, kterémi jsou 2 vstupní parametry. První parametr je název sloupce typu String. Druhý parametr je datový typ sloupce. Datovým typem sloupce můžeme být pouze jeden z těchto typů.

- StringCell.TYPE, pro datový typ String
- IntCell.TYPE, pro datový typ integer
- DoubleCell.TYPE, pro datový typ double

```
139
140         DataColumnSpec[] allColSpecs = new DataColumnSpec[1];
141
142         allColSpecs[0] = new DataColumnSpecCreator("Cluster", StringCell.TYPE).createSpec();
143
```

Obrázek 4.5: Definování sloupce.

Vytvoření nového řádku

Vytvoření jednoho řádku probíhá podobně jako v případě vytvoření sloupců tabulky. Je potřeba alokovat pole DataCell o velikosti N, kde N je jako v minulém případě počet sloupců. Nemůžeme se stát případ kdy by bylo méně sloupců než buněk v řádku a naopak. Tímto si vytvoříme datové buňky, do kterých budeme dávat data. Následně si musíme vytvořit identifikátor řádků. Pro tento případ tu je třída KeyRow, jejímž vstupním atributem je String. Když jsou připravené datové buňky (DataCell), pak jsou pomalu vkládány data. Data musíme dávat v takovém pořadí, v jakém byly definované sloupce. To samé platí také pro datový typ.
Pro String tvoříme instanci StringCell, pro Integer IntCell a pro Double DoubleCell.

Vložení řádku do tabulky

Předposlední krok je spojení klíče (identifikátoru) a datových buněk čímž vznikne jeden celistvý řádek. Pro dosažení spojení RowKey a buněk slouží třída DefaulRow, která vytváří řádek, kterého je možné přidat do tabulky. Tento řádek je vložen do tabulky. Všechny řádky se pak vkládají do třídy BufferedDataContainer.

5 Vytváření rozšiřujících balíčků

Tato kapitola popisuje postup vytvářených uzlu do KNIME prostředí a ukazuje postupný postup jakým byly tyto uzly vytvářeny. V některých případech byly vytvořeny úpravy, které pomáhají uživateli s určováním parametrů v reálném prostředí. Například pro DBSCANu byl zvolen přepočet parametru epsilon na jednotky kilometrů aby spíše odpovídaly reálným podmínkám při vzniku dopravních nehod.

5.1 KNIME User Adapter

Během vytváření rozšiřujících balíčků pro data miningový nástroj KNIME se autor této práce setkal s častým opakováním kódu při vytváření výstupní tabulky. Pro výstupní tabulku je potřebné definovat její SpectTable tj. seznam sloupců s daným typem, které bude výstupní tabulka mít. Dále je potřeba z výstupních dat vytvořit jednotlivé buňky, jim nastavit hodnotu a ty spojit do řádku, která bude přidána do výstupní tabulky.
Pro ulehčení této práce byl vytvořen Knime User Adapter framework, který za programátora řeší velkou část nepříjemných úkolů a zjednodušuje čitelnost kódu. Celá myšlenka tohoto frameworku je založena na anotovaní metod a skenování tříd pomocí java reflexe, z toho nemusí programátor implementovat žádné interface ani dědit abstraktní třídy. Je dokonce jedno jak se metody jmenují vše je zjištěno až za běhu programu. Jediná věc na kterou programátor musí myslet je, zda-li o anotoval metody.

5.1.1 Základní funkčnost

K použití tohoto frameworku je potřeba vytvořit dvě základní třídy. V první třídě - libovolně nazvané - je řešena logika. Třída s logikou má více možností jak být spuštěna. Toto rozhoduje programátor podle toho o jaký rozšiřující uzel se jedná. První možností spuštění je ForEachRow, tj. že logika se vztahuje pouze na jeden řádek v tabulce. Tato varianta je výhodná pro vytvoření uzlu, který filtruje řádky. Druhou možností je načtení potřebných dat a následné vrácení výsledků.
Druhá třída popisuje strukturu výstupních dat. Při spuštění uzlu se vytvoří instance třídy s logikou tato instance je předána do třídy KnimeUserAdapter. Zde je třída obsahující logiku pro-skenována pomocí java reflexe. KnimeUserAdapter nalezne potřebné metody, které jsou o anotovány. Na základě toho které metody nalezne se dále rozhoduje který z módů bude spuštěn.

5.1.2 Základní anotace pro třídu obsahující logiku uzlu

Tato sekce popisuje anotace, které jsou použity pro anotovaní metody ve třídě řešící logiku uzlu. Každá anotace, která je zde uvedena je určena pro anotovaní metody.

Anotace @BeforeStart

Anotace BeforeStart může být ve třídě obsahující logiku na více než jedné metodě nebo se zde nemusí nacházet vůbec. Slouží k preprocesingu dat. Pokud je o anotována metoda takto, pak daná metoda musí mít vstupní parametr DataRow row. V okamžiku spuštění jsou nejdříve provolány všechny metody které jsou takto o anotovány. Tato anotace obsahuje atribut order pomocí kterého programátor může určit pořadí provolání takto o anotovaných metod.

Anotace @Order

Tato anotace umožňuje určit pořadí v jakém se budou vykonávat metody o anotované @BeforeStart.

Anotace @ForEachRow

Tato anotace slouží k provolání logiky pro jednotlivé řádky. V případě že v dané třídě neexistuje takto o anotovaná metoda pak je využit druhý móde spuštění přes MainAlgorithm. Metoda která má tuto anotaci musí mít jako vstupní parametr DataRow row, pro získání řádku který bude vyhodnocovat.
Metoda, která má tuto anotaci nesmí být typu Void, ale musí vracet třídu, která tvoří řádek tabulky (výslednou třídu). Před spuštěním módu forEachRow je nahlédnuto na tuto metodu a zjištěno jaký má tato metoda návratový typ. Pak na základě tohoto návratového typu je vygenerován SpecTable (hlavička tabulky) a předupraveny typy buněk aby bylo možné pak vzít jenom výsledek a dosadit hodnoty.

Anotace @MainAlgorithm

Pokud je nad metodou uvedena tato anotace jedná se o metodu, která vykonává hlavní algoritmus.
Některé algoritmy potřebují nejdříve znát všechna data a na základě nich dokáží vygenerovat výsledky. Příkladem takového algoritmu je právě DBSCANu či APRIORI.
Při použití této metody musí být i definována i o anotována metoda, která vrací výsledky. Metoda musí být označena touto anotací @Results.

Anotace @Results

Tato anotace označuje metodu, která vrací výsledek algoritmu. Return type musí být vždy typu seznam (List<?>), ve kterém jsou uloženy data, které jsou výsledného

typu.
Po vrácení seznamu je vybrán první objekt toho seznamu a z něho je vygenerována hlavička tabulky. Pak jsou předupraveny typy buněk do tabulky.

5.1.3 Základní anotace pro třídu s výsledkem

Pro vygenerování výstupní tabulky je možné použít jakoukoliv POJO třídu (Plain old java object), která se bude skládat pouze z instančních proměnných a jejich getterů a setterů. Na základě počtu proměnných dané výstupní třídy je pak vygenerován odpovídající počet sloupců.
Jedna instance této výstupní třídy reprezentuje jeden řádek v tabulce. Jedna proměnná v této třídě je sloupcem v této tabulce. V případě, kdy proměná nemá nastavený getter nebo setter není počítána jako sloupec tabulky.

Anotace @Extended

V okamžiku kdy programátor chce pouze rozšířit vstupní tabulku o daný počet sloupců a dat, je možné využít tuto anotaci @Extended. Tato anotace okopíruje si uloží vstupní data se kterými chce pracovat vykoná potřebné operace a při zápis je znovu připojí.
Pokud je třída ve které je obsažena logika využívána v módu ForEachRow, je k danému řádku přidána výstupní třída. V druhém případě může dojít k tomu, že jsou nejdříve potřeba načíst data, se kterými se bude pracovat, z toho důvodu není možné ke každému řádku jenom přidat hodnotu. V tomto případě je nutné si daný řádek uchovat ve výstupní třídě. Pak následně je tento řádek načten při výpisu dat do tabulky jako první a jsou k němu přidány atributy z výstupní třídy.

Anotace @Column

Tato anotace umožňuje programátorovi pojmenovat sloupec pomocí vstupního parametru této anotace. Tato anotace se nachází nad proměnou. V případě, kdy nad touto proměnou není anotace Column je defaultně vybrán název proměnné a ten je dosazený jako název sloupce. Také může dojít k případu, kdy programátor má ve vstupních datech pojmenovaný sloupec MESTO a v této výstupní třídě chce pojmenovat proměnou MESTO, v takovém okamžiku by vznikly 2 sloupce se stejným názvem, které KNIME neumožňuje mít a dojde k chybě.

Anotace @Ignore

Někdy může dojít k situaci, kdy proměnná ve výstupní třídě nemá být na výstupu, z tohoto důvodu vznikla anotace @Ignore, která zamezuje tomu, aby takto o anotovaná proměnná byla vypsána.

Anotace @Row

Pro označení proměnné jako řádku vstupních dat, který má být rozšířen právě touto třídou je využita anotace @Row, která upozorňuje Knime User Adapter na přítomnost proměnné typu DataRow ve výstupní třídě a pro výstup je s tímto řádkem nadále počítáno.

```
private DBSCAN scanner = new DBSCAN(2);

@BeforeStart
public void forEachRow(DataRow row) {
    double[] array = getPoint(row, atributs);
    scanner.addNewNpoint(getNpoint(array, row));
}

@MainAlgorithm
public void start() throws CanceledExecutionException {
    scanner.dbscan(eps, pointCount, exec);
}

@Results
public List<Npoint> result() {
    return scanner.getList();
}
```

Obrázek 5.1: Ukázka využití Knime User Adapteru pro vytvoření uzlu užívajícího algoritmus DBSCAN.

5.2 DBSCAN

Tvoření vlastního uzlu, který bude využívat algoritmu DBSCAN bylo nutné správně vytvořit logiku tohoto uzlu ve třídě NodeModel a vytvoření dialogu k zadání vstupních parametrů ke konfiguraci tohoto uzlu.
Tento uzel slouží k vyhledávání dopravních nehod na základě GPS souřadnic. I přesto algoritmus DBSCAN nemusí vždy fungovat jenom na množině 2D pro tento příklad tomu tak je. Vstupními parametry toho algoritmu je parametr epsilon nebo také rozsah (range) určující vzdálenost od nehod, ve které by mělo být více nebo N dopravních nehod. Druhý parametr je minimální počet bodů (dále jen minPts). Tento parametr určuje, jaký je minimální počet nehod, které musí být v okolí dané nehody aby se dalo říci o nehodě že je součástí shluku. Pro zjednodušení bylo do tohoto uzlu přidán přepočet decimální stupňů na poloměr v metrech. To umožňuje uživateli uvažovat o rozsahu jako o vzdálenost místa nehody, která je v jednotkách metrů. Protože může dojít k většímu nebo menšímu rozsahu byly přidány jednotky od Km až po mm.

5.2.1 Převod jednotek

Tabulka 5.1: Tabulka převodu vzdáleností

	Decimal degrees	DMS	Porovnání	Vzdálenost
0	1.0	1° 00 0	Státy nebo větší regiony	111.32 km
1	0.1	0° 06 0	Větší města	11.132 km
2	0.01	0° 00 36	Města a vesnice	1.1132 km
3	0.001	0° 00 3.6	Sousedství, ulice	111.32 m
4	0.0001	0° 00 0.36	Ulice, parcela	11.132 m
5	0.00001	0° 00 0.036	Strom	1.1132 m
6	0.000001	0° 00 0.0036	Člověk	111.32 mm

V tomto případě hodnota 1.0 ve sloupci Decimal degrees je poloměr v oblasti ve kterém jsou vyhledávány dopravní nehody. Pro přepočet těchto jednotek kilometrů je využito přímé úměrnosti. Hodnota range která je zadána na vstupu tohoto uzlu je pak násobena konstantou 1.0 / 111.32. [17]

5.2.2 Vytvoření konfiguračního dialogu

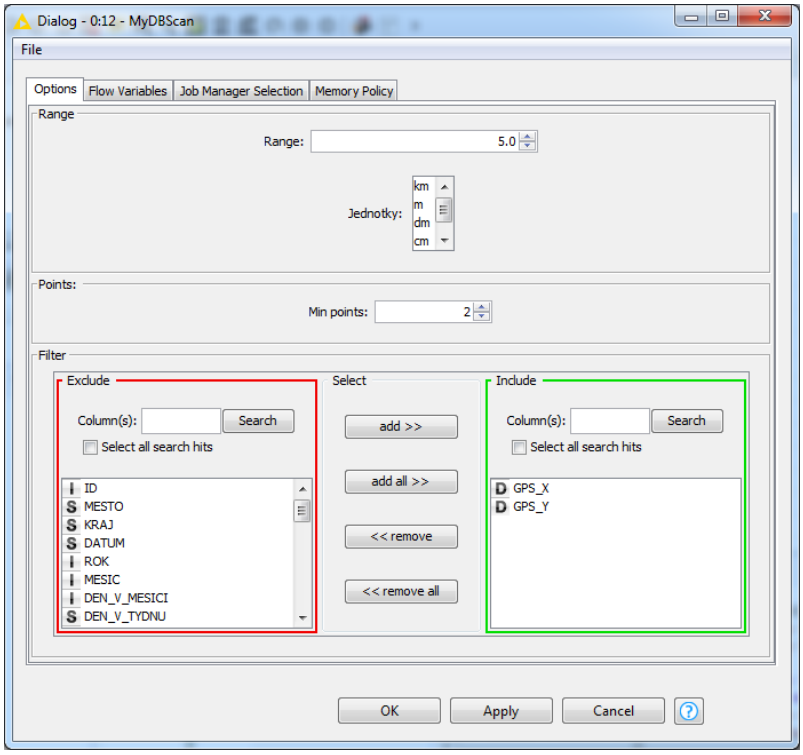
Vytvoření dialogu pro uzel DBSCAN bude sloužit pro komunikaci s uživatelem. V tomto dialogu budou vytvořeny tři skupiny komponent.
První skupina komponent je vstupní parametr range s jednotkami. Druhou skupinu tvoří pouze jedna komponenta s minimálním počtem bodů. Třetí skupina s komponenta s filtrováním sloupců.

```
createNewGroup("Range");
SettingsModelDoubleBounded range_settings = new SettingsModelDoubleBounded(
        "range", // Pojmenování proměnné
        0.1,// Defaulní nastavení
        Double.MIN_NORMAL,// Minimální hodnota
        Double.MAX_VALUE// Maximální hodnota
        );
DialogComponentNumber component_range = new DialogComponentNumber(
        range_settings, // vložení nastavení
        "Range:"// label
        ,0.1 // Výchozí hodnota
        );
addDialogComponent(component_range);
```

Obrázek 5.2: Ukázka kódu pro tvorbu dialogu.

Způsobem, který je uveden na obrázku jsou připraveny další komponenty. Pro výběr jednotek bylo potřeba vytvořit pole typu String. Zadat jednotky z kterých

bude možné si vybírat. U vytváření komponenty výběr sloupce je využitá komponenta DialogComponentColumnFilter, který má setting model SettingsModelFilterString(String name), jako atribut je zvolen název *filterColumn*, pod tímto názvem je pak dohledána pro NodeModel.



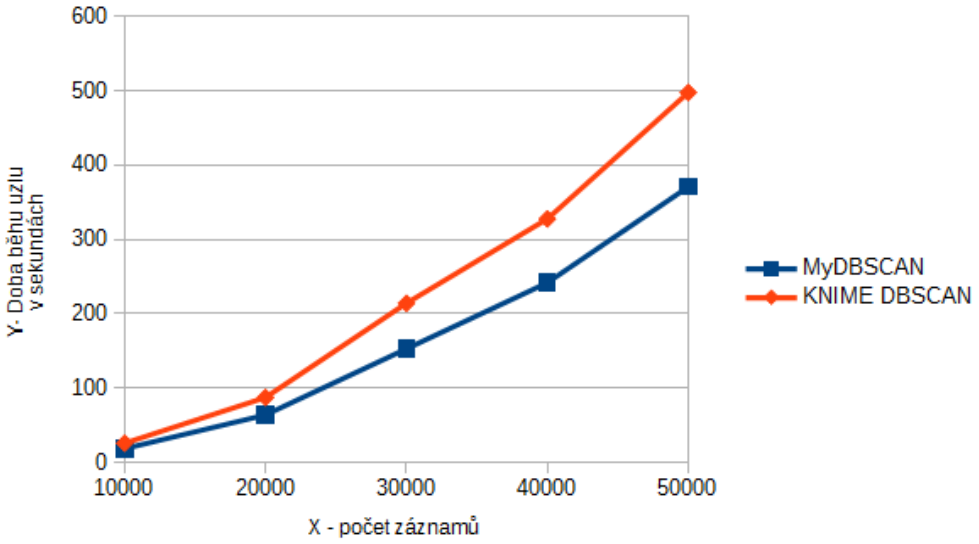
Obrázek 5.3: Výsledný dialog pro DBSCAN v KNIME.

5.2.3 Programování ve třídě NodeModel

V této třídě bylo nutné získat všechny vstupní parametry, které se po spuštění uzlu předají do logiky. Pro získání hodnoty typu Double, bylo získáno setting, které tuto hodnotu reprezentovalo ve vstupním dialogu uzlu. Po získání setting byla provolána metoda getValue(), pomocí které byla získána hodnota pro range. Stejným způsobem je řešeno i získání ostatních hodnot elementárního typu, kterými jsou hodnoty Integer a String.
Setting pro filtr, který byl přidán do dialogu má dvě metody, které může programátor použít. Obě metody vracejí List<String>, String v tomto případě bude název sloupce, který se nemá použít nebo název sloupce, který by se měl použít.
Po získání indexu sloupců, na kterých se nachází sloupce se kterými bude DBSCANu pracovat nezbývá než pro každou řádku v tabulce vybrat tyto sloupce z nich vytvořit bod a uložit si zbylá data řádku pro předání na výstup. Po spuštění a doběhnutí algoritmu jsou data předána na výstup. Na výstupu tohoto uzlu je pak stejná tabulka, která byla na vstupu se sloupcem určující určitý shluku.

5.2.4 Porovnání uzlů

Při porovnávání uzlu, který je součástí KNIME a tohoto uzlu bylo již na první pohled zřejmé, že takto vytvořený uzel je rychlejší, než uzel, který je vytvořený KNIME, i sám KNIME upozorňuje, že tento uzel není optimální pro práci větším množstvím dat.
Bylo provedeno měření a porovnání těchto uzlů. Porovnával se především počet záznamů na vstupu uzlu a čas který byl potřeba k dokončení běhu algoritmu.
Jedním z problémů, který se objevil byla konfigurace DBSCANu od KNIME. Tento uzel dovoluje zpracovávat data která jsou stranou ukládána, to umožňuje KNIME tím byla způsobena mnohem delší doba potřebná k běhu tohoto algoritmu. Využití operační paměti je v takovém případě mnohem nižší ale je běh algoritmu je velice pomalý. Další možností bylo nastavit uzel aby byl schopný načíst všechny data přímo do paměti. Tuto volbu jsme zvolili aby bylo možné porovnaná oba uzly, protože na stejném způsobu pracuje i uzel DBSCAN vytvoření v této bakalářské práci.



Obrázek 5.4: Porovnání uzlů

Na grafu je vidět porovnání obou uzlů. Z grafu je patrná časová náročnost výpočtu pro velké množství dat. Na ose X je vyobrazen počet záznamů, které byly přivedeny na vstupu uzlu. Na ose Y je hodnota času v sekundách, které ukazují dobu běhu uzlu po spuštění s vybraným počtem záznamů.

5.3 OPTICS

Podobně jako u uzlu který využívá algoritmu DBSCAN je vytvořen dialog, který obsahuje 3 komponenty. První komponentou je oblast ve které se musí nacházet všechny sousedské body. Druhou komponentou je minimální počet sousedských bodu

aby mohl být uzel požadován za shluk.
Tento uzel byl naprogramovává tak aby každou jeho část bylo možná využít i na jiném místě. Pro uzel využívající algoritmu OPTICS byly tvořeny další tři pomocné uzly, které budou popsány níže.
Výhodou tohoto algoritmu je především možnost vyrovnání se s kolísavou hustotu. Z toho důvodu byl vytvořen hlavní uzel OPTICS, který na výstupu přidává dva sloupce. Jedná se o sloupce CoreDistance a distance.
Pro zobrazení těchto uzlů a rozhodnutí pomocí jaké limitní hodnoty pak budou dané body přiřazeny do shluku je využit uzel OPTICSView. Po zvolení limitního kritéria k vytvoření shluku je využit uzel OPTICSCluster, který dané body zařadí do shluku. Posledním uzel, který kombinuje hlavní algoritmu OPTICS, OPTICSView a OPTICSCluster je OPTICSWithView.

5.3.1 Uzel OPTICS

Při spuštění tohoto uzlu jsou načteny všechny potřebné vstupní paramerty. Po validaci vstupních parametrů je spuštěn hlavní algorimus, který pro každý bod zjistí CoreDistance i distance. Všechny řádky jsou pak setříděny podle pořadí které jim algoritmus OPTICS přidal. Ke každé řádce jsou přidány dva hlavní sloupce. Tento uzel slouží pouze k vygenerování distance. Pro další zpracovaní jsou využity další pomocné uzly.

5.3.2 Pomocné uzly

Tyto pomocné uzly byly vytvořeny z důvodu možnosti znova vygenerování shluků v případě kdy OPTICS vrátí coreDistance, ale je na poprvé zvolena příliš velká hodnota limitního kritéria, podle kterého se pak tyto uzly rozdělují do shluků. Tímto je umožněno jednou spustit uzel s OPTICS, který může běžet delší dobu. OPTICSView je možné využít k zobrazení dendrogramu. Po vizualizaci dat v OPTICSView je možné určit lépe limitní kriterium. V případě špatně zvoleného limitního kritéria je možné se znovu kouknout na dendrogram a určit lepší hodnotu. OPTICSCluster následně na základě tohoto limitního kritéria roztřídí body do shluku.

OPTICSView

Uzel umožnující zobrazení dendrogram po vypočítání coreDistance. Pro tento uzel byl naprogramován graf, ve kterém je možné pohybovat s limitním kritériem. Po změně hodnoty limitního kritéria jsou graficky znázorněny shluku, které mohou po zvolení daného limitu být vytvořeny.

OPTICSCluster

Na základě vstupní hodnoty, kterou zadává uživatel, umožňuje přiřadit hodnotu shluku pro všechny body. Důležitou podmínkou pro použití tohoto uzlu je mít sloupce coreDistance a distance z hlavního uzlu OPTICS.

OPTICSWithView

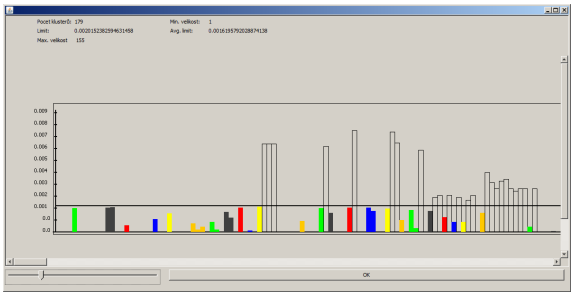
Toto uzel kombinuje všechny předchozí uzly. Umožňuje uživateli spustit všechny operace jako je zobrazení dat v dendrogramu a přiřazení dat do shluku během jednoho běhu tohoto uzlu. Po spuštění tohoto uzlu je spuštěn algoritmus OPTICS, který vygeneruje coreDistance a distance. Následně je celá metoda execute() zastavena. Po zastavení je zobrazen uživateli dendrogram, ve kterém je možné zvolit hodnotu limitního kritéria. Po zvolení této hodny a kliknutí na tlačítko OK je automaticky přezvána hodnota limitního kritéria do metody execute(), která pokračuje zařazovat jednotlivé body do shluku.

5.4 APRIORI

V této sekci je popsán postup, který byl aplikován při vytvoření nového uzlu, který využívá algoritmu APRIORI. Tyto podsekce jsou zkrácené z důvodu častého opakování stejných nebo velice podobných postupů.

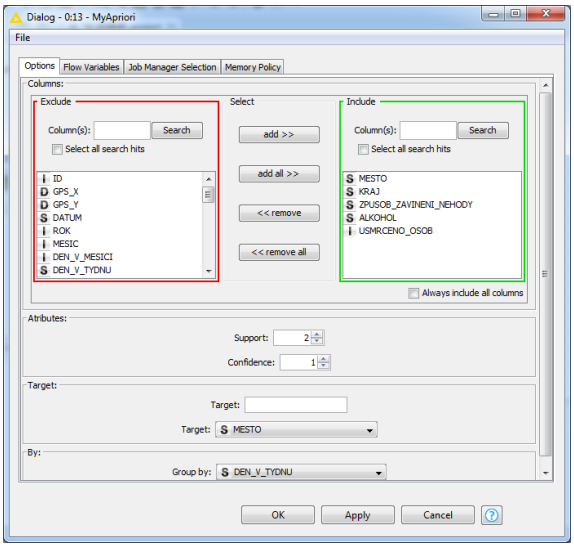
5.4.1 Tvorba NodeDialogu pro APRIORI

Vstupní parametry pro tento uzel jsou především support a confidence, které jsou důležité pro algoritmus APRIORI. Mimo tyto dva vstupní parametry byly vybrány další. Filtrování sloupců ze kterých jsou následně vygenerovány asociační pravidla. Vybraní sloupce podle kterého budou hodnoty seskupeny. Poslední vstupním parametrem je parametr Target, který umožňuje vyfiltrovat vygenerovaná asociační pravidla podle hodnoty, která je zadá zde.



Obrázek 5.5: Dialog pro vybrání limity podle které budou následně body zařazeny do shluku.

Vylepšením pro tento uzel je možnost group by. Toto vylepšení je přizpůsobené funkčnosti která je známá. Získané oblastni (shluky) můžou být ve větším počtu. Pro tyto shluky je použit algoritmus APRIORI. V jiných data miningových prostředích jako je například IBM Modeler by bylo nutné každý shluk vyfiltrovat a aplikovat manuálně algoritmus Apriori. Vytvořením této funkčnosti je uživateli umožněno rychlého zpracovaní informací o daném shluku.



Obrázek 5.6: Výsledný dialog pro Apriori v KNIME

5.4.2 Hlavní kroky v NodeModelu

V této třídě je postupováno velice podobně jako v předešlých uzlých.

- Získání vstupních proměnných z dialogu
- Zpracování dat ze vstupu
- Vytvoření skupin dat podle proměné group by
- Spuštění APRIORI pro každou skupinu
- Zapsání výsledků na výstup

5.5 DensityFinder

5.5.1 Motivace

Hlavní motivací pro vytvoření tohoto uzlu byla možnost přidání dat o hustotě dopravy na základě GPS souřadnic. Tato hustota se pak může využívat pro správné nastavení shlukující algoritmů.
Tento uzel komunikuje na API vystavené Ředitelstvím silnic a dálnic. Vstupním parametrem pro zavolání tohoto API jsou JTSK souřadnice, z tohoto důvodu bylo do uzlu nutné přidat funkci na převod z GPS souřadnic na souřadnice JTSK. Souřadnicový systém JTSK využívá souřadnicovou síť, která pokrývá Českou republiku a Slovensko.
Po převodu jednotek z GPS na JTSK je možné se dotázat na API zda-li v oblasti tohoto bodu existuje kominikace na které je měřena intenzita dopravy. Toto API vrací seznam komunikací i s ID vybrané komunikace, pomocí kterého se nadále je

možné dotázat na podrobnější informace. Někdy může nastat případ, kdy příjde odpověď ze serveru, ve které bude více komunikaci nalezených (např. GPS souřadnice je uprostřed křižovatky).V tomto případě se využije JTSK souřadnic, které popisuji jednotlivé zalomení kominikace (jedná se o seznam JTSK bodů, používající se na frontendu pro vykreslení komunikace). Pomocí tohoto seznamu bodů a vybrané metriky je vypočítána vzdálenost (chybovost), následně je vybrán ten úsek který je nejblíž k vybranému bodu. Po získání nejbližšího úseku je využito ID úseku k dotázání na hustotu dopravy. Odpovědí na tento dotaz je pak text ve formátu HTML, ze kterého jsou zpracována data. Tyto data jsou následně přidána k právě zpracovávanému řádku s GPS souřadnicí.
V této bakalářské práci byly vytvořeny 2 uzly pro zjišťování hustoty:

- DensityFinder - Určený pro rychle vyhledání hustoty dopravy. Není vhodný pro větší data protože neumožňuje backup výsledků (tj. při vypnutí uzlu jsou data znovu stahována). Umožňuje konfiguraci pro jednu úroveň.
- DensityFinderWithCache - Určen pro velké množství dat. Umožňuje backup výsledků. Průběžně jsou data ukládána do CSV souboru. Při znovu spuštění jsou data načtena z CSV souboru. Umožňuje více úrovňovou konfiguraci tzn. v případě, že ve vybrané oblasti neexistuje hustota pro tento bod je hledávána ve větší vzdálenosti od tohoto bodu.

5.5.2 DensityFinder

DensityFinder je určený především pro menší dataset (5 000 záznamů). Tento uzel neni vhodný pro větší datasety protože v případě jakéhokoliv výpadku ulzu nebo restartování jsou tyto data stahovány znovu.
Vstupními parametry pro tento uzel jsou GPS souřadnice. Mezi dodatečné konfigurační parametry patří oblast znázorněna hodnotami zoom, scale a resolution. Tyto dodatečné parametry by měl uživatel správně vybrat. V případě kdy jsou hodnoty příliš malé nemusí dojít k nalezení bodu. V případě kdy tyto hodnoty jsou příliš vysoké budou staženy všechny oblasti, následně je v těchto oblastech vyhledáván úsek, který leží nejblíže k vyhledávanému bodu. V tomto okmažiku může dojít k dlouhému vyhledání jednoho bodu (může se jednat i několik desítek sekund). K tomuto uzlu byla přidána možnost vytvoření intervalu (delay) mezi jednotlivými dotazy.

5.5.3 DensityFinderWithCache

Při vytváření uzlu DensityFinder bylo zjištěno několik nedostatků, které tento uzel dokáže vyřešit. Mezi hlavní problémy patří problém nečekaného vypnutí uzlu. V okamžiku kdy je spuštěn tento uzel s datasetem který obsahuje velké množství záznamů může zpracování takového datasetu trvat několik desítek hodin. Kdyby došlo k nečekanému přerušení, tento uzel si dokáže poradit. Po zpracování jednoho řádků tabulky jsou data uložena ve formátu csv. Jeden řádek ve formátu csv je roven řádku ve zpracovávané tabulce. Při nečekaném vypnutí je pouze tento soubor uzavřen. Po znovu spuštění tohoto uzlu jsou sekvenčně načítána data z backup souboru csv. V

momentě kdy již v backup souboru se nenachází další řádky a je znovu dotazováno se na API, pak nově zpracované řádky jsou přidány na konec souboru. Výhodou tohoto řešení je také možnost přenositelnosti práce. Uživatel může vzít tento backup soubor zkopírovat ho na jiný počítat a pokračovat ve zpracovávání datasetu.
Další možností tohoto uzlu je možnost si cachovat výsledky různých úseků. Když je zpracováván úsek o kterém jsou již známy hodnoty neni potřeba se znovu doptávat API ale jsou brány hodnoty z cache.
Pro cache a backup bylo potřeba vyřešit problém s ukládáním dat, resp. s místem kam by tento uzel mohl ukládat svá data. V dialogu uzlu je nutné zvolit tzv. working space, tj. místo které je určeno pro odkládání nebo načítání konfiguračních souborů. Na rozdíl od DensityFinderu je zde možné vyhledávat v několika úrovních. V případě kdy neni záznam vyhledán v první úrovni je hledáno v další, když není nalezeno ani zde je oblast znovu zvětšena. Oblast ve které tento uzel hledá je nastavena v konfiguračním souboru zoomProperies.properties, který musí být součástí working space. V tomto konfiguračním souboru je možné konfigurovat velikosti oblastní ve kterých je hledáno. Maximální oblast se nazývá LEVEL_0. Minimální oblast se nazývá LEVEL_8. Každý řádek nese informaci ve které oblasti byl nalezen, podle toho zle vyfiltrovat body, které mají velikou přesnost hustoty.

5.5.4 Seznam dostupných výsledků

Po dotázání na API je vráceno HTML, které je rozpársováno. Rozparsované výsledky jsou přiloženy k danému řádku jako výstupní hodnota. Data která byla tímto způsobem staženy nesou 69 informací o hustotě dopravy ve vybraném úseku.
Mezi hlavní data, která jsou získávána patří:

- Roční průměrná denní intenzita dopravy - Zahrnuje rozdělení pro automobily, autobusy, nákladní auta nebo traktory
- Roční průměrná denní intenzita dopravy, která rozlišuje svátky a všední dny.
- Hodinová intenzita dopravy - Může být buď Padesáti rázová intenzita dopravy nebo Špičková hodinová intenzita dopravy.
- Těžká nákladní vozidla - v jednotkách vozidlo/den
- Intenzita dopravy rozdělena do časových úseků - Měřené úseky jsou od 06-18, 18-22, 22-06 hodin.
- Emise
- Koeficienty nerovnoměrnosti dopravy
- Intenzita cyklistické dopravy

5.6 MyFilter

Uzel MyFilter umožňuje filtrovat data z tabulky na základě dotazu. Dotaz má podobnou strukturu jako SQL dotaz. Pro tento případ ale nepotřebuje klíčová slova SELECT, FROM. Je zde využita pouze část WHERE, kde se nachází podmínka, která musí být splněna aby tyto data byly předány ze vstupu na výstup. V případě nesplněné podmínky jsou data ignorována a zahozena.
Hlavní motivací pro vytvoření tohoto uzlu je zjednodušení dosavadních filtrujících uzlů v tomto prostředí.

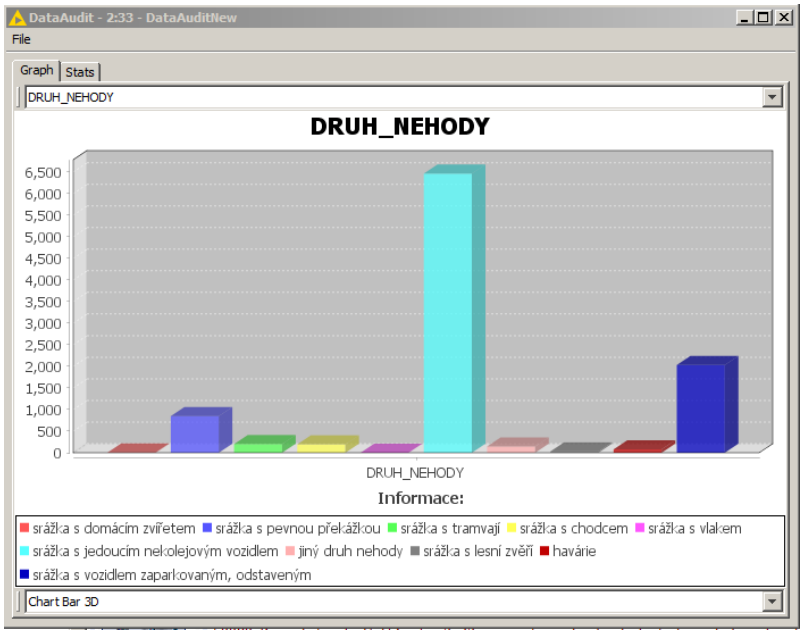
5.6.1 Vytvoření dialogu pro zadaní dotazu

Pro vytvoření dialogu k zadaní dotazu, který nadále bude rozpársován a vyhodnocen, bylo potřeba třeba naprogramovat vlastní komponentu i Settings. Dialog umožňuje programátorovi reagovat na události otevření dialogu a zavření. Událost pro otevření dialogu byla použita pro vložení vlastního dialogu, který byl naprogramován pomocí Java Swing.
V tomto dialogu je na levé straně zobrazen seznam všech operací, které jsou dostupné. Na pravé straně tohoto dialogu je seznam všech sloupců, které jsou dostupné ve vstupní tabulce. Další možností rozšíření bylo napovídání, která data se v daném sloupci vyskytují. Bohužel KNIME nedovoluje získat vstupní data, dokud není uzel spuštěn.
Po dvojkliku na uvedenou operaci nebo na uvedený název sloupce se tato operace/název sloupce připíše rovnou do dotazu. Dotaz má stejnou strukturu, která je známá z dotazu SQL. Kromě elementárních operací, tento uzel umožňuje i podmínky typu IN.

5.7 DataAudit

Data miningový uzel DataAudit je užitečným nástrojem pro vizualizaci dat a pro statistických údajů. Tento uzel v prostředí KNIME neexistuje, je zde podobný nástroj Statistix, který dokáže zobrazit statistická data jako jsou maximální hodnota, minimální hodnota, směrodatná odchylka nebo nejčastější hodnota. Tento uzel by vytvořen podle předlohy uzlu DataAudit z IBM Modeleru. Na rozdíl od uzlu, který vytvořeného v prostředí IBM Modeler, tento uzel umožňuje uživateli zobrazení dat v různých typech grafů.
K vizualizaci těchto dat byla využita knihovna JFreeChart.jar, která umožňuje zobrazovat data v grafech. Také umožňuje uživateli pracovat s vytvořeným grafem formou exportu obrázku do JPG, PNG formátu. Umožňuje přizpůsobení vzhledu grafu uživateli.
Uzel DataAudit při spuštění roztřídí data podle hodnot v daném sloupci a udržuje si informaci kolikrát se tato hodnota vyskytla pro daný sloupec. Problém se kterým se autor bakalářské práce setkal bylo zobrazení velkého počtu rozdílných dat v jednom grafu. Příkladem může být zobrazení grafu o GPS X souřadnicích. Protože

GPS souřadnice je typu Double je malá šance, že se budou shodovat tyto hodnoty. To způsobí že pro tento sloupec bude nalezeno velké množství rozdílných hodnot. Tento problém byl vyřešen limitem zobrazení rozdílných hodnot. Tzn. uživateli je zobrazeno právě 40 rozdílných hodnot. Toto číslo bylo zvoleno tak, aby bylo co největší ale přitom bylo ještě možné přečíst data, která jsou vyobrazena na grafu.



Obrázek 5.7: Zobrazení sloupce s počtem dopravních nehod v závislosti na povrchu vozovky

5.8 Přidání externích knihoven do KNIME projektu

Během vytváření rozšiřujících uzlů pro KNIME prostředí se autor této bakalářské práce setkal s problémem při využívání externích knihoven. Práce s externími knihovnami není úplně zřejmá na první pohled, protože v každém prostředí se chová rozdílně. Na příklad když je knihovna přidána do KNIME SDK prostředí není zde zřejmé, jestli bude tato knihovna připojena do buildu uzlu či naopak. KNIME SDK se chová na první pohled jako kdyby knihovna již byla přidána a je vše v pořádku, ale při otestování v emulovaném KNIME prostředí se při spuštění uzlu objeví zpravidla ClassNotFoundException, tzn. požadovaná třída nemohla být instanciována protože nebyla nalezena. Při prozkoumání buildu tohoto uzlu se následně zjistí, že externí knihovny nebyly do exportovaného JAR souboru přidány i přesto, že se KNIME SDK tváří jako že je vše bez chyby. Ačkoliv je na stránkách KNIME základní tutoriál, který ujasňuje základní věci, není zde zmíněna práce s externími knihovnami. Nejlepší způsob jakým pracovat s externími knihovnami je:

- Vytvoření složky /lib a nakopírování externích knihoven na toto místo

- Roz kliknutí souboru plugins.xml
- Přejí do záložky Runtime
- Do Classpath přidat složku /lib
- Přejít do Builds a zkontolovat že v Binary Build je zaškrtnuta složka /lib

5.9 Exportování uzlů z KNIME SDK

Vytvořené uzly v KNIME SDK, musí být nyní vyexportovány z tohoto prostředí do prostředí KNIME. K cílení správného exportu by měly být zkontrolovány všechny konfigurační parametry pro sestavení těchto uzlů. Pokud jsou využity některé z externích knihoven, musí být tyto knihovny přidány do konfiguračního souboru MANIFEST.MF. Mimo jiné zde je možné nakonfigurovat celou strukturu JAR souboru, který bude při exportu vytvořen.
Pro import nově vytvořených uzlů do KNIME prostředí bude stačit vytvořit z projektu JAR soubor. Zde je popsán správný postup podle kterého by měl programátor postupovat při exportování a sestavovaní JAR. Cesta, jak se dostat k exportu z KNIME SDK je File → Export..., po kliknutí se zobrazí dialog. Tento dialog umožňuje několik způsobů exportů projektu. Například exportování do projektu do archivu ZIP. Roz kliknutím položky Plug-in Development se zobrazí možnosti exportováni uzlu pro KNIME. Zde by mělo být zvoleno Deployable plug-ins and fregments. Následně se zobrazí nový dialog s parametry exportu a upřesněním kam tento JAR soubor exportovat. V dialogu se zobrazí projekty, které chceme exportovat. Pak zvolíme cestu kam budoucí plugins budou uloženy. KNIME SDK defaultně na uvedené místo vytvoří složku Plugins. V této složce se budou nacházet JAR soubory. Tyto soubory je pak potřeba nakopírovat do složky s nainstalovaným prostředím KNIME do složky plugins.
V případě, kdy programátor testuje tyto uzly sestavením a vložením do této složky, tak se vždy nemusí změny projevit. Při spuštění KNIME jsou cachovány tyto JAR, které budou použity. V momentě kdy programátor změní JAR v KNIME a znovu spustí prostředí změny, které byly provedené se nezmění. Aby byly změny provedeny musí být všechny pluginy, které byly vytvořeny programátorem odstraněny. KNIME prostředí musí být jednou spuštěno bez těchto pluginů a potom musí byt tyto pluginy zpět nakopírovány. Po těchto operacích se teprve změny znovu projeví

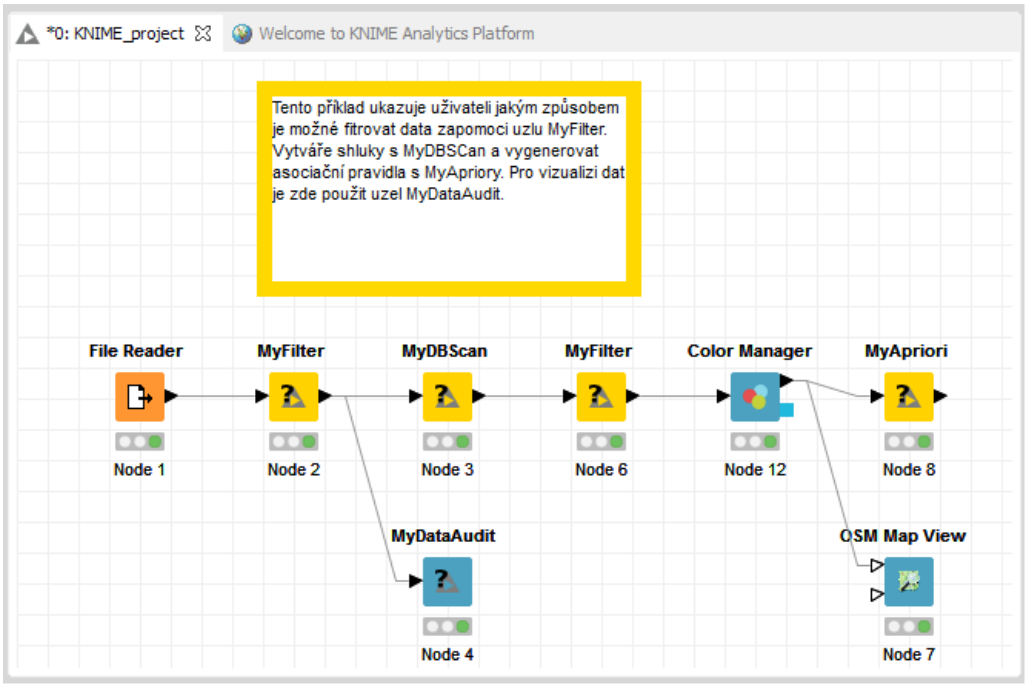
5.10 Spuštění nových uzlů v prostředí KNIME

Když jsou uzly vytvořené a exportované nastává fáze testování v čistém prostředí KNIME. Slovo čistém je zde myšleno tak, že se nejedná o žádné emulované prostředí. Když jsou uzly přidány do prostředí KNIME uživatel tyto nově vytvořené uzly může najít vlevo dole v menu. Ideální případ nastal když programátor vytvořil uzel, který nemá žádné chyby a dělá přesně to co je po uzlu požadováno.
Tento idealistický případ nepatří ale do reálného světa, ve většině případů se zjistí

že je potřeba tento uzel nějakým způsobem vylepšit. Problém nastává v okamžiku kdy chceme již nainstalovaný uzel pro určitý workspace změnit. Kdyby programátor udělal změnu v kódu který opravuje chybu následně udělal export tohoto uzlu a vložil do složky plugins, tak se tato změna pro právě používaný workspace neprojeví, i přesto že KNIME prostředí bylo vypnuto a zapnuto několikrát. Tato změna se projeví až v okamžiku kdy uživatel vytvoří nový workspace. Toto ale není chtěné, když uživatel má již vytvořené nějaké workflow a potřeboval jenom trošku vylepšit/optimalizovat uzel.
Problém je v KNIME, který užívá frameworku Eclipse OSGI, řešící modularitu aplikace. Nový plugin je nainstalován a už nejde pro určitý workspace odinstalovat.
Řešením jak tyto uzly odinstalovat je odstranění obsahu ve složce ${KNIME_HOME} /configuration/org.eclipse.osgi/.
Uvedená složka obsahuje nainstalované pluginy přes všechny workspaces. Když je obsah této složky smazán jsou všechny uzly znovu nainstalovány a tím se projeví i změny provedené při drobné změně uzlu.

6 Případová studie využití vytvořených uzlů

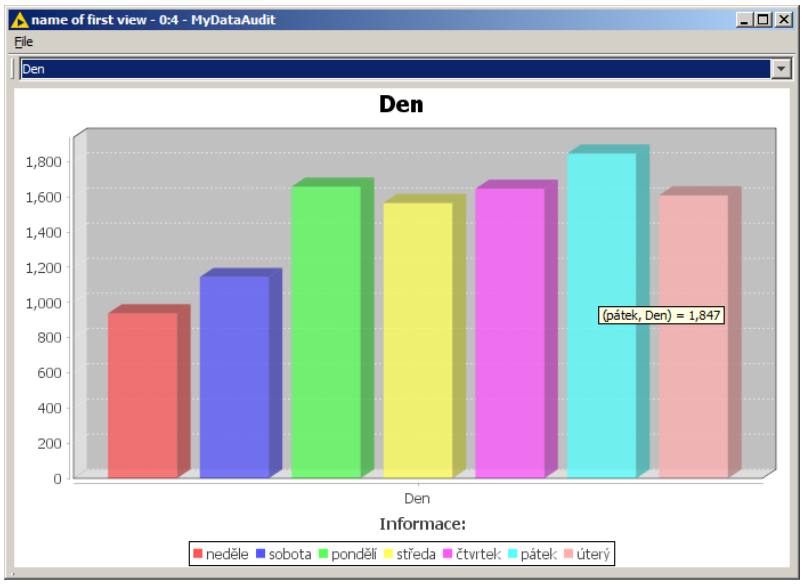
Tato kapitola znázorňuje možnost zapojení a využití uzlů, které byly vytvořeny v této práci. Znázorňuje spolupráci nově vytvořených uzlů a jejich výsledky. Data se kterými je pracováno, jsou reálná data se záznamy dopravních nehod od roku 2007 do roku 2018.
Tyto uzly budou demonstrovány na příkladě analýzy dopravních nehod v Liberci. Pro odfiltrování nechtěných dat je využit uzel MyFilter. Pomocí uzlu DBSCAN jsou vyhledány shluky a následně pomocí uzlu APRIORI jsou pro tyto shluky vyhledána asociační pravidla.



Obrázek 6.1: Zapojení nových uzlů v KNIME prostředí

6.1 Vyfiltrování vhodných řádků

Pro vyfiltrování dopravních nehod, které se staly v Liberci je využit vytvořený uzel MyFilter. Vstupní parametrem pro tento uzel je dotaz, který odfiltruje dopravní nehody netýkající se města Liberec. K vizualizaci dat byl použit uzel DataAudit pomocí kterého bylo zjištěna informace kdy se stávají nejčastější dopravní nehody.



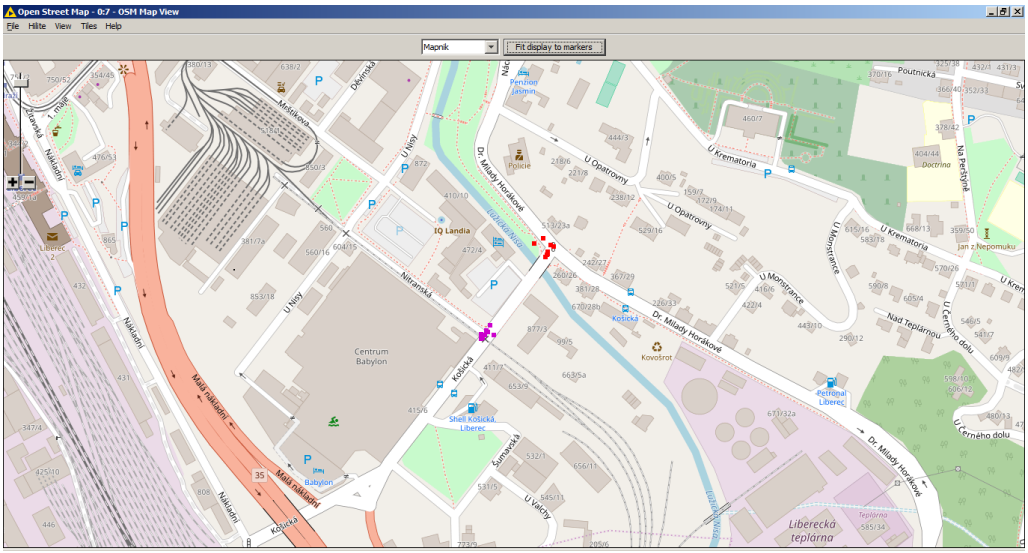
Obrázek 6.2: DataAudit zobrazující počty dopravních nehod v závislosti na dnu v týdnu.

Podle obrázku 6.2 se stávají dopravní nehody v Libereci nejčastěji v pátek. Proto do filtru byla přidána podmínka, aby byly vyhledány všechny dopravní nehody z obce Liberece, které se staly v Pátek. Výsledný select vypadá takto.

Obec = ”Liberec”and Den = ”pátek”

6.2 Vyhledání shluků dopravních nehod

K vyhledání shlukům dopravních nehod byl vybrán nově vytvořený uzel využívající algoritmu DBSCAN. Jako vstupní parametr pro oblast ve které budou dopravní nehody vyhledávány byla zvolena hodnota 20m.Pro minimální počet sousedských bodů, které jsou potřebné pro vznik shluku bylo zvoleno číslo 7.
Po doběhnutí uzlu byly odfiltrovány body, které byly označené jako šum. Po té body byly pomocí uzlu ColorManager obarveny, aby vynikly na mapě. Za pomoci uzlu OSM Map View byly tyto body zobrazeny viz. obrázek 8.3



Obrázek 6.3: Zobrazení nalezených shluků v Liberci

Na obrázku 8.3 jsou vidět 2 nalezené shluky nehod. První shluk nehod se nachází na ulici Košická. Druhým shlukem nehod je křižovatka ulic Dr. Milady Horákové s ulicí Košická.

6.3 Vyhledání asociační pravidel

Vyhledání asociačních pravidel bylo provedeno pro 2 nalezené shluky z minulé sekce. Vstupními parametry pro uzel APRIORI byly zvoleny support s hodnotou 7 a přesnost tvrzení do 70%. Skupina podle které budou generovány shluky dopravních nehod je zvolený sloupec s názvem shluku. Bohužel z důvodu malého počtu dopravních nehod nejsou v těchto shlucích závěry z uzlu APRIORI využitelné k odhalení příčin dopravní nehody v této oblasti.

> Nestížené podmínky, přehledná situace, nedošlo k sračce s pevnou překážkou => Nehodu zavinil řidič motorového vozidla.

V této oblasti se nachází pouze malé množství dopravních nehod, ze kterých se mohou vzít informaci pro vyhledání asociačních pravidel. Proto výsledek z uzlu APRIORI je ve většině případu podobný. Za předpokladu, kdy nejsou zhoršené dopravní podmínky, je přehledná situace a při nehodě nedošlo ke srážce s pevnou překážkou se dochází k závěru, pak nehodu zavinil řidič motorového vozidla.

7 Závěr

Úspěšně se podařilo vytvořit dva uzly, které využívají shlukujících algoritmů DBSCAN a OPTICS. Podařilo se vytvořit uzel, který využívá algoritmu APRIORI pro vyhledání asociačních pravidel s možností vytváření asociačních pravidel po skupinách. Tento uzel může ušetřit čas a práci s analýzou dopravních shluků. V této práci byl vytvořen uzel DataAudit, který nahrazuje dosavadní uzel Statistix a rozšiřuje ho o možnost vizualizace grafů a práci s nimi.
Pro práci s daty, zde byly vytvořeny dva uzly MyFilter a DensityFinder. Uzle MyFiteru se podařilo zjednodušit zadávání dotazu s pomocí doplňováních názvů sloupců a operací, které jsou dostupné.
Pro vyhledání hustoty dopravy na základě GPS souřadnic byly vytvořeny 2 typy uzlu DensityFinder, které dokážou postupně stahovat informace o hustotě dopravy. Dokáží doplnit k tabulce 69 nových hodnot. Tyto informace by mohly být v budoucnu využity pro dynamické přepočítávání hodnoty minimálního počtu sousedských bodů pro shlukový algoritmus DBSCAN.
Během vytváření této bakalářské práce byl vytvořen framework KnimeUserAdapter, který umožňuje programátorovi zjednodušení programování rozšiřujících uzlů v prostředí KNIME. Hlavní výhodou tohoto frameworku je větší čitelnost kódu. V případě kdyby bylo nutné uzel poupravit v budoucnu více čitelněší. Umožnuje rychlé vytvoření šablony výstupní tabulky. V prostředí KNIME je vytváření výstupních dat zdlouhavá činost. Tento framework užnuje změnit odebrat nebo přidat sloupec pomocí editace výstupní třídy.
V případě vytváření uzlu využívajícího algoritmu DBSCAN bylo dosaženo rychlejších výsledků, než u uzlu DBSCAN, který je vytvořen v KNIME. Tyto uzly mohou být užitečné pro analýzu dopravních nehod a hledání příčiny proč v dané oblasti nehody vznikají. Při tvorbě Apriori se podařilo vytvořit uzel, který je schopný vygenerovat asociační pravidla pro zadaný sloupec. Pro případ analýzy dopravních nehod se jedná o shluk, který byl vygenerován z DBSCANu.
Největším problémem během vytváření této bakalářské práce byl nedostatkem informací o způsobu vytváření rozšiřujících uzlů pro KNIME prostředí. Informace o tom, jak se dá vytvořit nový uzel pro KNIME se obtížně hledaly.
Tato bakalářská práce by mohla být také využiva jako příručka pro studenty začínající s prostředím KNIME, kteří by chtěli v tomto prostředí vytvářet své uzly.

Literatura

[1] 9WITTEN, I. H., Eibe FRANK a Mark A. HALL. Data mining: practical machine learning tools and techniques. 3rd ed. Burlington, MA: Morgan Kaufmann, c2011. Morgan Kaufmann series in data management systems. ISBN 0123748569.

[2] BERKA, Petr. Dobývání z databází. Praha: Academia, 2003. 366s. ISBN 80-200-1062-9

[3] LAMR, Marian. Big Data and Its Usage in Systems of Early Warning of Traffic Accident Risks. In: 2018 Sixth International Conference on Enterprise Systems (ES) [online]. IEEE, 2018, 2018, s. 154-157 [cit. 2019-01-03]. DOI: 10.1109/ES.2018.00031. ISBN 978-1-5386-8388-0. Dostupné z: https://ieeexplore.ieee.org/document/8588273/

[4] SPSS Modeler - Overview | IBM. [online]. Copyright © Copyright IBM Corporation 2018 [cit. 16.12.2018]. Dostupné z: https://www.ibm.com/products/spss-modeler

[5] Orange – Data Mining Fruitful & Fun. Orange – Data Mining Fruitful & Fun [online]. Copyright © University of Ljubljana [cit. 16.12.2018]. Dostupné z: https://orange.biolab.si/

[6] Weka 3 - Data Mining with Open Source Machine Learning Software in Java . Department of Computer Science: University of Waikato [online]. Dostupné z: https://www.cs.waikato.ac.nz/ml/weka/

[7] KNIME - Open for Innovation. KNIME - Open for Innovation [online]. Dostupné z: https://www.knime.com/

[8] Developer Guide | KNIME. KNIME - Open for Innovation [online]. Dostupné z: https://www.knime.com/developer-guide

[9] SCHILDT, Herbert. Mistrostvíí - Java. 1. vyd. Brno: Computer Press, 2014. Mistrostvíí. ISBN 978-80-2-1-145-8.

[10] HINNEBURG, Alexander a Daniel KEIM. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In: PROCEEDINGS OF THE FOURTH INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING [online]. New York:

AAAI, 1998, s. 58-65 [cit. 2017-04-29]. ISBN 978-1-57735-070-5. Dostupné z: http://www.aaai.org/Papers/KDD/1998/KDD98-009.pdf

[11] ESTER, Martin, Hans KRIEGEL, Jorg SANDER a Xiaowei XU. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING [online]. AAAI Press, 1996, s. 1-6 [cit. 2017-04-29]. ISBN 978-1-57735-004-0. Dostupné z: http://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[12] HAN, Jiawei a Micheline KAMBER. Data mining: concepts and techniques. San Francisco: Morgan Kaufmann Publishers, c2001. Morgan Kaufmann series in data management systems. ISBN 1558604898.

[13] ANKERST, Mihael, Markus M. BREUNIG, Hans-Peter KRIEGEL a Jörg SANDER. OPTICS: Ordering Points To Identify the Clustering Structure [online]. Institute for Computer Science, University of Munich, 1999 [cit. 2018-12-16]. Dostupné z: http://www.dbs.ifi.lmu.de/Publikationen/Papers/OPTICS.pdf

[14] ŘEZANKOVÁ, Hana, Dušan HÚSEK a Václav SNÁŠEL. Shluková analýza dat. 2., rozš. vyd. Praha: Professional Publishing, 2009, 218 s. ISBN 9788086946818.

[15] GOETHALS, Bart. Survey on Frequent Pattern Mining [online]. , 43 [cit. 2009-03-20]. Dostupné z: http://adrem.ua.ac.be/ goethals/software/survey.pdf

[16] AGRAWAL, Rakesh a Ramakrishnan SRIKANT. Fast Algorithms for Mining Association Rules. In: Proceeding VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases [online]. San Francisco: Morgan Kaufmann Publishers, 1994, s. 487-499 [cit. 2017-04-29]. ISBN 1-55860-153-8.

[17] Decimal degrees. (2019, April 11). Retrieved from https://en.wikipedia.org/wiki/Decimal_degrees

Seznam příloh

Příloha č. I. CD - ROM

Obsah CD-ROM

Přiložený CD-ROM obsahuje následující složky

- Kořenový adresář
  - Readme.txt - Návod k instalaci vytvořených uzlů do KNIME prostředí
- Dokumentace - Elektronická verze bakalářské práce
- Hotové uzly - Vyexportované uzly připravené k instalaci do KNIME prostředí.
- Zdrojový kód - Zdrojové kódy uzlů
- Workflow - KNIME projekt znázorňující možné zapojení uzlů