Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Lado Langof

Podatkovne baze za podatkovno rudarjenje

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: doc. dr. Matjaž Kukar

Ljubljana, 2015

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Lado Langof, z vpisno številko **63990083**, sem avtor magistrskega dela z naslovom:

Podatkovne baze za podatkovno rudarjenje

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

Ljubljana, 3. junij 2015 Podpis avtorja:

Za veliko število pripomb in predlogov, za večkratno pregledovanje vsebine naloge ter za usmeritve in pomoč pri izdelavi naloge se zahvaljujem svojemu mentorju ***doc. dr. Matjažu Kukarju****.*

Za slovnične nasvete in moralno podporo se zahvaljujem ***prof. slov. Sanji Jazbinšek****.*

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ARFF	Attribute-Relation File Format	Podatkovni format atribut-relacija
BQL	Bayesian Query Language	Bayesov poizvedovalni jezik
CDL	Common Data form Language	Jezik za splošno obliko podatkov
CRISP-DM	Cross Industry Standard Process for Data Mining	Splošni industrijski standardni proces za podatkovno rudarjenje
DMX	Data Mining Extensions	Razširitve za podatkovno rudarjenje
ETL	Extract, Transform, Load	izlušči, transformiraj, naloži
FDW	Foreign Data Wrappers	Ovoj za oddaljene podatke
GIST	Generalized Index Search String	Vrsta indeksa v PostgreSQL
HDF	Hierarchical Data Format	Hierarhični podatkovni format
JDBC	Java Database Connectivity	Java povezljivost do podatkovne baze
KDD	Knowledge Discovery in Databases	Odkrivanje znanja v podatkovnih bazah
ODBC	Open Database Connectivity	Odprta povezljivost do podatkovnih baz
OLAP	Online Analytical Processing	Večdimenzionalna podatkovna množica za analitično procesiranje
ORM	Object-relational mapping	Objektno-relacijske preslikave
PMML	Predictive Model Markup Language	Format za izmenjavo modelov podatkovnega rudarjenja
SEMMA	Sample, Explore, Modify, Model and Assess	Vzorči, raziskuj, spreminjaj, modeliraj, oceni
SOAP	Simple Object Access Protocol	Preprost protokol za dostop do objektov
SQL	Structured Query Language	Strukturiran povpraševalni jezik
SQL/MED	Management of External Data,	Upravljanje zunanjih podatkov

Povzetek

Naloga se ukvarja z iskanjem sinergij med orodji za podatkovno rudarjenje in sistemi za upravljanje podatkovnih baz (SUPB). Predstavljajmo si situacijo analitičnega problema nad podatki, ki jih je preveč za obdelavo izključno v glavnem pomnilniku in premalo, da bi motivirali postavitev podatkovnega skladišča ali porazdeljenega analitičnega sistema. Ciljno področje je torej en sam osebni računalnik, s katerim rešujemo probleme podatkovnega rudarjenja. Zanima nas, ali obstajajo orodja, ki omogočajo učinkovito obdelavo in pripravo takšne količine podatkov za nadaljnjo analizo.

Naloga se nanaša predvsem na drugi in tretji korak CRISP-DM standardnega modela podatkovnega rudarjenja, torej na razumevanje in pripravo podatkov, ne pa toliko na samo podatkovno rudarjenje. Preučuje, kako s pomočjo funkcionalnosti SUPB in orodij za luščenje, transformiranje ter nalaganje (ETL) čim bolj učinkovito pripraviti podatke za uporabo v podatkovnem rudarjenju. Podatkom zmanjšamo obseg in jih pretvorimo v ustrezno obliko. Optimizirani podatki, ki ne vsebujejo podvojenih zapisov, tiskarskih napak in drugih neželenih lastnosti ter vsebujejo le tiste atribute, ki jih lahko uporabimo, pozitivno vplivajo na hitrost in natančnost podatkovnega rudarjenja.

Cilj naloge je torej poiskati primerne načine (orodja oz. kombinacije orodij, metodologije) kako pridobiti relativno velike količine podatkov iz različnih virov in oblik, jih združiti in pretvoriti v obliko, ki jo lahko neposredno uporabimo za podatkovno rudarjenje, pri tem pa uporabljati SUPB in ETL orodja.

Abstract

This work is about looking for synergies between data mining tools and database management systems (DBMS). Imagine a situation where we need to solve an analytical problem using data that are too large to be processed solely inside the main physical memory and at the same time too small to put data warehouse or distributed analytical system in place. The target area is therefore a single personal computer that is used to solve data mining problems. We are looking for tools that allows us to effectively process and prepare such quantity of data for further analysis.

The main focus of this work is not on data mining itself but in particular on the second and third step of CRISP-DM process standard for data mining, that is *data understanding* and *data preparation* step. The question is how to use functionalities of various DBMS and ETL tools to prepare data as effectively as possible to use it in data mining. Unneeded data should be ignored and the remainder should be transformed into an appropriate form. Data mining execution time and accuracy should be improved when using optimized data that do not contain unneeded attributes, duplicate records, typos and other unwanted properties.

The objective of this work is thus to find appropriate practical methods (tools or combinations of tools, methodologies) for collecting relatively large amounts of data from different sources and in different forms, joining them and transforming this data to a format that can be used directly in data mining algorithms by using DMBS and ETL tools.

Poglavje 1

Uvod

Na področju podatkovnega rudarjenja se za hranjenje, izmenjavo in obdelavo podatkov pogosto uporabljajo tekstovni formati. Za hranjenje in obdelavo teh podatkov že vrsto let obstajajo rešitve v obliki različnih tipov podatkovnih baz.

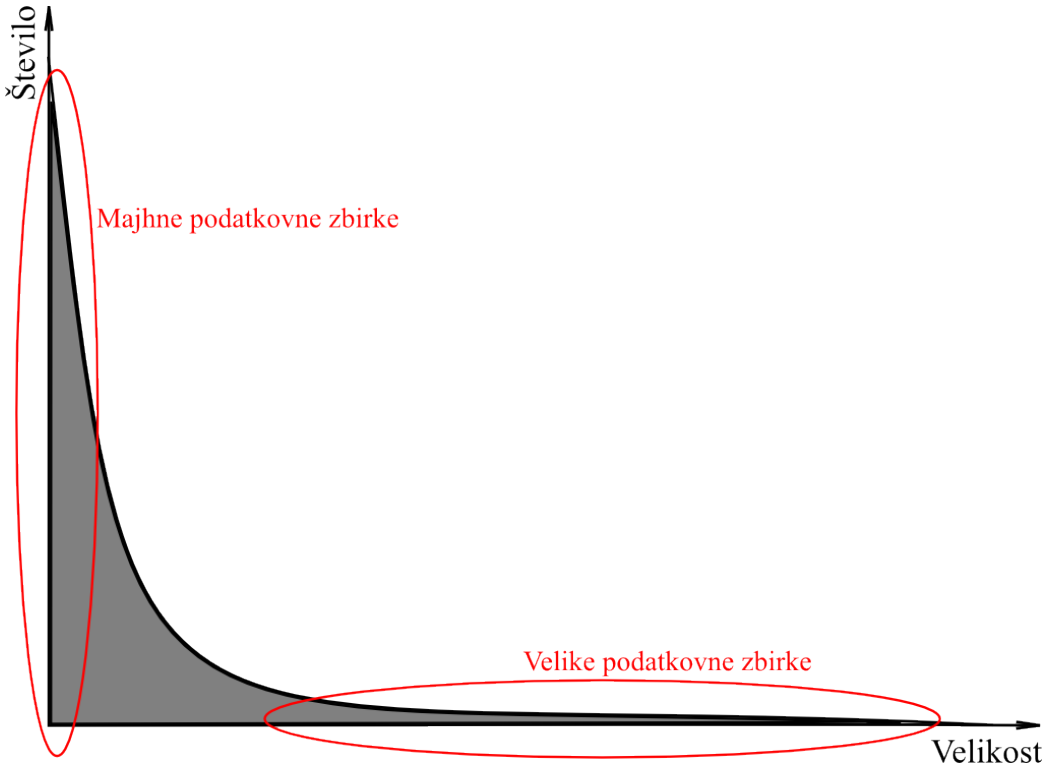
S pojavom velikih podatkov[1] postaja hranjenje podatkov v tekstovnih datotekah in njihova obdelava v pomnilniku vedno težje izvedljiva. Ob omejenih človeških, časovnih in finančnih kapacitetah je idealizirana postavitev specializiranega podatkovnega skladišča ali analitičnega porazdeljenega sistema v večini primerov nesmotrna – še posebej pri srednje velikih podatkih, ki so po velikosti blizu velilkosti pomnilnika in bi jih lahko obdelovali v enem samem osebnem računalniku.

Velikosti podatkovnih zbirk

Na sliki 1.1 je prikazano razmerje med številom in velikostjo velikih in malih podatkov. Malih podatkov je veliko, to so lahko že manjše Excelove razpredelnice. Zares velikih zbirk podatkov pa je malo, za njimi pogosto stojijo velike organizacije z velikim proračunom. Vmes je področje, kjer je podatkov premalo, da bi se

[1]Veliki podatki ("big data") so podatki, ki presegajo procesne zmožnosti konvencionalnih podatkovnih baz. So preveliki, se pridobivajo prehitro ali pa ne pašejo v strukture obstoječih arhitektur podatkovnih baz. Za pridobitev uporabne vrednosti iz teh podatkov moramo izbrati alternativen način procesiranja[1].

splačalo postavljati draga podatkovna skladišča in preveč za hranjenje v tekstovnih datotekah.



Slika 1.1: Razmerje med številom in velikostjo podatkovnih zbirk. Slika se zgleduje po [2], vendar ima obrnjeni osi. V svetu je veliko število majhnih podatkovnih zbirk in majhno število zares velikih zbirk.

Največje podatkovne zbirke imajo v času pisanja naloge več deset petabajtov podatkov. Primer je Facebook, ki naj bi imel v letu 2014 več kot 600 TB dnevnega prirasta podatkov in več kot 300 PB podatkov v podatkovnem skladišču[3]. Če opazujemo le velikosti podatkovnih zbirk, je težko reči pri kateri velikosti lahko govorimo o velikih podatkih. Ko smo omejeni na en sam računalnik in zato ne moremo izvajati paralelnega procesiranja oziroma "map-reduce" tehnik, postanejo podatki preveliki za procesiranje že veliko prej. Znani so primeri, ko se v produkcijskih sistemih statistika nad podatki izvajala več dni [4]. V letu 2000, ko je bila podatkovna zbirka podjetja WalMart znana kot ena največjih, naj bi preposta SELECT poizvedba potrebovala 30 dni le za sekvenčno branje vseh zapisov.

Ciljna velikost podatkov

Pri obdelavi podatkov na enem samem računalniku smo omejeni z velikostjo diska in z velikostjo pomnilnika. Z velikostjo diska zato, ker nanj ne moremo spraviti več podatkov, kot je njegova velikost. Z velikostjo pomnilnika pa zato, ker veliko programov deluje tako, da podatke pred obdelavo v celoti prebere v glavni pomnilnik. Pomnilnika pa ponavadi zmanjka veliko prej kot diska.

Ciljno področje naloge so podatki vmesne velikosti, torej takšni, ki so blizu velikosti pomnilnika ali večji in manjši od velikosti diska. Te podatke bi lahko s pravilnim pristopom zmanjšali in preoblikovali tako, da bi brez izgube informacij nad njimi izvajali algoritme podatkovnega rudarjenja, ki bi se zato izvedli hitreje, rezultat pa bi bil zaradi odprave napak v podatkih in poenotenja oblike podatkov bolj natančen. Pri orodjih za podatkovno rudarjenje, ki morajo podatke pred obdelavo v celoti prebrati v pomnilnik, pa bi z dovolj velikim zmanjšanjem velikosti podatkov takšno izvajanje sploh omogočili.

Vzorčenje

Statistika se problema iskanja značilnosti velikih množic loti z vzorčenjem. Znanje pridobljeno na vzorcu pa z neko verjetnostjo posplošuje na celotno množico. Pri vzorčenju se ukvarja s tem, kako vzorčiti in kako veliko množico vzorca vzeti, da bo vzorec dovolj reprezentativen.

Tehnike vzorčenja lahko razdelimo v tri skupine[5]:

- **Verjetnostno vzorčenje** – vsak primer ima enako verjetnost, da je izbran.
- **Premišljeno vzorčenje** – tisti, ki izbira vzorec, ga skuša narediti reprezentativnega (npr. na podlagi kriterijev, vsak n-ti zapis ipd.).
- **Vzorčenje brez pravil** – vzorec vzamemo brez vsakih pravil. Tak vzorec je premalo reprezentativen.

Pri vzorčenju lahko uporabljamo tudi kombinacijo teh načinov. Npr. celotno množico razdelimo v skupine glede na nek kriterij, nato pa iz vsake skupine vzamemo enako število naključnih predstavnikov skupine.

Enako idejo lahko uporabimo tudi pri pripravi podatkov za podatkovno rudarjenje. Če jih je preveč da jih obdelovali na enem računalniku, jih lahko na najbolj enostaven način zmanjšamo tako, da jih vzorčimo. Vendar pa moramo biti pri tem pazljivi. V praktičnem delu te naloge na strani 101 je primer, ki uporablja časovne vrste in vsebuje dnevne podatke o tem ali se je posamezen disk (iz množice diskov) pokvaril. Ker diski v računalnikih ponavadi obratujejo precej dni preden se pokvarijo, bi v primeru naključnega vzorčenja celotne množice zelo verjetno izgubili veliko tistih zapisov, ki so zabeležili odpoved diska.

Pridobivanje podatkov

Pred pričetkom izvajanja podatkovnega rudarjenja v ožjem pomenu moramo podatke najprej pridobiti, pretvoriti in shraniti v primerno obliko. Pridobimo jih lahko iz različnih virov: iz tekstovnih ali binarnih datotek, iz drugih podatkovnih baz, iz spletnih virov, iz človeških virov idr. Podatke pridobljene iz različnih virov moramo poenotiti. Poskrbeti moramo, da ni podvojenih zapisov, da v podatkih ni slovničnih napak, da so v pravem kodnem naboru, da poenoteno obravnavamo neznane in ničelne vrednosti ipd. Za shranjevanje podatkov lahko uporabimo tekstovne datoteke, binarne datoteke ali podatkovne baze. Ponavadi so podatki, pripravljeni za podatkovno rudarjenje, dobro strukturirani, a denormalizirani, ker znajo mnoga orodja za podatkovno rudarjenje (ne pa vsa) delati le z eno samo veliko podatkovno tabelo.

Denormalizacija in agregacija podatkov

Večina glavnih proizvajalcev SUPB kot tudi proizvajalcev aplikacij za podatkovne analize oglašuje orodja za podatkovno rudarjenje. Veliko teh orodij deluje na relacijskih podatkovnih bazah. Predpostavljajo, da so podatki v tabelah. Ta orodja so usmerjena v manipulacijo tabel (ali v mnogih primerih en sam tabelarični pogled podatkov)[7].

Pri denormalizaciji podatkov se pri relacijah tipa ena-na-ena (an. one-to-one) in več-na-ena (an. many-to-one) vse informacije ohranijo, pri relacijah tipa ena-na-več (an. one-to-many) pa zaradi agregacije izgubimo nekaj informacij[6]. Primer

je podan na sliki 1.2. Če bi npr. radi v eni tabeli prikazali podatke o posamezni stranki in agregirane podatke o njenih nakupih, potem pri združevanju podatkov izgubimo informacije o posameznem nakupu. V našem primeru izgubimo informacijo o vseh datumih nakupa (razen zadnjem) in posameznem znesku nakupa (razen, če ima stranka le en nakup). Z odebeljeno pisavo so v tabeli NAKUPI označene vrednosti, ki so se pri agregaciji v tabelo STRANKA_AGREGIRANO, izgubile. Na sliki 1.2 je prikazan tudi primer, ko denormalizacijo izvajamo tako, da nas zanimajo posamezni nakupi.

STRANKA

id_stranke	ime	priimek
1	Miha	Zajc
2	Janez	Novak

NAKUP

id_nakupa	stranka	datum	znesek
1	1	**2014-01-01**	**10,00 €**
2	1	2014-02-02	**15,00 €**
3	2	2014-02-05	8,00 €

STRANKA_AGREGIRANO

id_stranke	ime	priimek	zadnji_nakup	znesek_skupaj
1	Miha	Zajc	2014-02-02	25,00 €
2	Janez	Novak	2014-02-05	8,00 €

NAKUPI_DENORMALIZIRANO

id_nakupa	datum	znesek	id_stranke	ime	priimek
1	2014-01-01	10,00 €	1	Miha	Zajc
2	2014-02-02	15,00 €	1	Miha	Zajc
3	2014-02-05	8,00 €	2	Janez	Novak

Slika 1.2: Primer agregacije pri denormalizaciji podatkov.

Obstaja tudi nekaj tehnik podatkovnega rudarjenja, kjer lahko delamo direktno z normaliziranimi podatki, torej z relacijami. Primer so induktivne podatkovne baze in induktivne poizvedbe [6], ki pa jim manjkajo praktične izvedbe. Še eden od primerov je BayesDB[2] in vgrajen Bayesov poizvedovalni jezik (BQL), ki pa je v času pisanja naloge še v zgodnji fazi razvoja. Tudi Microsoft SQL Server Analysis Services ima podobne funkcionalnosti – prek razširitev za podatkovno rudarjenje (DMX)[3].

[2]Dostopen na naslovu: http://probcomp.csail.mit.edu/bayesdb/

[3]Glej: https://msdn.microsoft.com/en-us/library/ms132058.aspx

Od tekstovnih datotek do SUPB

Za shranjevanje in izmenjavo podatkov je uporaba tekstovnih datotek (npr. CSV, TSV, JSON) še vedno zelo priljubljena. Razlog je verjetno v enostavnosti uporabe. Ni nam treba poznati, nameščati in nastavljati SUPB ali drugih orodij. Tudi pravic dostopa in požarnih zidov nam ni potrebno nastavljati. Lahko jih enostavno objavimo na spletu kot datoteke. Za izmenjavo in hranjenje so tekstovne datoteke uporabne tudi, če je količina podatkov večja. Problem se pojavi, ko želimo nad temi podatki izvesti neko operacijo, npr. poiskati neko vrednost v celotni datoteki in jo spremeniti. Če je datoteka majhna s tem nimamo težav, pri 100 MB veliki datoteki pa je že očitno, da je lahko na videz preprosta operacija zamudna. Vsa orodja niso enako učinkovita. Za primer lahko navedem, da z urejevalnikom *gedit* nisem mogel odpreti 16 MB velike datoteke, v urejevalniku *vim* pa je odpiranje trajalo 4 sekunde. Ni vseeno na kak način program odpre datoteko – ali prebere celo ali le tistih nekaj blokov, ki jih potrebujemo. Orodja za podatkovno rudarjenje pogosto preberejo celotno datoteko s podatki v pomnilnik. In če ne prej, naletimo na omejitve, ko zaradi branja celotne datoteke zmanjka pomnilnika.

Velike podatkovne datoteke je nepraktično popravljati ročno. Zato operacije nad njimi opravljamo v korakih, ki so avtomatizirani. Na tak način pripravimo podatke, da lahko potem nad njimi izvajamo samo podatkovno rudarjenje. Na konferenci KDD 2003 je *U. Fayyad* izjavil, da v tipični seji podatkovnega rudarjenja večino časa porabi za pridobivanje in manipulacijo podatkov, ne pa toliko za samo podatkovno rudarjenje in raziskovanje [6]. Podobno ugotavljata tudi avtorja knjige Data Mining [8] *I. Witten* in *E. Frank*, ki navajata, da za celoten proces podatkovnega rudarjenja ponavadi vložita največ truda za pripravo vhodnih podatkov. Tudi v samem standardu CRISP-DM ([9], str 39) avtorja omenjata domnevo, da se 50–70 odstotkov časa in truda v projektu podatkovnega rudarjenja porabi v fazi priprave podatkov, 20–30 procentov v fazi razumevanja podatkov, medtem ko se skupaj samo 10–20 odstotkov porabi za faze modeliranja, evaluacije in razumevanja ter samo 5–10 odstotkov za fazo uvajanja.

Zgodovinsko gledano so orodja za podatkovno rudarjenje delovala nad podatki, shranjenimi v tekstovnih datotekah, pri čemer so bili atributi ločeni z vejico ali drugače. Večina orodij se premika k podpori podatkovnega dostopa preko ODBC

ali JDBC. Dejanski aplikacijski programerski vmesniki kot sta ODBC in JDBC pa so slabo opremljeni za prejemanje velikih količin podatkov. Veliko orodij za podatkovno rudarjenje deluje tako, da prekopira vse pomembne podatke ali v pomnilnik ali v njihov lastni diskovni prostor, šele nato deluje nad podatki. To je potratno – bolje bi bilo, če bi lahko uporabili podatkovno bazo za pridobivanje in shranjevanje podatkov tudi med izvajanjem algoritma za podatkovno rudarjenje ter vanjo preložili tudi nekaj algoritmičnih opravil. Nekaj orodij začenja s tem, vendar skozi ozko, lastniško integracijo s podatkovno bazo. Standardiziran dostop bi bil boljši, a zahteva razumevanje tipov dostopnih vzorcev, ki jih naredijo orodja za podatkovno rudarjenje[7].

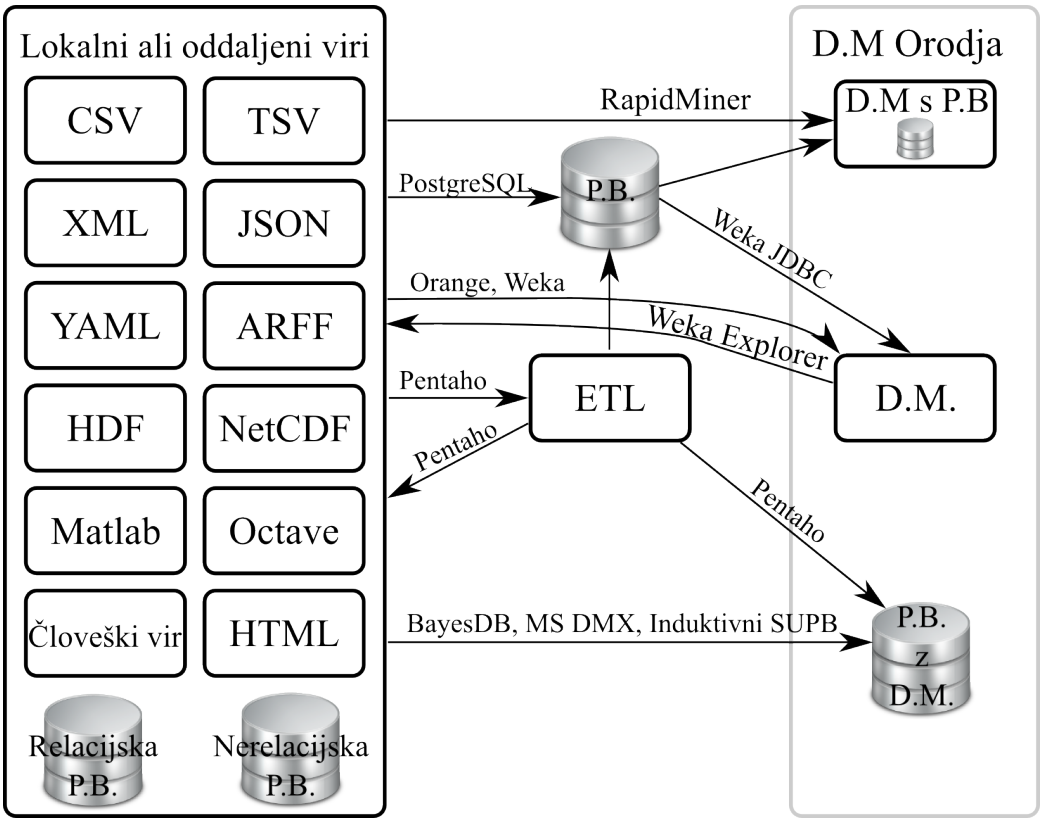
Namen naloge

Namen naloge je – glede na vnaprej določene kriterije (navedene v naslednjem poglavju) – poiskati čim bolj učinkovito orodje ali kombinacijo orodij, s pomočjo katerih lahko izvedemo vsa opravila druge in tretje faze CRISP-DM.

Podatke lahko pripravimo na več načinov, glede na to na kakšen način deluje orodje za podatkovno rudarjenje in katere formate podpira. Možne povezave med orodji so prikazane na sliki 1.3, možne pa so še druge povezave, ki na sliki niso označene.

Na levi strani slike so prikazani različni podatkovni viri. Medtem ko se v prvem koraku seznanimo s problemom, se drugi korak CRISP-DM vedno začne s pridobivanjem podatkov. Virov je lahko več in so lahko v različnih oblikah in formatih. Končni cilj so združeni podatki v obliki, ki jo lahko uporabimo direktno v orodjih za podatkovno rudarjenje. Na primer, imamo nekaj časovnih podatkov o stanju vremena v ARFF in nekaj podobnih podatkov v CSV datotekah. V ARFF datotekah so podatki o stanju vremena vzorčeni v 15-minutnih intervalih, v CSV datotekah pa v 5-minutnih intervalih. Pri združevanju podatkov iz teh dveh virov bi morali podatke pri 5-minutnem vzorčenju agregirati tako, da bi izračunali povprečja za zaporedne tri intervale ali pa bi uporabili le vsak tretji zapis. Šele nato bi lahko podatke združili. Poenotiti bi morali tudi imena stolpcev in oznake za manjkajoče vrednosti, odpraviti nerazumno visoke ali nizke vrednosti, izdelati izvedene atribute najvišje in najnižje dnevne temperature ipd. Problem je

izmišljen, v praksi pa je veliko podobnih.



Slika 1.3: Pri pripravi podatkov za podatkovno rudarjenje moramo pogosto delati z več kot enim orodjem. Idealno bi bilo, če bi lahko vse korake izvedli v enem samem orodju.

Načinov za dosego cilja je več. Najbolj pogost način je, da orodje za podatkovno rudarjenje zahteva podatke iz tekstovne datoteke. Ta mora biti v formatu, ki ga podpira orodje. Pretvorbo formata in transformacije podatkov lahko izvedemo z ETL orodjem. Drugi način je uvažanje vseh podatkov v SUPB (neposredno ali z ETL orodjem) in izvajanje transformacij v SUPB. Te podatke pa lahko potem izvozimo v tekstovni format in uporabimo s klasičnim orodjem za podatkovno rudarjenje. Še en način je uporaba orodja za podatkovno rudarjenje, ki ima integrirano SUPB in omogoča tudi vse transformacije, ki jih zahteva CRISP-DM. Obstaja tudi nekaj SUPB, ki omogočajo izvajanje transformacij in podatkovnega rudarjenja direktno v podatkovni bazi prek razširitev za jezik SQL (npr. BQL).

Nenazadnje bi lahko omenili tudi pridobivanje in transformiranje podatkov s programskimi jeziki, s pomočjo za to namenjenih programskih knjižnic.

Kriteriji ocenjevanja

V nalogi bomo ocenjevali SUPB in ETL orodja glede na podporo drugega in tretjega koraka CRISP-DM. Kriterije ocenjevanja lahko razdelimo na *kvantitativne* in *kvalitativne*. Prve lahko podamo na podlagi meritev ali dejstev, kvalitativnih pa ne moremo izmeriti, kot npr. ne moremo izmeriti kvalitete uporabniškega vmesnika, čeprav precej vpliva na izbiro produkta.

Kvantitativni kriteriji ocenjevanja SUPB in ETL orodij za opisan namen so:

- **Ekonomski in pravni kriteriji** – sem spada **cena** orodja. Pri projektih, kjer smo zelo omejeni s finančnimi viri, nas lahko cena odvrne od uporabe. Še en kriterij, ki je pogosto tudi povezan s ceno je, ali je produkt **odprtokoden** ali ne. Odprta koda je priročna, kadar lahko s poznavanjem programskega jezika, v katerem je orodje napisano, vplivamo na nadaljnji razvoj orodja ali pa ga prilagodimo po svojih željah. Pogosto so takšna orodja zastonj, lahko pa so tudi plačljiva (odvisno od licence). Slabost odprte kode pa je, predvsem pri zastonj produktih, da ponavadi pri napaki v izdelku nimamo pravne zaščite niti uradne podpore za odpravo napake.

- **Razširljivost** – ali je orodje takšno kot je, ali ponuja tudi vtičnike, programske vmesnike ali druge načine za dodajanje novih razširljivosti?

- **Vmesniki** – ali ima orodje grafični vmesnik? Zaradi enostavnost uporabe si želimo, da ima orodje uporabniški vmesnik. Če želimo opravila avtomatizirati, pa je bolj uporabno, da ga lahko upravljamo prek ukazov ali programskih vmesnikov. Ali lahko z orodjem tudi vizualiziramo podatke in rezultate?

- **Podprti tekstovni in binarni formati** – kateri formati datotek so podprti pri vhodnih in izhodnih operacijah? Npr. ARFF, HDF5, NetCDF, CSV ... Ali omogoča tudi uvoz podatkov iz gnezdenih struktur kot je npr. XML?

- **Dostop do drugih SUPB** - iz katerih SUPB in na kakšen način (npr. ODBC, JDBC) lahko pridobivamo podatke oz. v katere SUPB jih lahko izvažamo? Kakšna je hitrost prenosa podatkov?

- **Ročno dodajanje zapisov** – ali orodje omogoča tudi ročno dodajanje zapisov? To je lahko uporabno, če imamo dostop do človeških ali tiskanih virov in bi radi zapise ročno dodajali ali popravljali.

- **Tehnične omejitve** – kolikšno je **največje število atributov**, ki jih še lahko uvozimo v orodje? Problemi za podatkovno rudarjenje imajo lahko tudi več deset tisoč atributov. Nekateri od njih so pridobljeni iz drugih atributov, nekateri tudi z agregacijskimi funkcijami. Ali obstaja **omejitev glede količine podatkov na zapis**? Ali orodje omejuje **največjo skupno količino podatkov** na podatkovno tabelo, ali je omejitev le datotečni sistem? Na katerih **operacijskih sistemih** deluje orodje?

- **Poraba pomnilnika** – kako dobro zna orodje upravljati s pomnilnikom. Ali prebere celotno podatkovno tabelo v pomnilnik ali pa jo bere po blokih glede na zahtevo? Ali je pri podatkih, ki so blizu velikosti pomnilnika, opazna bistvena upočasnitev pri uporabi? Ali se to zgodi že prej?

- **Časi izvajanja** – merimo lahko čas izvajanja uvažanja podatkov in čas transformacij, vendar pa je ta kriterij odvisen od količine in tipa podatkov ter od transformacij, ki jih uporabimo. Zato neke splošne ocene časa izvajanja ne moremo navesti, lahko pa navedemo čas za nek konkreten problem. Ni vseeno, ali odstranjujemo prazne presledke na začetku in koncu besedilnega polja, ali pa iščemo robove v binarnih slikah.

  Pri konkretnih podatkih bi lahko merili čas izvajanja posameznih elementarnih korakov CRISP-DM. Vse meritve bi morali izvesti v več ponovitvah na enaki strojni opremi in navesti časovno povprečje (ekstremne vrednosti pa zavreči). Težava je tudi v načinu merjenja časa. Nekatere SUPB pri vsaki poizvedbi pokažejo čas izvajanja poizvedbe. Če je orodje grafično in ne prikaže časa izvajanja, potem je natančno izvajanje meritve težje izvedljivo.

Kvalitativni kriteriji so:

- **Uporabniška izkušnja** –Kakšna je **dokumentacija**? Ali se produkt **aktivno posodablja** ali gre za mrtev produkt na katerem nihče več ne dela? Ali obstaja **podpora** in uporabniški forumi? Kako zapleten je vmesnik? Ali zahteva veliko **učenja**? Ali ima tekstovni vmesnik dopolnjevanje izrazov (an. autocompletition)? Ali je izvajanje korakov enostavno, ali zahteva veliko mero angažiranosti uporabnika?

Poglavje 2

Standardne metodologije podatkovnega rudarjenja

Članek o prihajajočih standardih podatkovnega rudarjenja iz leta 2001[7] govori o pomanjkanju standardov podatkovnega rudarjenja. Trdi, da ni poskusa za en sam standard, ampak se namesto tega pojavljajo standardi, ki podpirajo različne vidike podatkovnega rudarjenja. Standarde razvršča v naslednje skupine:

- standardi za opravila, ki jih moramo izvesti pri podatkovnem rudarjenju (npr. formalna definicija vhodov in izhodov iz učne in testne faze klasifikatorja, npr. PMML),
- standardi za podporo tehnologije (npr. SQL je standard za dostop do podatkov),
- procesni standardi (zaporedje dogodkov v izvajanju projekta podatkovnega rudarjenja, npr. CRISP-DM).

Omenja pa še arhitekturne standarde (npr. kakšen naj bo vmesnik med orodjem za podatkovno rudarjenje in SUPB) in standarde, ki določajo, kako zajeti podatke na spletu.

Za izvajanje podatkovnega rudarjenja je na voljo le malo metodologij. Glede na anketo na strani KDnuggets iz leta 2007[10] je najbolj popularna metodologija CRISP-DM, omeniti pa moramo še SEMMA in KDD. Ostali anketiranci so uporabljali svoje lastne metodologije, metodologije organizacije, specifične metodologije

ali pa metodologije sploh niso uporabljali.

Azevedo in Santos v svojem članku[11] primerjata KDD, SEMMA in CRISP-DM. Pri vseh treh je podatkovno rudarjenje v ožjem smislu le eden izmed korakov celotnega procesa. Vse metodologije vključujejo izbiranje podatkov, predprocesiranje, transformiranje, samo podatkovno rudarjenje in interpretacijo oz. ocenjevanje. Ker je SEMMA neposredno povezana s produktom *SAS Enterprise Miner*, lahko na pet faz SEMMA gledamo kot na praktično implementacijo KDD procesa. Primerjava s CRISP-DM ni tako očitna. Vanj je kot prva faza vključena še faza razumevanja problema, ki podaja pomembno predznanje in cilje za končnega uporabnika, kot zadnja faza pa je vključena še faza uporabe v praksi. Faza razumevanja podatkov je pri CRISP-DM kombinacija KDD izbire in predprocesiranja podatkov. Pri pripravi podatkov gre za transformacije, modelirna faza pa je faza podatkovnega rudarjenja. Primerjava omenjenih treh metodologij je podana v razpredelnici 2.2.

KDD	**SEMMA**	**CRISP-DM**
1. Pred KDD	———	1. Razumevanje problema
2. Izbira	1. Vzorčenje	2. Razumevanje podatkov
3. Predprocesiranje	2. Raziskovanje	
4. Transformacije	3. Spreminjanje	3. Priprava podatkov
5. Podatkovno rudarjenje	4. Modeliranje	4. Modeliranje
6. Interpretacija/Vrednotenje	5. Ocenjevanje	5. Vrednotenje
7. Post KDD	———	6. Uporaba v praksi

Tabela 2.2: Primerjava korakov treh najbolj znanih metodologij podatkovnega rudarjenja[11]. Nekateri avtorji, npr. Fayyad in ostali[12], delijo KDD na 9 korakov, vendar lahko tri izmed njih štejemo pod samo podatkovno rudarjenje. Prvi korak je predkorak pridobivanja znanja, zadnji pa korak uporabe pridobljenega znanja. Glej tudi poglavje o KDD na strani 21.

Ker CRISP-DM natančno definira procesne korake in podkorake podatkovnega rudarjenja in ker gre za najbolj popularno metodo v praksi, v tej nalogi sledim korakom te metodologije, predvsem pa drugemu in tretjemu koraku, t.j. razumevanje in priprava podatkov.

2.1 CRISP-DM

CRISP-DM je najbolj popularen procesni model za podatkovno rudarjenje. Zasnovan je bil leta 1996, ko je bil trg za podatkovno rudarjenje še dokaj nerazvit. Projekt je bil voden s strani petih podjetij, ki so za namen razvoja CRISP-DM ustanovili posebno interesno skupino. Za izmenjavo idej so sklicali enodnevno konferenco, ki je doživela neverjetno velik odziv. Izkazalo se je, da v različnih podjetjih uporabljajo podobne procese za podatkovno rudarjenje, le proces je drugače razmejen in koraki so drugače poimenovani. Leta 1999 je bil izdelan osnutek, naslednje leto pa je izšla uradna verzija CRISP-DM 1.0. CRISP-DM je de facto standard za podatkovno rudarjenje v industriji.

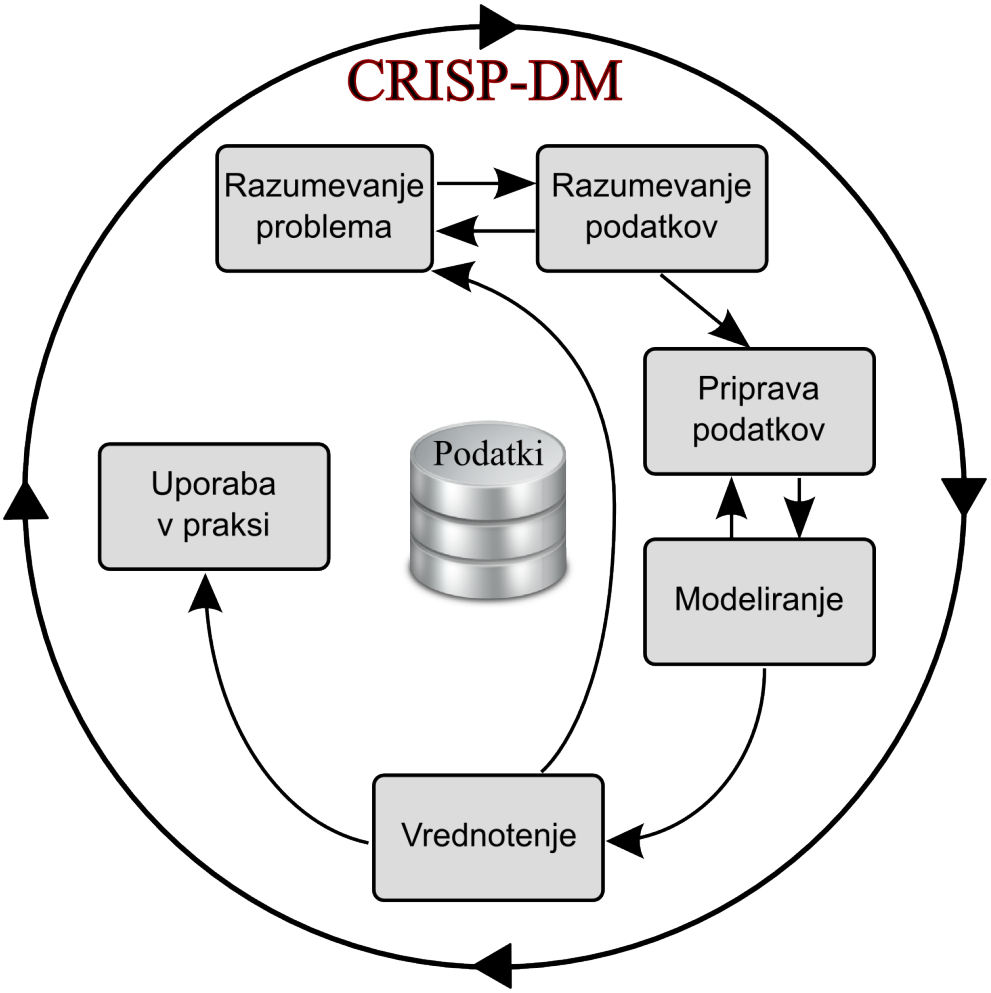
CRISP-DM je tudi metodologija. Uspešna je, ker je osnovana na praktičnih primerih. CRISP-DM je predstavljen kot hierarhični procesni model, sestavljen iz opravil. Standard opisuje opravila na *4 nivojih abstrakcije*. Prvi nivo je najbolj splošen, zadnji pa najbolj podroben. Na prvem nivoju je proces podatkovnega rudarjenja razdeljen na *6 faz*. Vsaka faza je sestavljena iz *splošnih opravil*, ki pokrivajo vse možne situacije podatkovnega rudarjenja. Tretji nivo – *specializiran nivo opravil* – opisuje, kako izvesti akcije splošnih opravil v konkretnih situacijah. Četrti nivo – *procesne instance* – pa opisuje akcije in odločitve znotraj posameznega specifičnega opravila. Organiziran je glede na opravila, definirana na višjih nivojih in predstavlja, kaj se je dejansko zgodilo.

CRISP-DM metodologija loči med *referenčnim modelom* in *uporabniškim priročnikom*. Prvi predstavlja hiter pregled faz, opravil in njihovih izhodov ter opisuje, kaj delati v projektu podatkovnega rudarjenja. Drugi pa daje podrobnejše nasvete za vsako fazo in vsako opravilo faze.

CRISP-DM referenčni model

CRISP-DM referenčni model predstavlja življenjski cikel projekta podatkovnega rudarjenja. Vsebuje faze projekta, opravila teh faz in relacije med njimi. V najbolj splošnem pogledu je razdeljen na 6 faz, ki so prikazane na sliki 2.1.

Zaporedje faz CRISP-DM ni togo. Zaželjeno je premikanje naprej in nazaj med fazami. Izhod faze je odvisen od tega, katera faza oz. opravilo mora biti izvedeno naslednje. Na sliki 2.1 so prikazane najpomembnejše in najbolj pogoste odvisno-



Slika 2.1: CRISP-DM, faze referenčnega modela

sti med fazami. Zunanji krog predstavlja ciklično naravo procesa podatkovnega rudarjenja. Pridobljeno znanje še ne pomeni konec procesa – lahko poraja nova, bolj specifična vprašanja. Proces podatkovnega rudarjenja se bogati iz izkušenj pridobljenih v predhodnih fazah.

Faze CRISP-DM so naslednje:

1. **Razumevanje problema (Business understanding)**
   Osredotoča se na razumevanje ciljev in zahtev projekta. S tem znanjem se definira problem podatkovnega rudarjenja. Izdela se tudi začetni plan za dosego ciljev.

2. **Razumevanje podatkov (Data understanding)**
   Začne se z začetnim zbiranjem podatkov. Sledijo aktivnosti za spoznavanje podatkov, identifikacijo kvalitativnih problemov, odkrivanje prvih spoznanj o podatkih, zaznavanje zanimivih podmnožic za oblikovanje hipotez glede skritih informacij.

3. **Priprava podatkov (Data preparation)**
   Pokriva vse aktivnosti za kreiranje končne množice podatkov – to so podatki nad katerimi se bodo izvajali algoritmi podatkovnega rudarjenja. Priprava podatkov se pogosto izvedene večkrat in v nobenem predpisanem vrstnem redu. Opravila vključujejo izbiro tabel, zapisov, atributov, pa tudi transformacije in čiščenje podatkov za modelirna orodja.

4. **Modeliranje (Modeling)**
   V tej fazi se izberejo in uporabijo različne modelirne tehnike ter nastavijo njihovi parametri. Nekatere tehnike imajo posebne zahteve glede oblike podatkov, zato se je pogosto potrebno vračati v fazo priprave podatkov.

5. **Vrednotenje (Evaluation)**
   V tej fazi je model izdelan. Pred njegovo uporabo ga je potrebno temeljito ovrednotiti in pregledati korake njegove priprave. Na koncu faze se odloča o njegovi uporabi v praksi. Model mora ustrezati postavljenim ciljem in ne sme biti pomanjkljiv. Preveri se tudi, ali je bilo kaj spregledano, ali je model pravilno zgrajen, ali je bilo odkrito še kaj, kar ni bilo v prvotem planu, ali lahko model testiramo v aplikacijah in ali model zadosti potrebam.

6. **Uporaba v praksi (Deployment)**
   Kreiranje modela še ne pomeni konec projekta. Pridobljeno znanje je potrebno organizirati in predstaviti. To pogosto vključuje uporabo modelov pri odločanju organizacije. Ta faza je lahko preprosta kot ustvarjanje poročila ali kompleksna kot ponavljajoč proces podatkovnega rudarjenja skozi organizacijo. V praksi jo ponavadi uporabi sama stranka in ne podatkovni analitik. Pomembno je, da stranka razume kaj mora narediti, da uporabi ustvarjeni model.

Ker se naloga ukvarja z aktivnostmi 2. in 3. faze CRISP-DM procesnega

modela, sta v nadaljevanju bolj podrobno opisani ti dve fazi. Opravila faz in njihovi izhodi so prikazani na sliki 2.2.

1 Razumevanje poslovnega problema	2 Razumevanje podatkov	3 Priprava podatkov	4 Modeliranje	5 Vrednotenje	6 Uporaba v praksi
1.1 Določanje poslovnih ciljev • Ozadje • Poslovni cilji • Kriteriji za poslovni uspeh **1.2 Ocenjevanje situacije** • Popis virov, zahtev in omejitev • Tveganja in nepredvideni izdatki • Terminologija • Stroški in koristi **1.3 Določanje ciljev podatkovnega rudarjenja** • Cilji podatkovnega rudarjenja • Kriteriji uspeha podatkovnega rudarjenja **1.4 Izdelava projektnega plana** • Projektni plan • Začetna ocena orodij in tehnik	**2.1. Začetno zbiranje podatkov** • Poročilo začetnega zbiranja podatkov **2.2 Opisovanje podatkov** • Poročilo opisa podatkov **2.3 Raziskovanje podatkov** • Poročilo raziskovanja podatkov **2.4 Preverjanje kvalitete podatkov** • Poročilo kvalitete podatkov	**3.1 Izbira podatkov** • Utemeljitve za vključitev/izključitev **3.2 Čiščenje podatkov** • Poročilo čiščenja podatkov **3.3 Ustvarjanje podatkov** • Pridobljeni atributi • Ustvarjeni zapisi **3.4 Integracija podatkov** • Združeni podatki **3.5 Preoblikovanje podatkov** • Preoblikovani podatki • Podatkovne množice • Opisi podatkovnih množic	**4.1 Izbira modelirnih tehnik** • Modelirne tehnike • Modelirne domneve **4.2 Izdelava testnega plana** • Testni plan **4.3 Izgradnja modela** • Nastavitve parametrov • Modeli • Opisi modelov **4.4 Ocenjevanje modela** • Ocena modela • Revizija nastavitev parametrov	**5.1 Vrednotenje rezultatov** • Ocena rezultatov podatkovnega rudarjenja glede na kriterije poslovnega uspeha • Odobreni modeli **5.2 Pregledovanje procesa** • Pregled procesa **5.3 Določanje naslednjih korakov** • Seznam možnih dejanj • Odločitev	**6.1 Planiranje uporabe v praksi** • Plan uporabe v praksi **6.2 Planiranje nadzora in vzdrževanja** • Plan nadzora in vzdrževanja **6.3 Izdelava zaključnega poročila** • Zaključno poročilo • Zaključna predstavitev **6.4 Pregled projekta** • Dokumentacija izkušenj

Slika 2.2: Na sliki so prikazane faze CRISP-DM, opravila posameznih faz in izhodi teh opravil.

CRISP-DM razumevanje podatkov

2.1 Začetno zbiranje podatkov

V tem opravilu moramo pridobiti dostop do podatkov in jih zajeti. Po potrebi jih uvozimo v orodje za razumevanje podatkov. Če je podatkovnih virov več, se je v tej fazi ali v fazi priprave podatkov potrebno ukvarjati z integracijo podatkov.

Izhod tega opravila je *poročilo začetnega zbiranja podatkov*, ki vključuje seznam zajetih podatkovnih množic, skupaj z lokacijami, metodami za zajem in problemi, ki so se pojavili.

2.2 Opisovanje podatkov

V tem opravilu raziskujemo površinske lastnosti podatkov.

Izhod je *poročilo opisa podatkov*, ki vključuje identifikacijo atributov, format in količino podatkov (npr. število zapisov in atributov v vsaki tabeli).

2.3 **Raziskovanje podatkov**

To opravilo naslavlja analitična vprašanja z uporabo poizvedb, vizualizacij in poročanja. Vključuje prepoznavanje ključnih atributov (npr. ciljnih atributov), relacij med atributi, rezultate preprostih agregacij, lastnosti značilnih podskupin in preproste statistične analize.

Izhod je *poročilo raziskave podatkov*, ki opisuje rezultate tega opravila. Vključuje prve ugotovitve in začetne hipoteze ter njihov vpliv na preostanek projekta. Po potrebi vključimo še diagrame in skice, ki bi lahko dali namig, katere podmnožice bi bilo smiselno še dodatno raziskati.

2.4 **Preverjanje kvalitete podatkov**

Tu preučujemo kvaliteto podatkov z vprašanji kot so: Ali so podatki popolni? Ali pokrivajo vse zahtevane primere? So pravilni ali vsebujejo napake? Kako pogoste so? Ali so v podatkih manjkajoče vrednosti? Kako pogoste so? Kako so predstavljene? Kje se pojavljajo?

Izhod je *poročilo kvalitete podatkov*, ki prikaži rezultate preverjanja kvalitete podatkov. Če obstajajo problemi glede kvalitete podatkov, navedemo možne rešitve, kar je odvisno od podatkov in predznanja.

CRISP-DM priprava podatkov

3.1 **Izbira podatkov**

Pri izbiri podatkov se odločamo, katere podatke bomo uporabili za analizo. Kriteriji so primernost podatkov za cilje podatkovnega rudarjenja, kvaliteta podatkov in tehnične omejitve (omejitve glede obsega podatkov, podatkovni tipi). Ta korak pokriva tako izbiro atributov kot izbiro primerov.

Izhod je *utemeljitev za vključitev/izključitev*, ki vsebuje seznam podatkov, ki morajo biti vključeni oz. izključeni in kakšni so razlogi za te odločitve.

3.2 **Čiščenje podatkov**

Pri čiščenju podatkov moramo dvigniti kvaliteto podatkov na nivo, ki ga zahteva izbrana analitična tehnika. To lahko vključuje izbiro čistih podmnožic

podatkov, vstavljanje ustreznih privzetih vrednosti ali bolj ambiciozne tehnike, kot je ocena manjkajočih podatkov z modeliranjem.

Izhod je *poročilo čiščenja podatkov*. To opisuje odločitve in akcije, ki so bile sprejete za reševanje kvalitativnih problemov odkritih med opravilom preverjanja kvalitete podatkov v fazi razumevanja podatkov. Za namen čiščenja je treba upoštevati tudi transformacije podatkov in možen vpliv teh transformacij na analitične rezultate.

3.3 Ustvarjanje podatkov

Ustvarjanje podatkov vključuje operacije za ustvarjalno pripravo podatkov, kot so priprava izvedenih atributov, dodajanje celotnih novih zapisov in dodajanje transformiranih vrednosti obstoječih atributov.

V tem opravilu pridobimo nove (izvedene) atribute, ki so sestavljeni iz enega ali več obstoječih atributov v istem zapisu. Preprost primer za pridobitev novega atributa "površina" je zapisan v formuli 2.1.

$$povrsina = dolzina * sirina \tag{2.1}$$

Izhod tega opravila so tudi na novo ustvarjeni zapisi, ki jih moramo opisati. Primer je kreiranje zapisov za stranke trgovskega podjetja, ki v preteklem letu niso pri njih nič kupile; to je lahko uporabno za modelirne namene.

3.4 Integracija podatkov

Pri integraciji podatkov združimo podatke več tabel ali zapisov v nove zapise.

Izhod tega opravila so *združeni podatki*. Pri združevanju tabel gre za združevanje dveh ali več tabel, ki vsebujejo različne atribute o istih objektih. Pri združevanju je potrebno upariti zapise teh tabel.

Združevanje podatkov lahko vključuje tudi agregacije. To so operacije, pri katerih so nove vrednosti izračunane s povzemanjem informacij iz več zapisov oz. tabel.

3.5 Preoblikovanje podatkov

To opravilo se nanaša na sintaktične spremembe nad podatki, ki ne spremenijo njihovega pomena, vendar jih lahko zahteva modelirno orodje.

Izhod so *preoblikovani podatki*. Nekatera orodja za podatkovno rudarjenje namreč zahtevajo pravi vrstni red atributov (npr. prvi atribut mora biti unikatni identifikator zapisa, zadnji atribut mora biti razredna spremenljivka ipd.). Modelirno orodje lahko tudi zahteva, da so zapisi sortirani glede na vrednost izhodnega atributa.

Včasih so podatki urejeni, modelirni algoritem pa potrebuje naključni vrstni red teh podatkov. Poleg tega so včasih potrebne sintaktične spremembe za zadostitev zahtev modelirnega orodja (npr. odstranjevanje vejic znotraj tekstovnih polj v CSV datotekah, skrajševanje vrednosti na največ 32 znakov ipd.).

V praktičnem delu naloge se bomo še vrnili na 2. in 3. korak CRISP-DM z vidika uporabniškega priročnika. Iskali bomo najprimernejša orodja za izvajanje teh korakov.

2.2 KDD

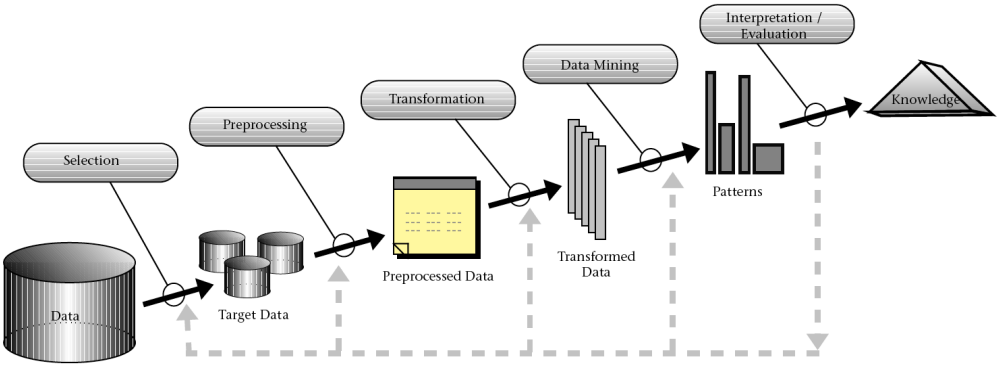
KDD je prav tako kot CRISP-DM procesni standard. Samo podatkovno rudarjenje je le ena izmed njegovih procesnih faz.

KDD je netrivialen proces zaznavanja veljavnih, novih, potencionalno uporabnih in razumljivih vzorcev v podatkih[12]. Vključuje uporabo podatkovne baze skupaj z izbiro, predprocesiranjem, vzorčenjem in transformacijami podatkov. Celoten proces vključuje vrednotenje in možno interpretacijo najdenih vzorcev, da ugotovimo, katere vzorce lahko smatramo za novo znanje[12].

KDD proces je interaktiven in ponavljajoč – ima več korakov, odločitve o njihovih ponovitvah pa sprejema uporabnik. Osnovni koraki KDD, ki so prikazani na sliki 2.3, so:

1. **Razvoj in razumevanje aplikacijske domene**, pridobivanje ustreznega predhodnega znanja ter identifikacija ciljev KDD procesa z vidika stranke.

2. **Kreiranje ciljne podatkovne množice** – izberemo podatkovno množico oz. podmnožico primerov in atributov.



Slika 2.3: KDD proces (Vir: www.infovis-wiki.net)

3. **Čiščenje in predprocesiranje podatkov** – odstranimo ali preučimo šum v podatkih. V tem koraku se tudi odločimo o strategiji ravnanja z manjkajočimi podatki. Izdamo kalkulacije za časovno sekvenčne podatke.

4. **Redukcija in projekcija podatkov** – iščemo atribute, ki so uporabni za končni cilj naloge. Zmanjšamo tudi dimenzionalnost podatkov ali uporabimo ustrezne transformacijske metode.

5. **Ujemanje ciljev KDD procesa (iz 1. koraka) z določeno metodo podatkovnega rudarjenja** (npr. klasifikacija, regresija, napovedovanje razrednih spremenljivk).

6. **Raziskovalna analiza ter izbira modela in hipoteze** – izberemo metode in algoritme. Odločimo se, kateri modeli in parametri bi lahko bili primerni za uporabo.

7. **Podatkovno rudarjenje** – v podatkih iščemo vzorce, ki nas zanimajo.

8. **Interpretacija ugotovljenih vzorcev** – korak vključuje tudi vizualizacijo podatkov glede na ugotovljen model. Možno je vračanje na prejšnje korake.

9. **Ukrepanje glede na odkrito znanje** – odkrito znanje uporabimo direktno, ga vgradimo v drug sistem ali pa odkrito znanje dokumentiramo in

predstavimo. V tem koraku se tudi preverjajo in rešujejo morebitni konflikti s prej ugotovljenim znanjem.

2.3 SEMMA

Tudi SEMMA je procesni standard podatkovnega rudarjenja. Čeprav naj bi bil neodvisen od same izbire orodja, je povezan s produktom *SAS Enterprise Miner*. Sestavlja ga 5 glavnih faz, ne vključuje pa faze razumevanja problema:

1. **Vzorčenje** – izbiramo podmnožico podatkov, ki je dovolj velika, da vsebuje ključne informacije, in jo še vedno lahko učinkovito upravljamo.
2. **Raziskovanje** – iščemo povezave med atributi, nepričakovane trende in anomalije. Uporabimo lahko tudi vizualizacijo, da bolje razumemo podatke.
3. **Spreminjanje** – ustvarjamo, izbiramo in preoblikujemo atribute ter se pripravimo na modelirno fazo.
4. **Modeliranje** – uporabljamo tehnike podatkovnega rudarjenja za ustvarjanje modelov.
5. **Ocenjevanje** – ocenjujemo rezultate modeliranja z vrednotenjem uporabnosti in zanesljivosti ugotovitev.

2.4 Splošno ogrodje za podatkovno rudarjenje, induktivne podatkovne baze in induktivne poizvedbe

Splošno ogrodje za podatkovno rudarjenje[6] je presek podatkovnega rudarjenja in podatkovnih baz. Gre za poskus poenostavitve celotnega procesa podatkovnega rudarjenja na način, da bi se vsi koraki izvajali znotraj induktivnih podatkovnih baz, koraki pa bi se lahko med seboj prepletali.

Induktivne podatkovne baze poleg shranjevanja in manipulacij podatkov omogočajo tudi kreiranje vzorcev in modelov. Te ustvarimo in z njimi manipuliramo s

pomočijo induktivnih poizvedb, ki so neke vrste razširjene SQL poizvedbe. Ustvarjene vzorce in modele lahko shranimo kot podatke v običajnih podatkovnih bazah. Rezultate induktivnih poizvedb lahko uporabimo tudi kot vhod v druge induktivne poizvedbe.

Induktivne podatkovne baze poznajo več tipov poizvedb glede na to, ali se poizveduje po podatkih, vzorcih, ali njihovih kombinacijah. Naslavljajo tudi specifikacijo jezika vzorcev in omejitev, ki jim mora vzorec zadostiti.

Glavna pomanjkljivost tega pristopa je pomanjkanje praktičnih produktov, ki bi jih lahko uporabili kot induktivne podatkovne baze. Je le teorija in nekaj praktičnih pristopov, ki niso integrirani v SUPB ali orodja za podatkovno rudarjenje. Primer poskusa takšne integracije je pristop "*Mining views*", ki ne poskuša razširjati poizvedovalnega jezika, ampak le razširiti shemo podatkovne baze z novimi tabelami, ki vsebujejo opise modelov[13]. Pristopu induktivnih podatkovnih baz manjkajo tudi formalizmi za predstavitev najdenih vzorcev in modelov.

Še en primer, kjer je za izdelavo modelov uporabljen prilagojen SQL jezik, je BayesDB – v času pisanja naloge še nedokončan produkt, ki za izdelavo modelov in poizvedovanje uporablja jezik BQL. Vendar pa ne podpira celotnega procesa podatkovnega rudarjenja. Podobno je z DMX pri Microsoft's SQL Server Analysis Services.

Poglavje 3

Standardi za datotečni prenos podatkov

Čeprav se naloga osredotoča na pripravo podatkov za podatkovno rudarjenje z uporabo podatkovnih baz in ETL orodij, pa pri pridobivanju in izmenjavi podatkov ne moremo mimo podatkov shranjenih v standardnih tekstovnih ali binarnih formatih.

Standardni podatkovni formati omogočajo izmenjavo podatkov med računalniki, ne glede na to, kateri operacijski ali datotečni sistem je v uporabi. Podatki, ki so ustvarjeni v operacijskem sistemu A na datotečnem sistemu X, morajo biti enako uporabni tudi v operacijskem sistemu B na datotečnem sistemu Y. Če se prijavimo v kakšnega izmed spletnih mest, kjer potekajo tekmovanja iz podatkovnega rudarjenja (npr. Kaggle), lahko vidimo, da so vsi podatki na voljo v obliki standardnih podatkovnih formatov, v tekstovni ali binarni obliki. V dokumentu priporočil odprtih podatkov[14] je narejen zelo kratek pregled nad podatkovnimi fomati, vendar z vidika odprtih podatkov, ne pa toliko z vidika podatkovnega rudarjenja, zato sem v tej nalogi nekatere izmed teh formatov izpustil in dodal druge formate, glede na lastno oceno primernost za podatkovno rudarjenje in glede na to kako pogosto sem jih srečal pri iskanju testnih podatkov.

Najbolj znane oblike tekstovnih formatov so CSV, JSON in XML. Veliko SUPB omogoča uvažanje in izvažanje podatkov v obliki CSV datotek. JSON in XML datoteke pa so zaradi svoje hierarhične strukture v uporabi še na enem področju:

pri spletnih storitvah kot REST oziroma SOAP. Tudi spletne storitve so lahko način kako priti do podatkov.

Binarnih tekstovnih formatov je veliko. Nekateri so lastniški, drugi odprtokodni. Dokument priporočil odprtih podatkov [14] priporoča uporabo odprtih (ker so specifikacije zastonj in dostopne vsakemu) in strojno berljivih podatkvnih formatov. Nekateri formati niso uporabni samo za izmenjavo, ampak tudi za hranjenje podatkov. Primeri takšnih tipov binarnih datotek so HDF, NetCDF, Octave in Matlab.

Tudi SUPB ponavadi shranjujejo podatke v binarni obliki, le da je v teh binarnih datotekah še veliko drugih podatkov (uporabniki, dovoljenja, funkcije ipd.). Poleg tega je lahko v eni binarni datoteki več podatkovnih baz. Lahko pa je ena podatkovna baza razdeljena v več datotek in se hrani na več strežnikih. Podatki podatkovnih baz so ponavadi shranjeni v lastniških formatih in so lahko vezani na neko arhitekturo, na kateri je postavljen SUPB, zato da je dostop do podatkov čim bolj učinkovit. Binarne datoteke, ki jih uporabljajo SUPB zato niso primerne za izmenjavo podatkov. Lahko pa podatke iz teh podatkovnih baz ponudimo kot dostop do SUPB z omejenimi pravicami, kot izvoz SQL datotek, ali pa prek spletnih storitev.

Nekateri formati so mešanica binarnih in tekstovnih podatkov. Tako so npr. LibreOffice Calc datoteke le ZIP arhivi XML datotek (podatki, metapodatki, opisi oblik), poddirektorijev in binarnih datotek. Podobno je z novejšimi Excel-ovimi formati.

Vsi binarni formati niso primerni za izmenjavo podatkov. Tako je npr. PDF zelo primeren za izmenjavo in tiskanje dokumentov, za hranjenje numeričnih podatkov in strojno uporabo podatkov pa ne. Obstajajo knjižnice (npr. PDF miner), ki omogočajo pridobivanje podatkov tudi iz PDF datotek, vendar takšen način pridobivanja podatkov ni praktičen.

Na izbiro primernega podatkovnega formata za izmenjavo (ali hranjenje) vpliva tudi vrsta podatkov. Slikovnih podatkov verjetno ni smiselno shranjevati v JSON datoteke. Za geografske podatke pa je primerno izbrati enega izmed formatov geografskih podatkov, npr GeoJSON.

Namen poglavja ni predelati vseh možnih tekstovnih in binarnih formatov, ker presega okvir te naloge, ampak le narediti kratek pregled nad najbolj pogostimi

formati. Če se ne ukvarjamo z geografskimi podatki, potem nam formati, kot so ADRG, ECW, RPF, DTED, verjetno nič ne povedo. Prav pa je, da poznamo osnovne tekstovne formate[1] in tiste binarne formate, ki se uporabljajo za izmenjavo podatkov. Ker drugi korak CRISP-DM zajema tudi uvažanje podatkov in ker se bomo s takšnimi podatkovnimi formati še srečevali, je v tem poglavju narejen kratek pregled najbolj pogostih tekstovnih in binarnih formatov za izmenjavo podatkov.

3.1 Tekstovni formati

Tekstovne datoteke so pravzaprav binarne, le da so predstavljene v ASCII formatu. Vsebujejo vrstice[2], na koncu pa je znak za konec datoteke. Preproste tekstovne datoteke ne vsebujejo metapodatkov. Datoteke so lahko tudi prazne (velikost 0 bajtov).

Pri tekstovnih datotekah moramo biti poleg samega formata pozorni na znakovni kod in način kodiranja.

Kodni nabori tekstovnih datotek

Pri tekstovnih datotekah moramo paziti na kodni nabor. Neupoštevanje tega se odraža v nepravilni pretvorbi šumnikov in drugih znakov. Črke, številke in znaki so predstavljeni s kombinacijami bitov. Tem kombinacijam pa daje pomen kodna tabela posameznega kodnega nabora. Različni kodni nabori uporabljajo različno število bitov za zapis enega znaka. Nekateri (npr. UTF-8) omogočajo zapis večjega števila znakov kot drugi (npr. ASCII), vendar za zapis porabijo tudi več bitov.

Najbolj znan kodni nabor je 7-bitni **ASCII**[3], ki se uporablja povsod, kjer ne potrebujemo posebnih znakov, potrebujemo pa zanesljiv pomen osnovnih znakov[4]. ASCII definira 128 znakov: 32 kontrolnih in 96 izpisljivih znakov.

7-bitni ASCII razširja 8-bitno kodiranje. Prvih 128 znakov je enakih ASCII,

[1] Na osnovi teh formatov so narejeni tudi drugi, bolj specifični formati, npr. iz JSON sta nastala GeoJSON in JSON-LD.

[2] Znaki za konec vrstice so v različnih okoljih različni.

[3] ASCII tabelo lahko najdemo na naslovu http://www.asciitable.com

[4] npr. HTTP protokol

preostalih 128 pa določa kodni nabor. Šumnike podpirajo ISO-8859-2 (Latin 2), CP852 (DOS Latin 2) in CP1250 (Windows-1250).

Ker različni 8-bitni kodni nabori med sabo niso združljivi, so se pojavili nabori, kjer so znaki opisani z več kot enim bajtom. Zaradi združljivosti se je pojavila družina **unicode** kodnih naborov, s pomočjo katerih lahko v enem dokumentu uporabimo znake več jezikov. Ti skušajo za vsak znak, ki obstaja v svetu, določiti neko številčno vrednost. Za kodiranje posameznega znaka pa se uporabi med 1 in 6 bajtov. Črka "c" tako zaseda en bajt, šumnik "č" pa dva bajta. Unicode uporablja 17 kodnih področij, kjer ima vsako 65536 znakov.

Najbolj znana unicode kodna nabora sta UTF-8 in UTF-16. Pri **UTF-8** se prvih 128 znakov kodira z enim bajtom (enako ASCII), 1920 znakov (grški, cirilični, koptski, armenski, hebrejski in arabski) z dvema bajtoma, 63488 znakov (jeziki različnih narodov) s 3 bajti, preostalih 2147418112 znakov pa s 4, 5 ali 6 bajti. Ker je vseh možnih znakov veliko, vsi še niso zasedeni.

Pri UTF-16 je prvih 65536 znakov predstavljenih z dvema bajtoma, preostali pa so predstavljeni s štirimi. Najpreprostejši UTF-32 za vsak znak porabi natanko 4 bajte. Drugi unicode kodni nabori (UTF-7, UTF-32, UCS-4 ...) so manj pogosti.

Za pravilno interpretacijo podatkov izven obsega 7-bitnega ASCII moramo poznati njihov kodni nabor. Zato je zaželjeno, da je v sami podatkovni datoteki navedeno kateri kodni nabor je uporabljen.

CSV

CSV je zelo pogost format, ki pa nikoli ni bil formalno dokumentiran. Tudi IANA MIME registracijsko drevo[5] ne vsebuje tipa za CSV, vsebuje pa tip "text/tab-separated-values", kar je navedeno tudi v IETF RCF4180. Obstaja več specifikacij in implementacij CSV, a formalna specifikacija ne obstaja.

Večina implementacij CSV ima naslednje značilnosti:

- Vsak zapis je v svoji vrstici, zadnja vrstica je lahko izjema.
- Opcijsko lahko vsebuje zaglavno vrstico, ki je prva vrstica v datoteki. Ta navaja imena atributov, ki morajo biti v enakem vrstnem redu kot podatki,

[5] http://www.iana.org/assignments/media-types/media-types.xhtml

ki sledijo.

- Atributi so med seboj ločeni z vejico, vsaka vrstica pa mora imeti enako število polj. Zadnjemu polju ne sme slediti vejica.
- Presledki so del podatkov in jih ne smemo ignorirati.
- Vsak atribut je lahko obdan z dvojnimi narekovaji, a nekateri programi jih ne znajo uporabljati. Če atribut vsebuje znak za novo vrstico, dvoji narekovaj[6] ali vejico, mora biti obvezno obdan z dvojnimi narekovaji.

CSV format se pogosto uporablja za uvažanje podatkov v SUPB. Pri delu s CSV je zelo uporabno orodje **csvkit**, ki je opisano v poglavju 7.1.

Primer CSV datoteke:

```
sepallength,sepalwidth,petallength,petalwidth,class
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
```

TSV

TSV je alternativa formatu CSV. Na kratko ga je specificirala IANA[7], registriran pa je tudi kot MIME tip "text/tab-separated-values". Atributi so ločeni s tabulatorjem. Ta znak pa je znotraj vrednosti atributa zaradi enostavnosti formata prepovedan. Vsak zapis je v svoji vrstici, enako kot pri CSV. Vsaka vrstica mora vsebovati enako število polj. Prva vrstica vsebuje imena polj, vendar je v nekaterih primerih izpuščena. Primer TSV datoteke[20]:

```
Name    Age     Address
Paul    23      1115 W Franklin
Bessy   5       Big Farm Way
Zeke    45      W Main St
```

[6] Te se zapiše kot dva dvojna narekovaja.

[7] https://www.iana.org/assignments/media-types/text/tab-separated-values

Podatke v TAB datoteki lahko pregledujemo v programih kot je MS Excel ali LibreOffice Calc. Pregledovanje in urejanje v tekstovnih urejevalnikih je ponavadi nepraktično, še posebej, če imajo podatki veliko število atributov ali pa so širši od dolžine tabulatorja, ker podatki istih atributov niso več prikazani eden pod drugim.

XML

XML datoteke se zaradi svojega bogatega in fleksibilnega formata pojavljajo tudi na področju podatkovnega rudarjenja. Z njimi lahko naredimo bogatejšo predstavitev podatkov kot pri formatih, ki uporabljajo tabelarični zapis (CSV, TSV ipd.).

Na XML dokumente lahko gledamo kot na urejena drevesa. Na začetku je definicija XML dokumenta, sledi pa korensko vozlišče. Vsa ostala vozlišča so potomci tega vozlišča in tudi sami imajo lahko potomce. Potomci so navedeni v nekem vrstni redu. Vsako vozlišče ima oznako, lahko pa vsebuje tudi poljubno število atributov. V končnih vozliščih (v listih drevesa) je lahko besedilo, ali pa kakšna bolj zahtevna vsebina (npr. slike). Za XML dokumente je lahko predpisana struktura v XML shemi. Candillier in ostali [17] opisujejo XML dokumente kot polstrukturirane – imajo neko strukturo, vseeno pa so bolj fleksibilni kot strukturirane podatkovne baze.

Za podatkovno rudarjenje in združevanje podatkov z drugimi viri bi bilo najbolj enostavno, če bi lahko vsebino XML pretvorili v tabelarično strukturo. Vendar pa tega – razen pri najbolj enostavnih XML dokumentih – ne moremo storiti brez izgub. Pri naslednjem primeru bi lahko podatke spravili v tabelarično strukturo, le atribut “id” bi morali spremeniti v element.

```
<?xml version="1.0" encoding="UTF-8"?>
<flowers>
        <flower id="1">
                <sepallength>5.1</sepallength>
                <sepalwidth>3.5</sepalwidth>
                <class>Iris -setosa</class>
        </flower>
        <flower id="2">
                <sepallength>7.0</sepallength>
```

```
                <sepalwidth>3.2</sepalwidth>
                <class>Iris -cersicolor</class>
        </flower>
</flowers>
```

Nekateri algoritmi znajo delati direktno nad XML dokumenti. Candillier in ostali [17] opisujejo načine, kako lahko poenostavimo XML dokument. Pri nekaterih algoritmih nas zanima le struktura dokumenta, pri drugih pa vsebina. To vpliva na potek poenostavitve XML. V vsakem primeru pa poskušamo odstraniti dele XML, ki za naše cilje niso pomembni. Poenostavitve vključujejo naslednje transformacije:

- ignoriranje vrstnega reda potomcev,
- odstranjevanje nepotrebne tekstovne vsebine,
- odstranjevanje nepotrebnih atributov oz. pretvarjanje atributov v elemente,
- odstranjevanje nepomembnih nižjih nivojev drevesa,
- odstranjevanje nepomembnih besed: predlogov, mašil ... (an. stop worlds, npr. členek "the").

Za poizvedovanje po XML dokumentu je v uporabi jezik XPath[8]. Z njim lahko pridobimo vsa vozlišča, ki ustrezajo nekemu pogoju. Npr.:

```
/flowers/flower[@id='1']
```

Nekatere SUPB imajo vgrajeno podporo za XML dokumente. Tako ima npr. PostgreSQL podporo za podatkovni tip XML, ki pri shranjevanju preverja, ali je XML pravilno strukturiran. Poleg tega ima tudi funkcijo *xpath*, prek katere lahko opravljamo poizvedovanje po XML dokumentu. Ne podpira pa indeksiranja atributov tipa XML.

Tudi Microsoft SQL Server podpira podatkovni tip XML, kamor lahko shranjujemo XML dokumente ali njihove dele. Za razlilko od PostgreSQL pa omogoča pa tudi indeksiranje XML atributov, poizvedovanje po več XML dokumentih z uporabo XPath, masovno uvažanje XML idr.

[8]Standard je dostopen na naslovu: http://www.w3.org/standards/techs/xpath

JSON

Za izmenjavo podatkov je XML primeren format, a ima nekaj pomanjkljivosti. Označbe zavzamejo veliko prostora in povečajo velikost datoteke. Aplikacije, ki uporabljajo XML, morajo vedeti, kako jih razčlenjevati.

JSON je bolj jedrnat, nima označb, in zato porabi manj prostora. Ne ukvarja se s tem, kaj je atribut in kaj element, ne uporablja imenskih prostorov, ne pozna XML sheme. Uporablja zavite oklepaje, dvopičja in dvojne narekovaje. JSON struktura nima nujno korenskega objekta. JSON je naravna predstavitev podatkov za programske jezike iz družine C. V JavaScriptu se lahko JSON uporabi direktno, brez interpreterja. Primer JSON strukture:

```
{
  "država": {
    "naziv": "Slovenija",
    "prazniki": [
      {"ime": "božič", "mesec": 12, "dan": 25},
      {"ime": "novo leto", "mesec": 1, "dan": 1}
    ]
  }
}
```

Enako strukturo bi v XML formatu predstavili takole:

```
<država naziv="Slovenija">
  <prazniki>
    <element>
       <ime>božič</ime>
       <mesec>12</mesec>
       <dan>25</dan>
    </element>
    <element>
       <ime>novo leto</ime>
       <mesec>1</mesec>
       <dan>1</dan>
    </element>
  </prazniki>
</država>
```

Pri XML lahko smo za poizvedovanje uporabljali XPath. Pri JSON glede poizvedovanja ni jasnega standarda. Nekateri jeziki imajo sintakso za dostop do elementov JSON že vgrajeno.

Avtor članka o **JSONPath** *Stefan Goessner*[23] se sprašuje, ali je potreba po poizvedovalnem jeziku za JSON in kakšen naj bo. Za JSONpath, ki ni tako bogat kot XPath, ni uradnega standarda, vendar je že nekaj knjižnic, ki ga podpirajo.

Še bolj dodelan kot jezik JSONpath je jezik **JSONiq**, ki je uporaben za poenostavljanje[9] JSON strukture. JSONiq omenjajo tudi kot SQL za nerelacijske SUPB, ki za shranjevanje vrednosti uporabljajo JSON strukture. JSONiq ima razširitev, ki omogoča, da z istim jezikom poizvedujemo tudi po XML.

Direktno podporo za JSON ima jezik R z uporabo paketa **rjson**. Še en format, ki ga lahko na tem mestu omenimo, je **MessagePack**. Gre za serializacijski format, s katerim lahko zmanjšamo velikost JSON datotek in pospešimo razčlenjevanje vsebine.

Podpora za JSON je vgrajena v nekatere SUPB. PostgreSQL pozna podatkovna tipa **json** in **jsonb**. Pri njunem shranjevanju preverja pravilnost JSON strukture. Tip *json* uporabimo, če hočemo da se pri shranjevanju ohrani struktura dokumenta (npr. presledki za zamik), sicer je priporočljivo uporabiti *jsonb*. Pri njem se JSON struktura pretvori v binarni format, kar upočasni uvoz, vendar precej pospeši procesiranje. Omogoča poizvedovanje o tem, ali je nek JSON dokument vsebovan v drugem. Poleg tega omogoča tudi indeksiranje, s čimer lahko iščemo po ključih JSON struktur v več JSON dokumentih hkrati. Primer poizvedbe:

```
SELECT jsonattr#>>'{drzava,prazniki}' AS prazniki
FROM drzave
WHERE jsonattr#>>'{drzava, naziv}'='Slovenija'
```

Microsoft SQL Server v verziji 2014 podpore za JSON nima, vendar naj bi v verziji 2016 dodal to podporo[18]. Na žalost tudi v tej verziji ne bo podpiral podatkovnega tipa JSON, niti ne indeksiranja. JSON bo shranjen samo kot niz. Bo pa pa nova verzija prinesla razširitve za jezik T-SQL, kot je npr. JSON PATH.

[9]Npr. uporaba agregacijskih funkcij.

YAML

Čeprav je YAML nekoliko zapostavljen format, je prav, da poleg JSON omenimo tudi njega. Nanj lahko gledamo kot na neko nadmnožico JSON, saj lahko v enovrstičnem načinu YAML zapišemo JSON.

JSON je zelo kompakten format, še posebej, če ga kodiramo kot MessagePack, vendar je neoblikovan JSON manj praktičen za pregledovanje in urejanje – iskati moramo ujemajoče oklepaje. Za ročno urejanje je YAML bolj primeren. Gnezdenje strukture temelji na zamikanju vrstic s presledki, podobno kot v programskem jeziku Python. YAML ima nekaj več pravil kot JSON, zato navajam le najpomembnejša:

- YAML datoteka lahko vsebuje več dokumentov, vsak se začne s tremi vezaji.
- Enovrstični komentarji se začnejo z znakom #, večvrstični niso podprti.
- V dokumentu lahko navedemo pare ključ-vrednost[10], sezname ali preslikave.
- Sezname in preslikave lahko navedemo v enovrstičnem ali večvrstičnem načinu, npr.

```
# Enovrstični seznam
[x, y]
# Večvrstični seznam
- x
- y

# Enovrstična preslikava
{ x:1, y:2 }
# Večvrstična preslikava
x: 1
y: 2
```

- YAML pozna več tipov podatkov kot JSON.
- YAML omogoča sklicevanje na druge dele dokumenta. Na vozlišča, ki jim pri vrednosti navedemo oznako "*&oznaka*" se sklicujemo z "**oznaka*".

[10] V obliki "ključ: vrednost". Vrednosti ni treba dati v narekovaje.

- Za večjo berljivost lahko dodajamo prazne vrstice.

Eriksson in Hallberg [24] v svoji nalogi primerjata JSON in YAML. Ugotavljata, da je kreiranje dokumenta in luščenje podatkov precej hitrejše pri JSON, verjetno zaradi kompleksnosti YAML formata, pri velikosti dokumentov sta si podobna, YAML pa je bolj berljiv. Zato naj bi bil JSON priporočljiv povsod, kjer je pomembna hitrost, pri kompleksnih strukturah in funkcionalnostih, ki jih JSON ne podpira, ter za ročno urejanje podatkov pa YAML.

Primer YAML datoteke:

```
---
- oseba: &001
  ime: Marko
  kratice: [M, MK]
  rojen:
    kraj: MB
    leto: 1985
- oseba: &002
  ime: Janez
  kratice: [J, JN]
  rojen: {kraj: LJ, leto: 1973}
  prijatelji: [&001]
```

ARFF

ARFF je tekstovni format, ki je bil razvit za uporabo z orodjem Weka. Ima dva ločena dela: *zaglavje* in *podatki*. Zaglavje vsebuje ime podatkovne množice ter seznam atributov in njihovih tipov [19].

Primer okrnjene ARFF datoteke iz znanega primera klasifikacije cvetov [8, str. 15] zgleda tako:

```
@RELATION iris
@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
```

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
```

ARFF pozna tri tipe deklaracij: @RELATION, @ATTRIBUTE in @DATA[11]. V prvi vrstici mora biti deklaracija @RELATION, kjer določimo ime podatkovne množice. Sledijo deklaracije atributov, kjer podamo ime in tip atributa. Deklaraciji @DATA sledijo podatki. Vsaka entiteta mora biti v svoji vrstici[12]. Vrednosti atributov so med seboj ločene z vejico, podane pa morajo biti v enakem vrstnem redu kot pri deklaraciji atributov. Če ime relacije, ime atributa, ali vrednost atributa vsebuje presledke, mora biti v narekovajih. Manjkajoče vrednosti so predstavljene z znakom "?".

ARFF sicer dopušča prazne vrstice, vendar nekateri uporabniki omenjajo, da prazne vrstice med deklaracijama @RELATION in @ATTRIBUTE ali med deklaracijami @ATTRIBUTE povzročajo napake pri branju datoteke. Če se vrstica začne z znakom "%", gre za komentar. ARFF pozna naslednje tipe podatkov:

- **numerični** (*NUMERIC*) – realna ali cela števila,
- **nominalni** – seznam možnih vrednosti navedemo v zavitih oklepajih, npr. {*Iris-setosa,Iris-versicolor,Iris-virginica*},
- **znakovni** (*STRING*) – nizi, ki vsebujejo presledke morajo biti v narekovajih,
- **datum** (*DATE*) – deklaraciji opcijsko sledi format; če ni naveden, se uporabi privzeta oblika "yyyy-MM-dd'T'HH:mm:ss", kar je kombinacija ISO-8601 formata za datum in čas.

Vrednosti znakovnih in nominalnih atributov so občutljive na velikost črk.

[11] Imena deklaracij so neobčutljive na velikost črk.

[12] Vrstice so med seboj ločene z znakom $\backslash r$.

ARFF podpira tudi t.i. redko posejane (an. sparse) datoteke. Te so uporabne pri podatkih, kjer je večina vrednosti ničelnih. Podatki z vrednostjo 0 niso eksplicitno predstavljeni, ampak se navedejo le neničelne vrednosti, ki so določene z zaporedno številko atributa [13], presledkom in vrednostjo atributa. Vrstice so obdane z zavitimi oklepaji.

Npr. Podatke z veliko ničelnimi vrednostmi lahko namesto v standardni obliki

```
@data
0, X, 0, Y, "class A"
0, 0, W, 0, "class B"
```

zapišemo v obliki redko posejanih vrednosti

```
@data
{1 X, 3 Y, 4 "class A"}
{2 W, 4 "class B"}
```

Izpuščene vrednosti atributov niso manjkajoče oz. neznane vrednosti (kar predstavlja znak "?"), ampak imajo vrednost 0.

ARFF datoteke lahko pregledujemo s tekstovnim urejevalnikom, še lažje pa z aplikacijo **ArffViewer** (na sliki 3.1), ki je del aplikacije Weka . V tej aplikaciji lahko ARFF tudi urejamo ali pretvorimo v CSV. Na žalost pa je aplikacija počasna – odpiranje 33 MB velike datoteke je trajalo 47 sekund. Pri delu z Arff datotekami lahko uporabljamo orodja Weka, Knime, ali RapidMiner. ARFF lahko z uporabo aplikacije Weka pretvorimo v CSV ali TAB[14].

[13] Štetje se začne z 0.

[14] ARFF lahko pretvorimo tudi v XRFF format, ki je XML oblika ARFF datoteke.

ARFF-Viewer- /home/lado/VIRTUALKE/DATA/005MB_Leukemia/ALL-AML_test.arff

File Edit View

ALL-AML_test.arff *

Relation: deepsTestOriginal

No.	attribute1 Numeric	attribute2 Numeric	attribute3 Numeric	attribute4 Numeric	attribute5 Numeric	attribute6 Numeric	attribute7 Numeric	attribute8 Numeric	attribute9 Numeric	attribute10 Numeric	at
1	-342.0	-200.0	41.0	328.0	-224.0	-427.0	-656.0	-292.0	137.0	-144.0	
2	-87.0	-248.0	262.0	295.0	-226.0	-493.0	367.0	-452.0	194.0	162.0	
3	22.0	-153.0	17.0	276.0	-211.0	-250.0	55.0	-141.0	0.0	500.0	
4	-243.0	-218.0	-163.0	182.0	-289.0	-268.0	-285.0	-172.0	52.0	-134.0	
5	-130.0	-177.0	-28.0	266.0	-170.0	-326.0	-222.0	-93.0	10.0	159.0	
6	-256.0	-249.0	-410.0	24.0	-535.0	-810.0	709.0	-316.0	27.0	14.0	
7	-62.0	-23.0	-7.0	142.0	-233.0	-284.0	-167.0	-97.0	-12.0	-70.0	
8	86.0	-36.0	-141.0	252.0	-201.0	-384.0	-420.0	-197.0	-60.0	-468.0	
9	-146.0	-74.0	170.0	174.0	-32.0	-318.0	8.0	-152.0	-148.0	17.0	
10	-187.0	-187.0	312.0	142.0	114.0	-148.0	-184.0	-133.0	12.0	97.0	
11	-56.0	-43.0	43.0	177.0	-116.0	-184.0	-105.0	-62.0	0.0	-40.0	
12	-55.0	-44.0	12.0	129.0	-108.0	-301.0	-222.0	-133.0	136.0	320.0	
13	-59.0	-114.0	23.0	146.0	-171.0	-227.0	-73.0	-126.0	-6.0	149.0	
14	-131.0	-126.0	-50.0	211.0	-206.0	-287.0	-34.0	-114.0	62.0	341.0	
15	-154.0	-136.0	49.0	180.0	-257.0	-273.0	141.0	-123.0	52.0	878.0	
16	-79.0	-118.0	-30.0	68.0	-110.0	-264.0	-28.0	-61.0	40.0	-217.0	
17	-76.0	-98.0	-153.0	237.0	-215.0	-122.0	-68.0	-118.0	6.0	208.0	
18	-34.0	-144.0	-17.0	152.0	-174.0	-289.0	361.0	-89.0	87.0	104.0	
19	-95.0	-118.0	59.0	270.0	-229.0	-383.0	172.0	-187.0	185.0	157.0	
20	-12.0	-172.0	12.0	172.0	-137.0	-205.0	358.0	-104.0	-25.0	147.0	
21	-21.0	-13.0	8.0	38.0	-128.0	-245.0	409.0	-102.0	85.0	281.0	
22	-202.0	-274.0	59.0	309.0	-456.0	-581.0	-159.0	-343.0	236.0	-7.0	

Slika 3.1: ARFF-Viewer

3.2 Binarni formati

Za razliko od tekstovnih datotek pa binarnih datotek ne moremo pregledovati in urejati v običajnih tekstovnih urejevalnikih, ampak za to rabimo namenske programe ali ustrezne knjižnice.

Binarne datoteke sestavljajo nizi bajtov, ki ne predstavljajo tekstovnih znakov. Lahko pa predstavljajo zapise slik, zvokov, kompresiranih datotek in drugih objektov. Nekatere binarne datoteke vsebujejo zaglavje z metapodatki, ki opisujejo vsebino datoteke. Prvih nekaj bajtov binarnih datotek ponavadi predstavlja podpis, ki določa format datoteke. V UNIX okolju lahko tip datoteke ugotovimo z ukazom “file”, npr.

```
$ file flower.jpg
flower.jpg: JPEG image data, JFIF standard 1.01
```

Nekatere binarne datoteke so lahko brez zaglavja (an. flat binary file). Binarne datoteke lahko kodiramo tudi v tekstovno obliko npr. s kodiranjem Base64. Pregledujemo in urejamo pa jih lahko tudi v t. i. “HEX” urejevalniku kot sekvenco

šestnajstiških (lahko tudi ASCII, decimalnih ali osmiških) vrednosti.

Primer prvih nekaj bajtov slike kot niza šestnajstiških vrednosti zgleda tako:

```
$ hexdump -C flower.jpg
00000000  ff d8 ff e0 00 10 4a 46  49 46 00 01 01 01 00 48  |......JFIF.....H|
00000010  00 48 00 00 ff e1 00 16  45 78 69 66 00 00 4d 4d  |.H......Exif..MM|
00000020  00 2a 00 00 00 08 00 00  00 00 00 00 ff fe 00 17  |.*..............|
00000030  43 72 65 61 74 65 64 20  77 69 74 68 20 54 68 65  |Created with The|
00000040  20 47 49 4d 50 ff db 00  43 00 06 04 05 06 05 04  | GIMP...C.......|
```

HDF

HDF je odprtokoden datotečni format, podatkovni model, knjižnice in orodja. HDF5 nadomešča starejši standard HDF4 in odpravlja nekaj njegovih omejitev. Datoteke formata HDF5 prepoznamo po končnici *h5*.

HDF *format* je prenosljiv in kompakten. Vsi podatki zbirke so v eni sami datoteki, kar je prednost pri prenašanju, lahko pa je pomanjkljivost pri arhiviranju. Format je ustvarjen za velike in kompleksne podatke ter učinkovite vhodno-izhodne operacije. Nima omejitev glede števila in velikosti podatkovnih objektov. Zelo hiter je pri pisanju in branju podatkov[15]. Zelo primeren je za časovne podatke in kjer je podatkov več od velikosti pomnilnika. Format pogosto uporabljamo tudi za prilagoditve, brez katerih bi morali razvijati svoj datotečni format.

HDF5 poenostavlja podatkovno strukturo. Ta pozna le dva tipa objektov, ki jih lahko na nek način primerjamo z datotekami in direktoriji v datotečnem sistemu:

- **Podatkovne množice** – podatkovna množica je množica podatkovnih struktur (sestavljeni tipi) ali osnovnih podatkov (numerični tipi, znakovni tipi, polja) in metapodatkov[16].
- **Skupine** – to so vsebniki, ki vsebujejo podatkovne množice in druge skupine.

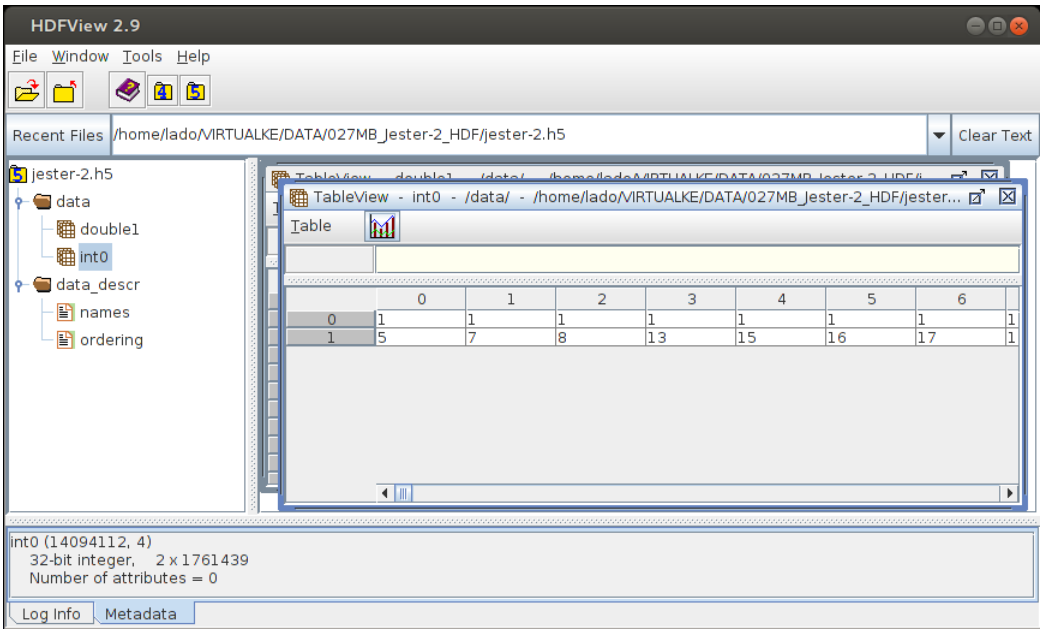
Do posameznih podatkovnih množic in podskupin dostopamo v POSIX sintaksi, npr. "/pot/do/vira/".

HDF ponuja *knjižnice* za programske jezike C, C++, Fortran 90, Java, Python (npr. PyTables, h5py). Vključuje tekstovna in grafična *orodja* za upravljanje,

[15] Še hitrejši je z uporabo kompresije. Podpira pa tudi enkripcijo podatkov.

[16] Metapodatki hranijo opis podatkovnih elementov, obliko podatkov in druge informacije potrebne za interpretacijo podatkov. Definira jih uporabnik.

urejanje, pregledovanje in analiziranje podatkov. Med aplikacijami velja omeniti odprtokodna orodja za vizualizacijo *ParaView* in *VisIt*. Preprosta aplikacija za urejanje in pregledovanje HDF5 je *HDFView*, ki je prikazana na sliki 3.2. Plačljiva orodja, ki imajo vključeno podporo za HDF5 so *IDL*, *Matlab*, *Mathematica*, *EnSight*, *Tecplot*, *FileViewPro* idr. Tudi Octave zna odpirati HDF datoteke.



Slika 3.2: HDFView

NetCDF

NetCDF je skupina prenosljivih podatkovnih formatov, knjižnic in podpornih aplikacij, namenjenih za znanstvene podatke. Razvija in vzdržuje ga Unidata, skupnost izobraževalnih in raziskovalnih ustanov, z namenom izmenjave podatkov geoznanosti[17]. Pri NetCDF lahko učinkovito dostopamo do manjših podmnožic tudi pri velikih podatkih. NetCDF je primeren za shranjevanje časovnih, geografskih, meteoroloških in drugih podatkov. Eno NetCDF datoteko lahko hkrati bere več procesov in vanjo piše en proces, prek OPeNDAP lahko tudi preko omrežja.

[17]Geoznanosti ali znanosti o zemlji so geologija, geofizika, meteorologija, fizikalna geografija, oceanografija, seizmologija, vulkanologija ipd.

NetCDF datoteke imajo končnico *.nc*. Pozna pa tudi interpretacijo podatkov v CDL tekstovni obliki.

NetCDF podpira tri različne binarne formate:

- klasični format (privzet),
- 64-bitni "offset" format,
- NetCDF-4/HDF5, ki je v bistvu HDF5 format z nekaj omejitvami. V glavi NetCDF dokumenta je opis dokumenta in metapodatki v obliki "ime: vrednost".

NetCDF ponuja knjižnice za jezike R, Perl, Python, Ruby, Haskell, Mathematica, MATLAB, IDL, Octave. Med aplikacijami lahko omenimo tekstovni *NetCDF Operators suite*, grafični *ncBrowse*, *Ncview*, *Panoply*, *NCL* jezik.

Matlab datoteke

Matlab za shranjevanje podatkov uporablja binarne datoteke. Te se kreirajo z uporabo funkcije "save". Ta zapiše podatke iz pomnilnika v datoteko. Te datoteke imajo končnico ".mat". Podatke lahko preberemo s funkcijo "load".

Poznamo dve vrsti Matlab datotek, starejša se je uporabljala do verzije 4, novejša pa od verzije 5 naprej. Matlab ohranja kompatibilnost za starejšo obliko. Starejša verzija je podpirala le shranjevanje 2-dimenzionalnih matric in znakovnih nizov. Novejša verzija pa podpira še večdimenzionalne numerične matrice, znakovne matrice, redke matrice, celične matrice, strukture in objekte. Matlab datoteke lahko uporabljamo tudi preko knjižnic za C, Fortran, Java in Python, vendar le v okoljih, kjer je podprt Matlab. Za uporabo Matlab datotek s temi knjižnicami nam binarnega formata Matlab datotek ni potrebno poznati, za potrebe izdelave svojega vmesnika pa je ta binarni format razložen v dokumentu [21].

Preden pri uporabi Matlab[18] naložimo datoteko v pomnilnik, lahko z ukazom "whos" preverimo imena spremenljivk ter njihove dimenzije, velikost in razred oz. tip spremenljivk, npr.

[18]Enako je pri Octave.

```
octave:4> whos -file caltech256-30.matlab
Variables in the file caltech256-30.matlab:

   Attr Name          Size                     Bytes  Class
   ==== ====          ====                     =====  =====
        K          7680x7680                235929600  single
        Ktest      6400x7680                196608000  single
        te_files      1x6400                   167725  cell
        te_label   6400x1                       51200  double
        tr_files      1x7680                   201270  cell
        tr_label   7680x1                       61440  double

Total is 108162560 elements using 433019235 bytes
```

Matlab ponuja kar nekaj funkcionalnosti za delo z velikimi podatki. Med tistimi, ki ne zahtevajo skaliranja strojne opreme v vertikalni ali horizontalni smeri, lahko omenim funkcijo **memmapfile**, prek katere lahko preslikamo datoteko ali del datoteke v rang pomnilniških naslovov znotraj Matlab naslovnega prostora. Tako lahko dostopamo do podatkov, ki so preveliki, da bi jih lahko shranili v pomnilnik. Matlab lahko dostopa do teh podatkov na enak način kot bi dostopal do njih, če bi bili v pomnilniku.

Druga takšna funkcija je **matfile**, ki omogoča direkten dostop do Matlab spremenljivk iz Matlab datotek na disku z uporabo Matlab ukazov za indeksiranje, ne da bi se celotne spremenljivke naložile v pomnilnik. Tako lahko bločno procesiramo podatke, ki so preveliki, da bi se v celoti prebrali v pomnilnik.

Tretja funkcija, ki omogoča dostop do podatkov, ki so preveliki za branje v pomnilnik, je funkcija **datastore**. Poleg datotek lahko uporablja še zbirke datotek kot enotno entiteto ali pa prek *Database Toolbox* dostopa celo do tabel v podatkovnih bazah. V podatkovno bazo lahko piše ali iz nje bere z uporabo SQL stavkov. Simultano lahko dostopa do več baz. Podpira podatkovne baze, ki so dosegljive prek ODBC ali JDBC. Vsebuje pa tudi orodje *Database Explorer*, ki omogoča brskanje po podatkih v podatkovni bazi, tudi brez znanja SQL.

Ker Matlab podpira tudi skaliranje v horizontalni smeri, ker ima veliko funkcij za procesiranje slik in vsebuje podporo za podatkovno rudarjenje, je uporaben tudi za podatkovno rudarjenje v širšem smislu.

Octave datoteke

Octave je zastonj alternativa programu Matlab. Razpoložljiv je pod GNU GPL licenco. Tako Octave kot Matlab se že dolgo uporabljata za analizo podatkov, modeliranje in tudi strojno učenje[22].

Poleg širokega nabora vgrajenih funkcij, ponuja Octave še veliko dodatnih paketov, med drugim tudi paket za glajenje šumnih podatkov (an. data-smoothing), vmesnik za dostop do PostgreSQL podatkovne baze, orodje za manipulacijo podatkov podobno R "dataframe", paket za pridobivanje podatkov in shranjevanje v različne formate (io), orodje za statistično in strojno učenje (nan), statistične funkcije, orodje za analizo časovnih podatkov (tsa) idr.

Octave je tako kot Matlab zelo uporaben zaradi svojih numeričnih, statističnih, matematičnih in grafičnih funkcij. Oba sta še posebej uporabna pri delu z matrikami. Kdor je delal z Matlab in Octave, ve, da sta lahko pri nekaterih problemih ali pri nepravilni uporabi precej počasna. Zato sta uporabna za hitro prototipiranje, pri komercialnih produktih pa se redko uporabljata. Zato se prototipi izdelani v Matlab/Octave pogosto prevedejo v hitrejši prevajalni jezik, npr. C. Nabor vgrajenih Octave funkcij lahko[19] razširimo z dodajanjem C, C++ ali Fortran programov.

Octave lahko bere in piše TSV datoteke s funkcijami *dlmread* oz. *dlmwrite*, CSV datoteke z uporabo *scvread* in *csvwrite*, lahko pa z uporabo funkcij *load* in *save* shranjuje podatke v binarne datoteke Octave ali Matlab. Primer shranjevanja matrike v datoteko:

```
A = [ 1:3; 4:6; 7:9 ];
save myfile.mat A
```

Pri shranjevanju lahko podatke dodajamo k obstoječim podatkom. Shranimo jih lahko v tekstovni ali binarni obliki. Octave omogoča tudi shranjevanje podatkov v **HDF5** format. Podatke lahko pri shranjevanju tudi kompresiramo.

Podatki se pri branju preberejo v glavni pomnilnik, npr.:

```
load myfile.mat
A
      -| A =
      -|
```

[19] Z uporabo funkcije *mkoctfile*.

```
-|    1   2   3
-|    4   5   6
-|    7   8   9
```

Pri branju datoteke lahko preberemo le posamezne atribute. Če vsebuje datoteka le numerične podatke, vrne Octave matriko vrednosti, sicer vrne strukturo, ki vsebuje imena spremenljivk.

3.3 Druge vrste binarnih in tekstovnih podatkov

V tem poglavju smo naredili kratek pregled nad najbolj pogostimi podatkovnimi formati, ki se uporabljajo oz. bi se lahko uporabljali za pridobivanje in shranjevanje podatkov pri podatkovnem rudarjenju. Vseh možnih formatov je preveč, zato smo opisali le najbolj pogosto uporabljane.

Pravzaprav bi vse formate lahko razdelili po skupinah, glede na vrste podatkov, kjer se uporabljajo. Omenjeni formati so uporabni za tekstovne, numerične, geografske in časovne podatke. Omenili bi lahko tudi multimedijske podatke: slikovne, video in zvočne. Tudi nad njimi lahko izvajamo podatkovno rudarjenje, vendar je opisovanje teh formatov izven obsega te naloge.

Omenimo pa lahko, da moramo multimedijske vrste podatkov pred uporabo ustrezno transformirati. Za binarne slike izdelamo histograme, poiščemo robove, področja, uporabimo morfološke operatorje, izdelamo PCA analizo, ali jih drugače transformiramo. Pri vektorskih slikah lahko iščemo prekrivanja objektov. 3D slike lahko pretvorimo v več 2D slik. Pri zvokovnih podatkih lahko izdelamo frekvenčne spektre, jih pretvorimo v Mel-ove skale, uporabimo okenske funkcije ipd. Pri časovnih podatkih lahko uporabimo glajenje šuma. Nad PDF dokumenti lahko izvedemo optično prepoznavanje znakov, in še bi lahko naštevali.

Podatki so lahko shranjeni v podatkovni bazi ali v datotekah, v binarnih ali tekstovnih, v eni ali več datotekah. Če so shranjeni v več datotekah, pa orodje za podatkovno rudarjenje ne zna delati z več datotekami hkrati, moramo podatke najprej združiti v skupno datoteko ali jih uvoziti v podatkovno bazo.

Poglavje 4

Pregled primernosti orodij za pripravo podatkov

4.1 Kriteriji za izbiro orodij za pripravo podatkov

Kot je bilo prikazano že na sliki 1.3 na strani 8, lahko za pripravo podatkov za podatkovno rudarjenje uporabljamo različna orodja. Pri večjih količinah podatkov so za to primerna ETL orodja in izvajanje transformacij znotraj SUPB (ELT). Tudi nekatera orodja za podatkovno rudarjenje vsebujejo ETL gradnike ali delujejo v povezavi s SUPB. Da so orodja uporabna, morajo zadostiti nekaterim kriterijem. Orange, orodje za podatkovno rudarjenje, naj bi z verzijo 3.0 dodalo podporo za delo s podatki shranjenimi v SUPB[15]. Vir navaja najbolj pogoste operacije za delo s podatki, ki so jih razvijalci odkrili pri pregledu izvorne kode v verziji 2.7. Večino teh kriterijev lahko navedemo tudi kot kriterije, ki jim mora zadostiti dobro orodje za pripravo podatkov za podatkovno rudarjenje. Dobro orodje bi moralo imeti vgrajeno (ali drugače omogočati):

- osnovne agregacijske funkcije (povprečje, varianca, mediana, najvišja in najnižja vrednost),
- prikaz porazdelitev vrednosti diskretnih in zveznih atributov (vrednosti v kvantilih),

- prikaz povezanosti dveh atributov (kontingenčne in kovariančne matrike),
- filtriranje zapisov na osnovi podanih kriterijev (tudi naključno vzorčenje) in možnost pridobivanja posameznih vrstic,
- izbiro atributov,
- izdelavo izvedenih spremenljivk na podlagi vrednosti ene ali več drugih spremenljivk.

Iščemo torej eno orodje ali kombinacijo orodij na relaciji med podatkovnimi viri, ETL orodji, SUPB in orodji za podatkovno rudarjenje, ki omogoča pripravo podatkov za podatkovno rudarjenje. Prednost ločenih orodij je v tem, da jih lahko uporabimo še za kak drug namen (npr. podatke uvožene v SUPB lahko objavimo na spletu), prednost enega vseobsegajočega orodja pa enostavnost namestitve in večja povezanost med koraki.

4.2 Opis orodij

Kandidati za orodja so torej SUPB, ETL in tista orodja za podatkovno rudarjenje, ki vsebujejo ETL gradnike.

Pri SUPB se postavlja vprašanje kakšne vrste SUPB izbrati. Relacijske ali nerelacijske? Prednost slednjih je visoka hitrost shranjevanja in pridobivanja podatkov, vendar pa imajo ponavadi veliko manj funkcionalnosti, poleg tega pa visoka hitrost pride do izraza šele, ko se izvajajo na več kot enem računalniku. Nerelacijske SUPB lahko razdelimo v 4 skupine[16]:

- SUPB tipa Ključ-vrednost (npr. *Riak*),
- stolpično usmerjene SUPB (npr. *Big Table*, *Cassandra*, *MonetDB*, *Druid*, *HBase*),
- SUPB za shranjevanje dokumentov (npr. *MongoDB*, *CouchDB*),
- SUPB za delo z grafi (npr. *Neo4J*)

Izmed teh vrst so za predpripravo podatkov uporabne le stolpično usmerjene SUPB, ki so najbižje relacijskim SUPB. Zaradi stolpičnega načina shranjevanja

pa so hitre pri izvajanju agregacijskih funkcih in pri analitičnih obdelavah. Struktura, ki je podobna tabelam, so družine stolpcev (an. column families). Podatki iz vsakega stolpca se shranijo v svoji datoteki. Družina stolpcev lahko vsebuje super stolpce – to je slovar, ki vsebuje druge stolpce. Stolpec je terka imena, vrednosti in časovnega žiga. Stolpične SUPB podpirajo dve vrsti poizvedb, po ključu, ali po rangu ključev. Ključ določa lokacijo dejanskega podatka. Podatki se shranjujejo glede na definiran vrstni red.

Preostale vrste nerelacijskih SUPB nimajo podpore za izvajanje agregacijskih funkcij in hitrega filtriranja na podlagi vrednosti v podatkih.

Poleg omenjenih vrst nerelacijskih SUPB bi lahko omenili še SUPB, kjer se vrednosti shranjujejo v obliki polj (an. array). Predstavniki so **rasdaman**, **SciDB (Paradigm4)**[1], **ArrayDB**. V polja lahko shranjujemo slike, statistične podatke, znanstvene meritve. Velikost polj je teoretično neomejena. Omogočajo tudi poizvedovanje. Standardni SQL podira shranjevanje podatkov v obliki polj, vendar samo enodimenzionalna polja.

Prednost relacijskih SUPB je SQL poizvedovalni jezik. Omogočajo enostavno filtriranje zapisov, izbiro atributov, izdelavo izvedenih spremenljivk. Izpolnitev ostalih kriterijev je odvisna od SUPB. Obetavne so tiste SUPB, ki imajo dobro podporo za agregacijske funkcije, prikaz porazdelitev vrednosti in poizvedovanje povezanosti atributov. Najbolj popularne relacijski SUPB[2] so: *Oracle, MySQL, Microsoft SQL Server, IBM DB2, Microsoft Access, SAP Adaptive Server, Teradata, FileMaker, Hive, Informix* in druge.

Obstajajo tudi poskusi združevanja različnih tipov SUPB. Fujitsu je razvil orodje, ki teče na PostgreSQL in ni odvisno od kapacitete pomnilnika. To orodje sproti posodablja vrednosti v stolpično usmerjenih podatkih glede na spremembe v vrstično usmerjenih podatkih[3].

[1] SciDB zaradi kompleksne namestitve in težav pri z namestitvenimi skriptami na žalost nisem uspel preizkusiti.

[2] http://db-engines.com/en/ranking

[3] http://www.postgresql.org/about/news/1573/

Poglavje 5

Izvajanje 2. in 3. faze CRISP-DM na primeru dveh SUPB

V tem poglavju je prikazana primerjava izvajanja opravil 2. in 3. faze CRISP-DM na primeru dveh relacijskih SUPB: PostgreSQL in Microsoft SQL Server.

5.1 Izbira SUPB

PostgreSQL

PostgreSQL sem izbral zato, ker gre za eno najbolj znanih odprtokodnih relacijskih SUPB z najdaljšo zgodovino. Lahko se pohvali z zanesljivostjo, saj se že vrsto let uporablja v različnih produkcijskih okoljih. Je ena izmed najbolj *ANSI/ISO SQL:2008* združljivih SUPB. Je hitra in lahko upravlja s terabajti podatkov. Ima vtičnike za uporabo naravnega jezika, večdimenzionalno indeksiranje, geografske poizvedbe, podatkovne tipe po meri, podporo za več skriptnih jezikov idr.

Microsoft SQL Server

Microsoft SQL Server je precej znana komercialna relacijska SUPB. Ima več različic[1], a le *Express* je zastonj. Plačljive verzije imajo vključeno ETL orodje *"SQL Server Integration Services (SSIS)"* in orodje za izvajanje podatkovnega rudarjenja *"SQL Server Analysis Services (SSAS)"*. Vendar pa moramo pri tem upoštevati ekonomski kriterij – cene plačljivih verzij močno presegajo ceno osebnega računalnika. Naše ciljno področje pa je en sam osebni računalnik, zato je v nadaljevanju uporabljena verzija *Express*, ki nima omenjenih orodij, podatkovne baze pa so pri tej izdaji omejene na 10 GB (pri ostalih 524 PB). Vseeno pa lahko preverimo kako lahko z SQL oz. T-SQL v tem SUPB izvajamo transformacije, ki so del 2. in 3. faze CRISP-DM.

5.2 CRISP-DM 2.1 (Začetno zbiranje podatkov)

5.2.1 Zajemanje podatkov

Zajemanje podatkov je začetni del zbiranja podatkov, ki se pri CRISP-DM izvaja v fazi 2.1. Uvozimo lahko vse podatke ali le izbrane. Omejimo lahko tudi izbor atributov.

Viri zajemanja podatkov so lahko različni. Za uspešen uvoz pa morajo biti podatki v elektronski obliki. Elektronski viri podatkov so tekstovne datoteke (npr. CSV), binarne datoteke (npr. HDF) ali podatkovne baze. Včasih moramo podatke zajemati tudi iz več različnih virov. Za nadaljnjo obdelavo podatkov, je najbolj praktično, da podatke uvozimo v SUPB. Pred uvozom moramo vedeti kaj uvažamo. Pri tem si lahko pomagamo z orodjem za razumevanje podatkov.

Če je podatkovnih virov malo, lahko podatke uvozimo direktno v SUPB. Različni SUPB imajo različno podporo za uvoz podatkov. Pri nekaterih moramo najprej kreirati sheme podatkovnih tabel, druge znajo pri uvozu ugotoviti tipe podatkov in avtomatično kreirati podatkovne tabele. Pogosto je za uvoz podatkov enostavneje uporabiti ETL orodja – z njimi lahko izvajamo tudi druge CRISP-DM

[1] *Enterprise, Business inteligence, Standard, Developer, Web, Express.*

korake.

Če podatke pridobivamo iz zunanjih virov, moramo omeniti **SQL/MED** standard. Ta omogoča, da do zunanjih podatkov dostopamo, kot da bi bili shranjeni v lokalni podatkovni bazi. Za SQL poizvedbo so vidne kot lokalne tabele. Žal pa so v praksi implementacije pogosto pomanjkljive.

PostgreSQL

Pri PostgreSQL moramo pred uvozom podatkov najprej kreirati takšno podatkovno shemo, da se vrstni red in tipi atributov ujemajo z atributi v datoteki, ki jo uvažamo. Pri velikem številu atributov je to zamudno opravilo. Ko je shema kreirana je podatke najenostavneje uvoziti z uporabo stavka COPY, npr.:

```
COPY fruitfly FROM '/tmp/fruitfly.csv'
```

COPY podpira CSV, TSV in lasten binarni format. Uvoz iz binarnega formata je najhitrejši, a je uporaben le pri izmenjavi podatkov med PostgreSQL strežniki. Odvisen je tudi od verzije strežnika. Tekstovni format je podoben TSV, le da je lahko ločilo med polji poljuben znak. Stavku COPY lahko kot parameter navedemo še kodni nabor, znak za narekovaj, ničelni niz ipd. Ob morebitni napaki se uvažanje prekine.

Za uvoz podatkov v PostgreSQL je pogosto enostavneje uporabiti ETL orodje. Če uvažamo CSV datoteko, je zelo uporabno orodje *csvkit*, ki pri uvozu podatkov samo ugotovi tipe atributov, kreira podatkovno shemo in uvozi podatke. Več o njem je napisano v poglavju 7.1. Za geografske podatke lahko uporabimo funkcijo *ogr2ogr*, ki je del *GDAL* knjižnice[2].

Pri uvažanju podatkov v PostgreSQL smo omejeni tudi s številom atributov. Podatkovna tabela ima lahko največ med 250 do 1600 atributi[3]. Za namen naloge je bila napisana PL/PgSQL funkcija, ki preverja to omejitev – nahaja se v prilogi na strani 116. Pri tekstovnih atributih je omejitev števila atributov 1600. V prilogi na strani 115 se nahajajo omejitve števila atributov še za druge SUPB.

PostgreSQL je z verzijo 9.1 omogočil branje po standardu SQL/MED, z verzijo 9.3 pa tudi pisanje. Z zunanjimi podatkovnimi viri se povezujemo prek razširitev

[2] http://www.gdal.org/
[3] Odvisno od tipa atributov.

kot so **Dblink, DBI-Link** in različni **FDW**. Npr. **file_fdw** lahko uporabimo za poizvedovanje po tekstovnih datotekah[4] z SQL stavki. Žal pa moramo pri definiciji oddaljene tabele navesti pričakovano množico atributov[5]. Z uporabo stavka SELECT ... INTO lahko podatke iz oddaljene tabele uvozimo tudi v lokalno tabelo.

Če potrebujemo podatke iz oddaljene PostgreSQL podatkovne baze, lahko uporabimo **Dblink**. Dblink najprej naložimo v PostgreSQL bazo[6] kot razširitev:

```
CREATE EXTENSION dblink;
```

Nato Dblink uporabimo kot funkcijo, ki ji podamo *libpq* povezovalni niz in SQL poizvedbo, ki naj se izvede na oddaljenem PostgreSQL strežniku. Slednji mora biti nastavljen tako, da omogoča TCP/IP povezave iz strežnika iz katerega poizvedujemo. Primer poizvedbe[7]:

```
SELECT * FROM dblink(
  'host=192.168.1.105 port=5432 dbname=cinema user=postgres',
  'SELECT actor_id, name FROM actors'
) As t_external(id integer, igralec text)
ORDER BY igralec ASC;
```

Pri Dblink smo omejeni s številom zapisov, ki jih lahko pridobimo oz. uvozimo. Oddaljeni strežnik namreč poskuša pridobiti celoten rezultat poizvedbe. Ta rezultat nato pošlje na strežnik, ki poizveduje. Slednji prebere celoten rezultat poizvedbe v pomnilnik. Če pričakujemo veliko število vrstic, je bolje, da odpremo povezavo kot kurzor z *dblink_open* in nato naenkrat zajamemo obvladljivo število vrstic. Priročen način Dblink poizvedb je uporaba pogledov (views). Dblink naj bi v prihodnje nadomestil **postgres_fdw**.

Naslednji način pridobivanja podatkov v PostgreSQL je **DBI-Link**. Če Dblink podpira samo povezovanje med PostgreSQL podatkovnimi bazami, pa lahko z DBI-Link dostopamo tudi do drugih podatkovnih baz. In sicer tistih, do katerih lahko dostopamo prek Perl-ove DBI knjižnice[8]. Sam DBI modul se ne zna povezati z

[4] Podprti so isti formati kot za stavek COPY.

[5] Ni potrebno, da se imena atributov ujemajo. Tudi ni potrebno, da se tipi popolnoma ujemajo, če se le da pridobljeno vrednost shraniti v naveden tip.

[6] Ali v template1.

[7] V praksi sem pri tem primeru uporabil še parameter *password*.

[8] DBI-Link je napisan v jeziku PL/PerlU.

nobenim SUPB, zato je pred uporabo potrebno namestiti modul za dostop do ustreznega SUPB, npr. *libdbd-sybase-perl* za *Sybase* in *Microsoft SQL Server*. DBI-Link je v opuščanju, dokumentacija zanj je zelo skromna, zamenjujejo ga FDW.

Zadnji način, ki nadomešča ostale načine pridobivanja zunanjih podatkov v PostgreSQL, so **FDW**. Ti implementirajo standard SQL/MED. Prek FDW lahko iz PostgreSQL dostopamo do drugih relacijskih podatkovnih baz (*PostgreSQL, Oracle, MySQL, Informix, Firebird, SQLite, Sybase, Microsoft SQL server, MonetDB*), ODBC in JDBC virov, do nerelacijskih podatkovnih baz (*CouchDB, mongoDB, Redis, Neo4j, Kyoto Tycoon*), do datotek (CSV, JSON) in drugih podatkovnih virov (*Twitter, LDAP, PGStrom, Apache Hadoop, Amazon S3, VirtDB*, različne spletne storitve). FDW, ki so napisani v jeziku C imajo v mnogih primerih boljše performanse kot Dblink oziroma DBI-Link.

FDW za posamezen SUPB ali drug vir podatkov moramo najprej namestiti. Vsaka vrsta potrebuje svoj FWD. Dober vir namestitev je "PostgreSQL extension network" (PGXN). Ponavadi namestitev zahteva, da PostgreSQL skupaj s FDW prevedemo iz izvorne kode. Obstaja pa tudi PGXN odjemalec, ki olajša nameščanje.

Na tem mestu podajamo primer uporabe *file_fdw* za branje podatkov iz CSV datoteke.

```
CREATE EXTENSION file_fdw;
CREATE SERVER test_server FOREIGN DATA WRAPPER file_fdw;
CREATE FOREIGN TABLE fruitfly (
  partners int,
  itype int,
  thorax int,
  sleep real,
  class int
) SERVER test_server
OPTIONS (
  filename '/tmp/fruitfly.csv',
  format 'csv'
);
SELECT * FROM fruitfly;
```

Pri nekaterih FDW moramo pred kreiranjem oddaljene tabele navesti še preslikavo uporabniškega računa z uporabo stavka "CREATE USER MAPPING".

FDW za PostgreSQL lahko napišemo tudi sami. Eden izmed enostavnejših načinov je uporaba python knjižnice **Multicorn**. Napisane FDW pa lahko uporabimo le za branje podatkov. Tudi performanse so lahko slabe.

Microsoft SQL Server

Pri Microsoft SQL Server je korak zajemanja podatkov enostavnejši. Plačljive verzije vključujejo orodje **SQL Server Integration Services**, ki je pravo ETL orodje. Omeniti velja še orodje **Microsoft SQL Server Migration Assistant**, ki omogoča pretvorbo Oracle, Sybase, MySQL ali Access podatkovne baze v Microsoft SQL Server podatkovno bazo.

V brezplačni verziji[9] je na voljo le čarovnik za uvoz in izvoz podatkov[10]. Z njim lahko pridobimo podatke iz drugih podatkovnih baz, preglednic in tekstovnih datotek. Podpira tiste SUPB, do katerih lahko dostopamo prek ogrodja .NET[11] (MSSQL, Oracle, ODBC, OLE DB). Pri tekstovnih datotekah podpira CSV in TSV. Pri uvozu nam ni potrebno ročno kreirati podatkovne tabele in atributov, saj jih čarovnik sam ustvari. Moramo pa določiti tipe atributov (privzet tip je "varchar"). Pri uvozu moramo izbrati tudi pravilni kodni nabor in podatkovno bazo, kamor se bodo podatki uvozili. Če uvažamo podatke iz drugih SUPB, uvozi čarovnik le definicije tabel in podatke, ne pa tudi indeksov in drugih omejitev (an. constraints).

Tudi Microsoft SQL Server ima omejitev glede števila atributov. Največ jih je lahko **1024**. Izjema so široke tabele oz. redko posejane tabele, ki imajo lahko največ **30000** atributov. Pri tem pa je velikost vrstice omejena na 8019 bajtov – zato mora imeti večina podatkov v vrstici vrednost NULL.

5.2.2 Časovno ali drugo omejevanje podatkov

Omejevanje podatkov se izvaja v CRISP-DM fazi 2.1. Pogosto med vsemi zbranimi podatki potrebujemo le en del teh podatkov. Npr. zanimajo nas le podatki,

[9] Express edition.
[10] SQL Server Import and Export Wizard.
[11] .NET Framework data provider.

ki niso starejši od enega leta; ali pa podatki, ki se nanašajo na neko geografsko območje. Ravno enostavno omejevanje podatkov je ena izmed prednosti relacijskih podatkovnih baz. Omejevanje podatkov je najbolj dodelano pri podatkovnih skladiščih (OLAP kocka), kar presega namen naloge.

PostgreSQL

Za časovno omejevanje podatkov ima večina relacijskih podatkovnih baz dobro podporo. PostgreSQL ima v tem delu dodatno prednost v tem, da omogoča uporabo aritmetičnih operatorjev $+$, $-$, $*$ in $/$. Datumu lahko enostavno prištejemo teden, npr.:

```
SELECT date '2014-09-28' + integer '7' as datum; -- 2014-10-05
```

Enostavno lahko tudi ugotovimo število dni med dvema datumoma:

```
SELECT date '2014-10-07' - date '2014-03-27' as st_dni; -- 194
```

Na enostaven način lahko filtriramo vse zapise, ki niso starejši od enega leta in pol:

```
SELECT *
FROM nakupi
WHERE datum > (current_date - interval '1 year 6 month');
```

PostgreSQL je pri omejevanju podatkov uporaben tudi zaradi velikega števila razširitev. Ena bolj znanih prostorskih oz. geografskih razširitev je **PostGIS**. Omogoča hranjenje in poizvedovanje po prostorskih podatkih z uporabo lokacijskih poizvedb. Dodaja dodatne podatkovne tipe, funkcije, operatorje in vrste indeksov. Za uvažanje geografskih podatkov lahko uporabimo eno izmed orodij, ki podpirajo geografske podatke, npr. *GeoKettle*, *Safe FME*. PostGIS zna delati z vektorskimi in rastrskimi podatki.

PostGIS moramo namestiti posebej. Razširitev v podatkovno bazo naložimo z naslednjimi ukazi:

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION postgis_tiger_geocoder;
```

Za uporabo geografskih objektov moramo najprej kreirati tabelo, ki ima stolpec tipa "geometry"ali "geography". Primer:

```
CREATE TABLE gtest (
  gid serial primary key,
  name varchar(20),
  geom geometry(POINT)
);
```

Objekte (točke, linije, poligone, trikotnike ipd.) vstavljamo z SQL stavki:

```
INSERT INTO gtest (name, geom)  VALUES (
  'Točka',
  ST_GeomFromText('POINT(25 40)')
);
```

Geografske podatke pregledujemo kot nize ali jih filtriramo. Tako lahko poiščemo točke, ki so v radiju 100 enot od točke (10, 10):

```
SELECT gid, name, ST_AsText(geom) AS geom
FROM gtest
WHERE ST_DWithin(geom, 'POINT(10 10)', 100.0);
```

V praksi je še veliko možnosti uporabe, npr. kolikšna je dolžina vseh cest na nekem območju, koliko kvadratnih kilometrov je velika neka država, ali se dva področja prekrivata ...

Microsoft SQL Server

Microsoft SQL Server ne pozna aritmetičnih operatorjev za delo z datumi, kot jih pozna PostgreSQL. Tudi funkcij ima nekoliko manj. Vseeno pa lahko dokaj enostavno omejimo zapise glede na časovni atribut, npr. za izbiro zapisov, kjer je atribut datum mlajši kot leto in pol:

```
SELECT *
FROM nakupi
WHERE datum > DATEADD(MONTH, -18, GETDATE());
```

Microsoft SQL Server ima že vgrajeno podporo za prostorske podatke, imena funkcij so podobna kot pri PostGIS. Npr. namesto *ST_Intersect* je funkcija *STIntersection*. Ima pa PostGIS precej več funkcij. Če moramo podatke prenesti iz

druge podatkovne baze, lahko uporabimo že omenjeno funkcijo ogr2ogr, ki je del GDAL knjižnice.

5.2.3 Izdelava poročila zajemanja podatkov

Tudi izdelava poročila zajemanja podatkov spada v CRISP-DM fazo 2.1. Pri tem opravilu opišemo katere podatkovne vire smo uporabili v projektu in zakaj.

V poročilu navedemo tabele, poglede, datoteke in druge objekte znotraj virov, ki smo jih uporabili. Navedemo atribute, ki smo jih zajeli, in ali so kateri izmed atributov pomembnejši od drugih in zakaj. Nekateri atributi iz podatkovnih virov so morda za nas nepomembni in jih ne uvozimo ali pa jih odstranimo.

V poročilu navedemo tudi, katere zapise smo uporabili pri različnih virih (npr. časovno omejevanje podatkov). Zaradi nekonsistentnosti med viri (npr. različna natančnost pri časovnih podatkih) se lahko v združenih podatkih pojavijo novi problemi, kar moramo navesti tudi v poročilu. Preverimo tudi, ali smo zajeli vse potrebne podatke. Pridobivanje nekaterih podatkov lahko zahteva veliko človeških opravil.

V poročilu navedemo tudi kako smo zajeli podatke, kako smo beležili manjkajoče in ničelne vrednosti ter kako smo zajeli manjkajoče atribute.

5.3 CRISP-DM 2.2 (Opisovanje podatkov)

Nekaterih opravil v tem koraku ne moremo avtomatizirati, ker vključujejo človeške vire. Takšna opravila so: preučevanje zbranih rezultatov, odločanje, ali je nek podatek pomemben za izbran cilj podatkovnega rudarjenja, pogovori s strokovnjaki področja in pridobivanje njihovega mnenja o pomembnosti atributov, odločanje, ali je potrebno obtežiti podatke (odvisno od modelirne tehnike).

5.3.1 Število vrstic in število stolpcev

V tem koraku moramo pridobiti podatek o številu primerov in atributov v združenih podatkih.

Če imamo podatke shranjene v obliki CSV ali TAB, lahko število atributov preštejemo že v ukazni lupini, npr.:

```
head -n 1 fruitfly.csv | wc -w
```

Okvirno število primerov lahko pridobimo s pomočjo regularnega izraza, ki ne upošteva praznih vrstic ali vrstic s komentarji:

```
grep -v '^\s*$\|^\s*#' fruitfly.csv | wc -l
```

Pri relacijskih podatkovnih bazah je po uspešnem uvozu podatkov ugotavljanje števila primerov enostavno – uporabimo SQL agregacijsko funkcijo COUNT:

```
SELECT COUNT(*) FROM fruitfly;
```

PostgreSQL

Pri PostgreSQL lahko pridemo do hitrega a manj natančnega rezultata z uporabo stavka ANALYZE. Ta lahko naredi statistiko za vse tabele v podatkovni bazi, za posamezno tabelo ali le za en atribut. PostgreSQL sicer pri vklopljeni nastavitvi *autovacuum* to statistiko samodejno posodablja v ozadju. Če statistiko poženemo ročno v načinu VERBOSE, se izpiše okvirno število zapisov:

```
ANALYZE VERBOSE fruitfly;
```

Tako štetje je hitro, ker PostgreSQL ne prešteje vseh vrstic, ampak vzame nek vzorec[12] in izračuna okvirno število vrstic in druge statistike omenjene v nadaljevanju. Izračun je hiter[13] tudi pri večjih tabelah, so pa rezultati vsakič malo drugačni, ker pri vsakem zagonu izbere drugačno vzorčno množico. Statistike se shranijo v sistemski katalog *pg_statistic*. Javno dostopen pogled na ta katalog pa je *pg_stats*, ki prikazuje le informacije o tabelah, do katerih ima trenutni uporabnik bralne pravice. S temi statistikami si pomaga tudi načrtovalec poizvedb, ki jih uporablja za določanje najučinkovitejših planov poizvedb.

[12]Do največ 10000 naključno izbranih vrstic, privzeto pa le 100 vrstic. Število je odvisno od vrednosti konfiguracijske nastavitve *default_statistics_target*, lahko pa jo spremenimo le za en atribut z uporabo stavka "ALTER TABLE ... ALTER COLUMN ... SET STATISTICS".

[13]Razen če se mora statistika izvesti na velikem številu atributov in uporabljamo velik vzorec. Za čas izvajanja statistike na praktičnem primeru glej razpredelnico 8.1 na strani 103.

Pridobivanje števila atributov se med različnimi SUPB razlikuje. Pri PostgreSQL jih lahko pridobimo s poizvedbo po sistemski tabeli *information_schema.columns*:

```
SELECT COUNT(*)
FROM information_schema.columns
WHERE table_schema = 'public' AND table_name   = 'fruitfly';
```

Microsoft SQL Server

Pri Microsoft SQL Server je štetje števila primerov enako kot pri PostgreSQL – z uporabo agregacijske funkcije COUNT. Le ime tabele moramo navesti drugače - navedemo ime baze, ime sheme in ime tabele:

```
SELECT COUNT(*) FROM test.dbo.fruitfly;
```

Tudi pri Microsoft SQL je pri večjih podatkih možno hitro štetje števila vrstic z uporabo sistemske tabele sys.sysindexes, vendar je ta način precej težji kot pri PostgreSQL.

Atribute in njihove tipe lahko vidimo z uporabo orodja *Microsoft SQL Server Management Studio*. Število atributov pa lahko preštejemo skoraj enako kot pri PostgreSQL – s poizvedovanjem po sistemski tabeli:

```
SELECT COUNT(*)
FROM test.INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA='dbo' AND TABLE_NAME='fruitfly';
```

5.3.2 Računanje osnovne statistike

PostgreSQL

V relacijskih podatkovnih bazah kot je PostgreSQL lahko za preverjanje skrajnih vrednosti atributov uporabimo agregacijske funkcije. Za številčne vrednosti lahko pri PostgreSQL uporabimo funkcije *min, max, avg, stddev_samp, var_samp*, ki vrnejo minimalno in maksimalno vrednost, povprečje, standardno deviacijo in varianco. Omenjene vrednosti lahko za posamezen stolpec dobimo z enim samim SQL stavkom, npr.:

```
SELECT count(THORAX), min(THORAX), max(THORAX),
```

```
  avg(THORAX), stddev_pop(THORAX), var_pop(THORAX)
FROM fruitfly;
```

Pri analizi podatkov iščemo tudi najbolj pogoste vrednosti. Te lahko pridobimo s poizvedovanjem po tabeli *pg_stats*, ki smo jo opisali v prejšnjem podpoglavju in je povezana z operacijo ANALYZE. Za prikaz osnovne statistike naredimo takšno poizvedbo[14]:

```
SELECT *
FROM pg_stats
WHERE schemaname='public' AND tablename='fruitfly'
```

Izpisana statistika vključuje seznam najbolj pogostih vrednosti za vsak stolpec in histogram, ki prikazuje približno porazdelitev teh vrednosti. V spodnjem primeru izpisa so 1, 8 in 0 najbolj pogoste vrednosti. 40 % zapisov ima vrednost 1, 40 % vrednost 8, 20 % pa vrednost 0. Drugih vrednosti v tem primeru ni. Statistika prikaže tudi število različnih vrednosti (*n_distinct*[15]), kar je uporabno, če iščemo atribute, ki imajo same nedefinirane vrednosti, ali pa so vse vrednosti enake.

```
schemaname             | public
tablename              | fruitfly
attname                | PARTNERS
inherited              | f
null_frac              | 0
avg_width              | 4
n_distinct             | 3
most_common_vals       | {1,8,0}
most_common_freqs      | {0.4,0.4,0.2}
histogram_bounds       |
correlation            | 0.423963
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
```

[14]Za lepši prikaz v orodju *psql* pred tem uporabimo ukaz za razširjen izpis "\x on".

[15]Ta vrednost je lahko tudi negativna, če PostgreSQL predvideva, da se bo število različnih vrednosti povečevalo. V tem primeru deli število različnih vrednosti s številom vrstic.

Malo manj uporabna za naš namen je prikazana korelacija, ki ne pomeni korelacije med stolpci ampak govori o koreleaciji med logično in fizično urejenostjo podatkov na disku[16]. Če pa želimo izračunati korelacijo dveh atributov pa uporabimo funkcijo *corr*, npr.:

```
SELECT corr("PARTNERS", "class") FROM fruitfly;
```

V prilogi na strani 117 je funkcija *maxcorr*, ki je bila izdelana za namen te naloge in omogoča prikaz vseh ali le najvišjih korelacij med vsemi pari atributov. Funkcije za izračun mediane PostgreSQL nima, je pa takšna funkcija objavljena na PostgreSQL Wiki[17].

Microsoft SQL Server

Pri Microsoft SQL Server lahko za izračun osnovne statistike nad stolpci uporabimo enake funkcije, le imena so nekoliko drugačna. Primer poizvedbe:

```
SELECT count(THORAX), min(THORAX), max(THORAX),
  avg(THORAX), stddevp(THORAX), varp(THORAX)
FROM test.dbo.fruitfly;
```

Tudi Microsoft SQL Server vodi statistike tabel in na njihovi podlagi izdeluje plane poizvedb. Statistika se posodablja avtomatsko (npr. po spremembi zapisa), čeprav včasih to ne zadošča. Za kreiranje statistike lahko uporabimo stavek CREATE STATISTICS ali pa jo kreiramo prek grafičnega vmesnika. Pri Microsoft SQL Server lahko v statistiko vključimo celotno množico podatkov in ne le vzorca (PostgreSQL je pri statistiki omejen na 10000 zapisov, razen če statistiko izvajamo s pomočjo funkcij nad vsemi zapisi). Tudi pri Microsoft SQL Server se statistika hrani v sistemskih tabelah. Vsebuje histogram porazdelitev podatkov v atributu.

Za prikaz statistike, ki smo jo predhodno kreirali, lahko uporabimo takšen stavek:

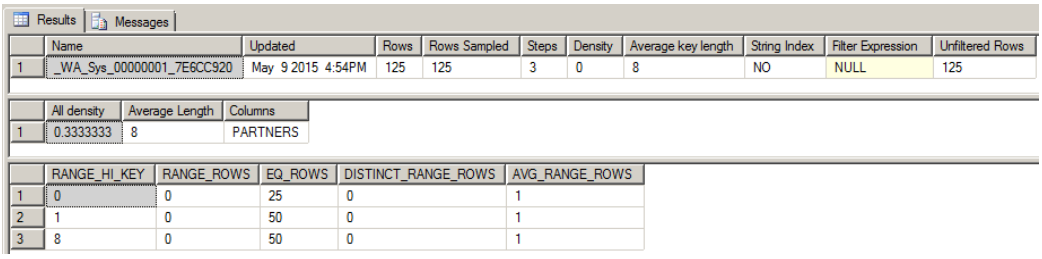
```
USE test;
GO;
```

[16] Najpočasnejše skeniranje indeksa je pri vrednosti 0, najhitrejše pa pri skrajnima vrednostma -1 in 1.

[17] https://wiki.postgresql.org/wiki/Aggregate_Median

```
DBCC SHOW_STATISTICS (fruitfly, THORAX) ;
```

Tudi Microsoft SQL Server prikaže porazdelitev vrednosti atributa, kar je prikazano na sliki 5.1.



Slika 5.1: Prikaz statistike porazdelitve vrednosti za atribut THORAX pri Microsoft SQL Server Express Edition.

Funkcije za izračun korelacije med dvema atributoma pri Microsoft SQL Server nisem našel. Je pa možen izračun odvisnosti med atributi znotraj Microsoft SQL Server Integration Services (SSIS), ki je na voljo le pri plačljivih verzijah tega SUPB.

Tudi izračuna mediane ne podpira, je pa možno napisati poizvedbo za izračun mediane[18].

5.3.3 Podvojevanje podatkovne baze ali posamezne tabele

Ta korak sicer ni del CRISP-DM, vendar – ker bomo v nadaljevanju podatke spreminjali – lahko naredimo kopijo podatkovne baze ali kopijo tabele s podatki.

PostgreSQL

V PostgreSQL napravimo kopijo celotne baze z ukazom:

```
CREATE DATABASE testdb_copy WITH TEMPLATE testdb;
```

Lahko pa naredimo kopijo le posamezne tabele, npr.:

[18] http://sqlblog.com/blogs/adam_machanic/archive/2006/12/18/medians-row-numbers-and-performance.aspx

```
CREATE TABLE fruitfly_copy AS TABLE fruitfly;
```

Microsoft SQL Server

Pri Microsoft SQL Server lahko v brezplačni verziji naredimo kopijo celotne podatkovne baze z uporabo že omenjenega čarovnika za uvoz podatkov. Najprej kreiramo novo podatkovno bazo in vanjo prek čarovnika uvozimo podatke – podobno kot pri uvozu CSV datotek, le da v tem primeru za vir izberemo lokalni Microsoft SQL Server. Uvozimo lahko podatkovne tabele in poglede[19]. Na enak način lahko uvozimo le izbrane podatkovne tabele. Plačljive verzije imajo možnost prenesti celotno podatkovno bazo[20], vključno z uporabniško definiranimi procedurami, omejitvami idr. Kopijo podatkov lahko naredimo tudi v binarno datoteko in jo kasneje uvozimo[21].

5.3.4 Odstranjevanje podvojenih zapisov

Podvojeni zapisi se ponavadi pojavljajo, ko tabela nima definiranega primarnega ali unikatnega ključa. Podvojeni so lahko celotni zapisi ali le določeni atributi.

PostgreSQL

PostgreSQL za vsaka tabelo hrani sistemske atribute, ki jih ne vidimo, razen če jih eksplicitno zahtevamo. Ti atributi so: *oid, tableoid, xmin, cmin, xmax, cmax in ctid*[22]. Za odstranjevanje podvojenih zapisov je uporaben atribut **ctid**, ki za vsak zapis v tabeli hrani unikatno vrednost. Predstavlja fizično lokacijo zapisa znotraj tabele. Če želimo odstraniti zapise, kjer so podvojeni vsi (nesistemski) atributi, lahko to naredimo z odstranitvijo vseh zapisov, ki niso prvi med enakimi, npr. (pozor na zadnji znak *):

```
DELETE FROM fruitfly WHERE ctid NOT IN (
```

[19]Pri testnem uvozu na *MS SQL Server 2008 R2 Express Edition* so se uvozile samo podatkovne tabele, pogledi pa ne.

[20]Prek "Tasks" in "Copy database... ".

[21]Preko "Tasks" in "Backup" oz. "Tasks" in "Restore Database... "

[22]Teh imen tudi ne moremo uporabiti pri kreiranju atributov.

```
    SELECT min(ctid)
    FROM fruitfly
    GROUP BY fruitfly.*
);
```

Če pa želimo odstraniti zapise, kjer se podvajajo izbrani atributi, naredimo enako, le da pri GROUP BY eksplicitno navedemo te atribute in uporabimo še del stavka ORDER BY, s čimer kontroliramo, kateri atributi bodo določali prvi zapis in bodo zato ostali v tabeli, npr.:

```
DELETE FROM fruitfly WHERE ctid NOT IN (
    SELECT min(ctid)
    FROM fruitfly
    GROUP BY THORAX, SLEEP, class
    ORDER BY PARTNERS DESC
);
```

V gnezdenem SELECT stavku bi lahko namesto agregacijske funkcije uporabili tudi stavek SELECT DISTINCT ON. PostgreSQL poleg stavka SELECT DISTINCT, ki prikaže le naštete unikatne atribute, pozna tudi stavek SELECT DISTINCT ON. Pri slednjem v oklepaju navedemo kateri atributi morajo biti unikatni, na koncu pa naštejemo vse atribute, ki jih želimo prikazati. Primer:

```
DELETE FROM fruitfly WHERE ctid NOT IN (
    SELECT DISTINCT ON (THORAX, SLEEP, class) ctid
    FROM fruitfly
    ORDER BY partners DESC
);
```

To pomeni, da med vsemi zapisi, kjer se pojavlja enaka kombinacija atributov THORAX, SLEEP in class, izberemo le identifikator vrstice *ctid* in na podlagi tega odstranimo vse preostale vrstice. Namesto brisanja odvečnih zapisov bi lahko izbrane vrstice enostavno prekopirali v novo tabelo[23].

Microsoft SQL Server

Namesto sistemskega atributa *ctid* lahko pri Microsoft SQL Server uporabimo sistemski atribut *physloc*[24], ki naj bi predstavljal fizično lokacijo vrstice. Za brisanje v celoti podvojenih zapisov in zapisov, ki imajo podvojene le izbrane atribute, bi tako lahko uporabili stavek[25]:

```
DELETE FROM test.dbo.fruitfly WHERE %%physloc%% NOT IN (
    SELECT MIN(%%physloc%%)
    FROM test.dbo.fruitfly
    GROUP BY THORAX, SLEEP, class
    ORDER BY partners
)
```

Microsoft SQL Server ne pozna konstrukta DISTINCT ON, kot ga pozna PostgreSQL.

5.4 CRISP-DM 2.3 (Raziskovanje podatkov)

V tem koraku se osredotočamo na poizvedovanje, vizualizacijo in poročanje. To nam olajša izbiro transformacij, ki sledijo v nadaljnjih korakih.

5.4.1 Vizualizacija podatkov

Z vizualizacijo podatkov se ukvarja področje raziskovalne podatkovne analize (an. **EDA** - Exploratory Data Analysis). Pri EDA raziskovalec raziskuje podatke brez kakršnihkoli vnaprejšnjih idej, z namenom, da bi odkril, kaj mu lahko podatki povedo o fenomenu, ki ga preučuje[25]. EDA se uporablja skupaj s potrditveno podatkovno analitiko (an. CDA - Confirmatory Data Analysis), ki se ukvarja s testiranjem statističnih hipotez in intervali zaupanja. Analitik raziskuje podatke z opazovanjem vzorcev in strukture, kar vodi v hipoteze. Osnova EDA je uporaba znanstvenih in statističnih vizualizacij, kar je včasih edini način odkrivanja vzorcev. Z EDA iščemo tudi anomalije v podatkih.

[23] Z uporabo stavka CREATE TABLE ... AS SELECT ...

[24] Ta sicer ni uradno dokumentiran.

[25] Navesti moramo vse atribute. V GROUP BY ne moremo uporabiti znaka * kot pri Postgresql.

Za vizualizacijo podatkov se uporabljajo različne tehnike. EDA se ukvarja tudi z vizualnimi transformacijami podatkov – podatkom skuša zmanjšati dimenzionalnost (npr. z uporabo PCA), ker strukturo lažje odkrijemo na podlagi manjšega števila dimenzij značilk. V uporabi so različni vizualizacijski gradniki. Za ugotavljanje oblike distribucije se uporabljajo histogrami, q–q grafi, škatle z brki (an. boxplot) idr. Za vizualizacijo se uporabljajo raztreseni grafikoni (an. scatterplot), grafikoni s pikami (an. dot plots), dendrogrami idr.

Nekatera EDA orodja lahko delujejo le nad tekstovnimi podatki, druga omogočajo tudi raziskovanje podatkov shranjenih v podatkovni bazi, npr. *DbVisualizer, SQL Power Architect, JFreeChart, Tableau Public*. Mnoga orodja so na voljo kot storitve, dostopne prek spleta, npr. *Datawrapper*[26], *RAW*[27] – druga pa so uporabna kot knjižnice za razvijalce spletnih aplikacij, npr. *D3.js, FusionCharts, dygraphs*[28]. Zelo močni orodji za EDA sta tudi *Matlab* in *Octave*. Primer vizualizacije v aplikaciji RAW je prikazan na sliki 5.2:

5.4.2 Izdelava poročil

Za izdelavo poročil lahko uporabimo orodje za avtomatsko izdelavo poročil. Eno takšnih je npr. *Jasper Reports*, ki lahko izdela HTML, PDF, Excel, LibreOffice ali Word poročilo.

Uporabimo lahko tudi odprtokodno razširitev za *Eclipse* – *BIRT*, ki omogoča vizualizacijo in izdelavo poročil. Vsebuje del za oblikovanje poročil in del za zaganjanje.

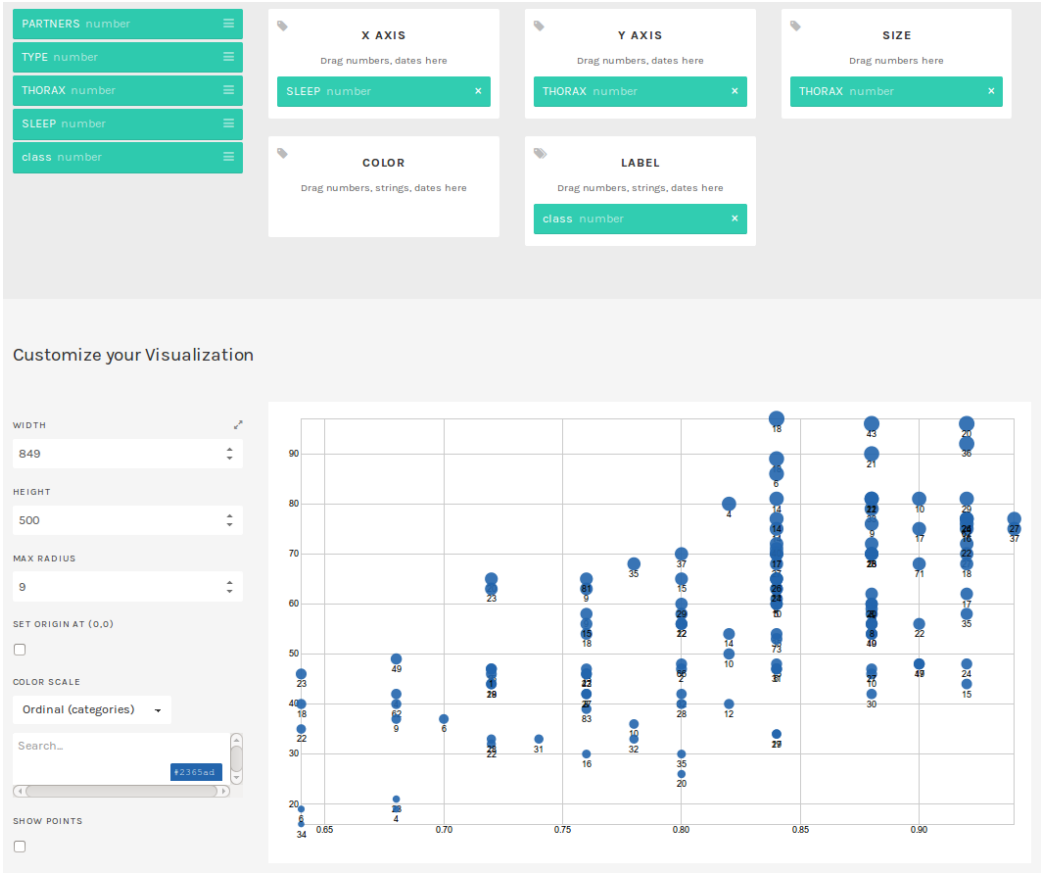
Tudi Pentaho ima orodje *Pentaho Report Designer*, ki omogoča izdelavo poročil v PDF, Excel, HTML, tekstovni datoteki, RTF, XML ali CSV.

Še eno orodje za izdelavo poročil je *DBxtra Report Designer*. To komercialno orodje deluje le v Windows okolju, a ima dobro podporo za različne SUPB. Za izdelavo poročil ne zahteva poznavanja SQL. Omogoča pregledovanje podatkov, poročila pa lahko izvozi v Excel, Word, PDF, HTML, CSV, slikovne datoteke ali

[26]Datawrapper se je pri testiranju z uporabo 1 MB velike CSV datoteke v brskalniku Firefox izkazal za zelo nestabilnega.

[27]Tudi RAW je bil pri vizualizaciji 1 MB velike datoteke precej neodziven. Dostopen je na naslovu: http://raw.densitydesign.org/

[28]Dygraphs je primeren tudi za vizualizacijo večjih količin podatkov



Slika 5.2: Primer vizualizacije z raztresenim grafikonom v spletni aplikaciji RAW.

XML. Poročila lahko shrani tudi v obliki, ki jo drugi uporabniki lahko pregledujejo z zastonj orodjem *Desktop Report Viewer*.

5.5 CRISP-DM 2.4 (Preverjanje kvalitete podatkov)

V tem koraku preverimo kvaliteto podatkov. Zanima nas ali so podatki popolni. Ali pokrivajo vse zahtevane primere? Ali so točni? Ali vsebujejo napake in koliko je teh napak? Ali v podatkih obstajajo manjkajoče vrednosti? Kako so predstavljene manjkajoče vrednosti, kje se pojavljajo in kako pogoste so? Izhod tega koraka je *poročilo kvalitete podatkov*.

5.5.1 Preverjanje formata

Če lahko na rezultat podatkovnega rudarjenja vpliva napačna velikost črk, potem lahko vse vrednosti atributov poenotimo tako, da imajo vsi nizi samo velike ali samo male črke. Odstranimo tudi morebitne začetne in končne presledke ali druge znake.

PostgreSQL

V PostgreSQL lahko za pretvorbo v velike ali male črke uporabimo funkcijo *upper* oziroma *lower*. Če želimo, da je velika samo prva črka, uporabimo funkcijo *initcap*. Za odstranjevanje začetnih in končnih presledkov lahko uporabimo funkcije *trim*, *btrim*, *replace* ali *regexp_replace*. Funkciji *regexp_matches* in *split_part* sta uporabni, če želimo iz nizov v atributih izluščiti podnize.

Še ena funkcionalnost PostgreSQL, ki je pri tem koraku lahko uporabna, je *unaccent*. To je slovar, ki iz besed odstrani vse diakritične znake. Tako lahko iščemo besede, ne glede na to, ali vsebujejo znake za naglas. Podamo mu ime datoteke s pravili ali pa uporabimo privzeto datoteko *unaccent.rules*. Primer pretvorbe s privzetimi pravili:

```
book=# CREATE EXTENSION unaccent;
CREATE EXTENSION
book=# SELECT unaccent('čžšČŽŠ');
  unaccent
------------
 czsCZS
(1 row)
```

Microsoft SQL Server

Microsoft SQL Server ima za pretvorbo v velike ali male črke istoimenske funkcije kot PostgreSQL (*upper* oz. *lower*). Funkcije podobne *initcap* ne pozna, lahko pa kreiramo podobno uporabniško funkcijo in v njej uporabljamo razdruževanje in združevanje nizov ter funkciji *upper* in *lower*. Za odstranjevanje začetnih in končnih presledkov lahko uporabimo funkcije *ltrim*, *rtrim* in **replace**. Zamenjav z uporabo regularnih izrazov ne pozna. Prav tako ne pozna funkcije za razbijanje

nizov na podnize. Tudi funkcije za odstranjevanje diakritičnih znakov ne pozna, lahko pa podobno funkcionalnost dosežemo s stavkom kot je:

```
SELECT CAST(N'čžšČŽŠ' AS VARCHAR(MAX))
COLLATE SQL_Latin1_General_Cp1251_CS_AS
```

5.5.2 Preverjanje črkovanja

Tekstovni podatki lahko vsebujejo napake, ki bi jih radi čim bolj avtomatično odstranili.

PostgreSQL

Funkcij, ki bi odpravljale napake črkovanja v PostgreSQL nisem našel. Nekatere funkcije pa lahko pri iskanju napak pomagajo. Za to lahko uporabimo **Trigram** (del razširitve *pg_trgm*). To je skupina treh zaporednih znakov vzetih iz niza. Funkcija *pg_trgm* razbije niz na največje možno število trigramov. Z uporabo trigramov lahko primerjamo podobnosti med nizi. Uporabimo jih lahko le na stolpcih, kjer smo definirali GIST ali GIN indeks. Trigrami so uporabni za popravljanje uporabnikovih vnosov, ko se uporabnik zatipka. Tako lahko tudi poiščemo nize, ki se ne ujemajo, vendar so podobni. Za to uporabimo operator "%". Npr.:

```
CREATE INDEX actors_name_trigram
  ON actors USING gist (name gist_trgm_ops);
SELECT * FROM actors WHERE name % 'Conneri';

     actor_id |     name
    ----------+--------------
            5 | Sean Connery
```

V nadaljevanju podajam primer, kako lahko poiščemo tipkarske napake v atributih, ki vsebujejo nize. Denimo, da imamo v nekem stolpcu shranjena imena mest. Če je podatkovna tabela dovolj velika, se bo vsako veljavno mesto pojavilo več kot enkrat. Če smo se pri vnosu podatkov pri istem mestu zatipkali večkrat, je malo verjetno, da smo se zatipkali na točno istem mestu. Primer takšnih podatkov bi bil:

```
CREATE TABLE mesta (ime text, predlagano_ime text);
CREATE INDEX mesta_ime_idx
  ON mesta USING gist (ime gist_trgm_ops);
INSERT INTO mesta (ime) VALUES ('celje'), ('Koper'),
  ('CECJE'), ('Celje'), ('KoPER'), ('Kranj'),
  ('Kranj'), ('Kramj'), ('Celje'), ('Kranj');
SELECT * FROM mesta;

  ime   | predlagano_ime
-------+-------------
 celje |
 Koper |
 CECJE |
 Celje |
 KoPER |
 Kranj |
 Kranj |
 Kramj |
 Celje |
 Kranj |
(10 rows)
```

Ker je uporaba velikih in malih črk v tem primeru nekonsistentna, lahko najprej popravimo imena tako, da so imena zapisana z veliko začetnico, ostale črke pa so male. To v PostgreSQL naredimo s funkcijo *initcap*.

```
UPDATE mesta SET ime=initcap(ime);
```

Če še enkrat prikažemo seznam mest, vidimo, da smo se zatipkali pri besedah "Cecje" in "Kramj". Ker predpostavljamo, da enake napake ponovimo le enkrat, je najenostavneje, da najprej izdelamo začasno tabelo, ki hrani različna imena mest ter kolikokrat se ponavljajo.

```
CREATE TEMP TABLE mesta_cnt AS
SELECT ime, COUNT(ime)
FROM mesta
GROUP BY ime
ORDER BY count DESC;
```

Sedaj lahko z uporabo trigrama (funkcija *similarity*) poiščemo mesta z najbolj podobnim imenom in nastavimo vrednost atributa "predlagano_ime".

```
UPDATE mesta AS M SET predlagano_ime=(
  SELECT COALESCE(
    (
      SELECT ime
      FROM mesta_cnt
      WHERE ime!=M.ime AND similarity(ime, M.ime) > 0
        AND count >=2
      ORDER BY similarity(ime, M.ime) DESC
      LIMIT 1
    ), M.ime
  )
)
```

V notranji SQL zanki smo za vsako ime v naši prvotni tabeli poiskali najbolj podobno ime v začasni tabeli (vrednost similarity večja od 0), ki se pojavlja vsaj dvakrat. Če takšnega imena nismo našli, smo predlaganemu imenu priredili kar prvotno ime (zato funkcija COALESCE, ki vrne prvo neprazno vrednost). Dobimo naslednji rezultat:

```
SELECT * FROM mesta;
  ime   | podobno_ime
--------+-------------
 Celje  | Celje
 Koper  | Koper
 Cecje  | Celje
 Celje  | Celje
 Koper  | Koper
 Kranj  | Kranj
 Kranj  | Kranj
 Kramj  | Kranj
 Celje  | Celje
 Kranj  | Kranj
```

Vidimo, da poizvedba pri zatipkanih imenih "Kramj" in "Cecje" predlaga pravilno ime. Vidimo, da popravljanje tiskarskih napak ni trivialna naloga in tudi rešitev ni popolnoma zanesljiva (npr. da bi se na istem mestu zatipkali več kot enkrat),

vendar je vsekakor lahko uporabna.

Še ena funkcija, ki bi jo lahko uporabili pri iskanju podobnih nizov je **Levenshteinova razdalja**, ki je iz paketa *fuzzystrmatch*. Uporabimo jo lahko podobno kot trigrame:

```
test=# CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
test=# SELECT levenshtein('Celje', 'Celje');
 levenshtein
-------------
           0
(1 row)

test=# SELECT levenshtein('Celje', 'Cecje');
 levenshtein
-------------
           1
(1 row)

test=# SELECT levenshtein('Maribor', 'Kranj');
 levenshtein
-------------
           6
(1 row)
```

Levenshteinova razdalja vrne pri enakih nizih vrednost 0. Bolj kot so si imena med seboj različna, večja je ta razdalja. Pri trigramih je ta vrednost med -1 (za najbolj tuji imeni) in 1 (za enaki imeni) in je realno število. Levenshteinova razdalja je vedno celo število, saj v resnici pove, koliko spreminjanj posameznih znakov niza je potrebnih, da ga spremenimo v drug niz. Na tem mestu lahko omenimo še **soundex**, ki omogoča iskanje podobno zvenečih imen, a le za angleška imena.

Microsoft SQL Server

Za Microsoft SQL Server nisem našel funkcije, ki bi znala popraviti zatipkane besede. Tudi funkcije za levensheinovo razdaljo ali za trigrame nima. Našel pa sem način, kako v Microsoft SQL Server vključiti v Javi napisano knjižnico *SimMetrics*,

ki poleg Levenshteinove, ponuja še druge razdalje[26] in naj bi imela tudi dobre performanse.

5.6 CRISP-DM 3.1 (Izbira podatkov)

V tem koraku se moramo odločiti, katere zapise in atribute vključiti v analizo. Kriteriji za izbiro so odvisni od ciljev podatkovnega rudarjenja, kvalitete podatkov in tehničnih omejitev (omejitev podatkovnih tipov, omejitev količine podatkov, ipd.).

5.6.1 Izbira podatkovnih podmnožic

Atribute, ki jih potrebujemo za podatkovno rudarjenje, izberemo v SELECT delu SQL stavka, izbiro primerov pa v WHERE delu stavka. Izbrati moramo primere, ki bodo predstavljali našo učno množico. Za kriterij lahko npr. izberemo, da podatki niso starejši od dveh let.

PostgreSQL

V PostgreSQL lahko dokaj enostavno razdelimo primere na naključno *testno in učno množico*. Če želimo npr. 10 % podatkov v testni množici in ostale v učni, lahko uporabimo funkcijo *random* in že omenjen sistemski atribut *ctid* ter tako ustvarimo dve novi podatkovni tabeli za testno in učno množico, npr.:

```
CREATE TABLE fruitfly_testna
AS SELECT ctid AS ctid2, * FROM fruitfly WHERE random() < 0.1;

CREATE TABLE fruitfly_ucna
AS SELECT ctid AS ctid2, * FROM fruitfly WHERE ctid NOT IN
  (SELECT ctid2 FROM fruitfly_testna)
```

Microsoft SQL Server

Kreiranja testne in učne množice na podlagi naključnih primerov pri Microsoft SQL Server ni bilo mogoče narediti na tak način kot pri PostgreSQL, ker deluje

primerjanje pogoja v SQL drugače kot pri PostgreSQL in zato ni možno neposredno uporabiti funkcije za kreiranje naključnih števil. Če v Microsoft SQL Sever vnesemo stavek

```
SELECT * FROM test.dbo.fruitfly WHERE RAND() < 0.1;
```

potem kot rezultat ne dobimo približno desetine vseh zapisov kot bi pričakovali, ampak vse zapise (v povprečju pri vsakem desetem poskusu) ali pa nobenega. Če v WHERE delu stavka ni navedenega nobenega atributa ali znaka *, potem se najprej izračuna pogoj "$rand() < 0.1$" in nato rezultat tega pogoja uporabi na vseh zapisih. Če je z vidika optimizacije poizvedbe to zaželjena lastnost, pa to za naš namen nikakor ni zaželjeno. PostgreSQL izračuna pogoj pri vsakem zapisu.

Predlog na Microsoftovem spletnem mestu[29] je, da namesto preproste RAND funkcije v WHERE delu stavka uporabimo kompleksnejši izraz, ki vsebuje znak *:

```
... WHERE
(ABS(CAST((BINARY_CHECKSUM(*) *RAND()) as int)) % 100) < 10
```

Če uporabimo še omenjeni sistemski atribut %%physloc%%, potem bi izdelava naključne testne in učne množice v Microsoft SQL Server zgledala takole:

```
SELECT %%physloc%% AS physloc2, *
INTO dbo.fruitfly_testna
FROM dbo.fruitfly
WHERE
  (ABS(CAST((BINARY_CHECKSUM(*) * RAND()) as int)) % 100) < 10

SELECT %%physloc%% AS physloc2, *
INTO dbo.fruitfly_ucna
FROM dbo.fruitfly
WHERE %%physloc%% NOT IN
  (SELECT physloc2  FROM dbo.fruitfly_testna)
```

Pri Microsoft SQL Server torej zahteva izdelava testne in učne množice na podlagi naključnih primerov nekaj več pisanja.

Vse zapise bi lahko na testno in učno množico razdelili tudi na podlagi drugih pogojev, npr. na podlagi zaporedne številke zapisa. V našem primeru smo

[29] https://msdn.microsoft.com/en-us/library/cc441928.aspx

pri ustvarjanju testne in učne množice ustvarili tudi dodaten atribut *ctid2* oz. *physloc2*, ki bi ga lahko po koncu kreiranja množic odstranili.

5.6.2 Testi pomembnosti in korelacijski testi

Te teste izvedemo za izbiro uporabnih in iskanje neuporabnih atributov. Atributi, ki imajo pri vseh primerih enako vrednost so neuporabni. Odvečni so lahko tudi tisti atributi, ki jih pridobimo iz drugih atributov (npr. razdalja v kilometrih in razdalja v miljah). Pri klasifikaciji so uporabni tisti atributi, ki imajo visoko korelacijo s ciljnim razredom. Korelacijski testi so že opisani pri CRISP-DM koraku 2.2 v poglavju 5.3.2 na strani 59. Pri dokončni izbiri atributov, ki jih bomo vključili v podatkovno rudarjenje lahko ponovno izvedemo korelacijske teste. V prilogi te naloge na strani 117 je funkcija, ki pokaže pare atributov z najvišjimi korelacijskimi koeficienti.

Korelacijske teste lahko pri manjših podatkih izvedemo s programi kot sta Microsoft Excel ali LibreOffice Calc, z uporabo funkcije *CORREL*.

5.7 CRISP-DM 3.2 (Čiščenje podatkov)

V tem koraku odpravljamo šum v podatkih in upravljamo s posebnimi vrednostmi, ki bi lahko vplivale na rezultat podatkovnega rudarjenja.

5.7.1 Odpravljanje šuma v podatkih

Šum v podatkih je pogost pojav. Lahko je posledica fizikalnih omejitev pri meritvah. Pri zbiranju podatkov pri ljudeh pa je lahko posledica slabe komunikacije, navajanja neresnic ipd. Šum želimo, če je le mogoče, pred samim podatkovnim rudarjenjem odstraniti. Že ena zelo izstopajoča vrednost (an. outlier) lahko precej vpliva na povprečje.

Izstopajoče vrednosti lahko obravnavamo kot manjkajoče vrednosti, jim nastavimo skrajne vrednosti ali drugo. Pri njihovem odkrivanju si lahko pomagamo z raziskovalno podatkovno analizo (glej poglavje 5.4.1).

Ena izmed metod odstranjevanja izstopajočih vrednosti je **interkvartilni razmik**, ki je mera statistične razpršitve. Metoda poteka tako, da podatke najprej

razvrstimo po naraščajoči vrednosti, nato jih razdelimo v 4 enake dele (najprej na 2, nato še na 2). Kvartili so zgornje meje teh delov (Q_1, Q_2, Q_3 in Q_4). Imajo vrednost elementa ali povprečje sosednjih elementov. Interkvartilni razmik je razlika med Q_1 in Q_3, kot je prikazano v enačbi 5.1.

$$IQR = Q_3 - Q_1 \tag{5.1}$$

Preprost način postavitve spodnje in zgornje meje med veljavnimi in izstopajočimi vrednostmi je z uporabo formule 5.2 oz. 5.3.

$$L = Q_1 - 1.5 \cdot IQR \tag{5.2}$$

$$H = Q_3 + 1.5 \cdot IQR \tag{5.3}$$

Niti PostgreSQL niti Microsoft SQL Server nimata funkcij za odstranjevanje izstopajočih vrednosti. V prilogi na strani 119 je PL/PgSQL agregacijska funkcija (napisana za namen te naloge), ki na podlagi interkvartilnega razmika izračuna spodnjo in zgornjo mejo veljavnih vrednosti za izbran atribut. Atribut je lahko celoštevilski ali realen.

Šum lahko odstranjujemo tudi v drugih vrstah podatkov, ne samo pri numeričnih. Odpravljanje šuma v tekstovnih podatkih je opisan v poglavju 5.5.2. V slikovnih podatkih ga odstranjujemo z uporabo ustreznih filtrov, npr. medianin filter za šum tipa sol in poper, Gaussov filter za odstranjevanje Gaussovega šuma. Pri zvokovnih podatkih lahko uporabljamo filtre za prepuščanje nizkih oz. visokih filtrov (an. high pass filter) ipd. Za odpravo šuma v slikovnih podatkih sta zelo uporabni orodji Matlab (Image Processing Toolbox) in Octave (paket image[30]), ki že vsebujeta precej filtrov (funkcija *fspecial*). Matlab lahko uporabimo tudi za procesiranje zvoka[31].

5.7.2 Upravljanje posebnih vrednosti

Primer posebnih vrednosti je, če imamo v podatkih ankete za neodgovorjeno vprašanje vrednost 99. Pojavljajo se tudi, kjer so podatki odrezani (npr. 00 za 100 let starega človeka). V prvem primeru bi lahko npr. vrednosti 99 spremenili

[30] http://octave.sourceforge.net/image/

[31] http://www.mathworks.com/discovery/audio-signal-processing.html

v vrednost NULL, v drugem pa bi lahko vrednosti 00 popravili s SQL stavkom UPDATE v vrednost 100.

5.8 CRISP-DM 3.3 (Ustvarjanje podatkov)

V koraku ustvarjanja podatkov se iz obstoječih atributov izdelajo novi izvedeni atributi. Kreirajo se lahko tudi čisto novi zapisi. Izvedeni atributi so sestavljeni iz enega ali več obstoječih atributov v istem zapisu. Npr. za potrebe modeliranja dodamo nov atribut *površina*, ki je pridobljen z množenjem *dolžine* in *širine*[32].

Včasih ustvarimo izvedene atribute zato, ker nekateri modelirni algoritmi ne znajo delati z določenimi tipi podatkov. Pogosto moramo atribute normalizirati tako, da so vse vrednosti med 0 in 1, ali diskretizirati, da ima atribut le nekaj možnih vrednosti[33]. Pri uteženi normalizaciji kreiramo nove atribute zato, da damo prvotnim atributom uteži. Izvedenih atributov ne smemo dodajati le zato, da bi zmanjšali število vhodnih atributov, ampak morajo prispevati k modelirnemu algoritmu. Tako je npr. prihodek na osebo bolj uporaben atribut kot prihodek na gospodinjstvo.

Drug tip izvedenih atributov je agregacija enega samega atributa, ki je ponavadi izvedena zato, da ustreza potrebam modelirnega orodja. Npr. pri podatkih o nakupih stranke lahko pri vsakem nakupu dodamo povprečno vrednost nakupa te stranke.

Včasih se v tem koraku izdelajo tudi popolnoma novi zapisi, npr. pri segmentaciji podatkov je uporabno, da imamo za vsak segment zapis, ki predstavlja prototipičen primer.

5.8.1 Transformacija obstoječega atributa

Obstoječ atribut ponavadi transformiramo zaradi potreb modelirnega orodja. Pogost primer uporabe je **diskretizacija** atributa. Veliko modelirnih orodij ne zna delati z numeričnimi atributi ali pa jih njihova uporaba precej upočasni.

[32]Po izračunu površine, bi bilo v tem primeru za namen priprave podatkov atributa dolžine in širine smiselno odstraniti.

[33]Npr. intervalni tip o starosti lahko spremenimo v ordinalni tip [mlad, sredje, star]

PostgreSQL

Pri PostgreSQL lahko za spreminjanje obstoječega atributa uporabimo stavek UPDATE, vendar pa se mora tip vrednosti, ki jo nastavljamo, ujemati s tipom atributa. Primer:

```
UPDATE fruitfly SET "SLEEP"="SLEEP"*100;
```

Pri spreminjanju tipov atributov ali pri zahtevnejših transformacijah lahko napišemo PL/PgSQL funkcijo in jo kličemo s konstruktom USING, npr.

```
ALTER TABLE tbl ALTER COLUMN col TYPE float
USING custom_function(col);
```

Funkcije za diskretizacijo atributov PostgreSQL na žalost nima. Ob izračunu mej bi lahko diskretizacijo izvedli z CASE stavkom, vendar je to pri velikem številu atributov zamudno opravilo. Poleg tega bi morali predhodno izračunati mejne vrednosti posameznih razredov glede na izbran način diskretizacije (nadzorovan ali nenadzorovan).

Microsoft SQL Server

Tudi pri Microsoft SQL Server za spreminjanje obstoječega stolpca uporabljamo stavek UPDATE:

```
UPDATE test.dbo.fruitfly SET SLEEP=SLEEP*100;
```

Diskretizacija atributov je podprta le v Analisis Services.

5.8.2 Transformacija enega ali več atributov v nov atribut

Transformacijo v nov atribut naredimo tako, da najprej kreiramo nov atribut, nato pa nastavimo njegovo vrednosti na podlagi vrednosti drugih atributov, npr.:

```
ALTER TABLE oseba ADD polnoleten BOOLEAN;
UPDATE oseba SET polnoleten =
    CASE
        WHEN starost < 18 THEN false
        ELSE true
```

```
    END;
```

Namesto kreiranja novega atributa, lahko tudi izdelamo pogled, kjer je nov atribut transformacija več drugih atributov.

PostgreSQL

V PostgreSQL, od verzije 9.3 naprej, lahko izdelamo *materializirane poglede* z uporabo stavka CREATE MATERIALIZED VIEW. To je podobno kot bi kreirali tabelo, le da se podatki posodobijo prek stavka REFRESH MATERIALIZED VIEW, direktno pa se njihovih vrednosti ne da spreminjati. Navaden pogled nas le reši zapletenih SQL stavkov, materializiran pogled pa ima rezultate poizvedbe fizično shranjene in zato pri ponovnem poizvedovanju istega pogleda lahko prihrani precej časa (še posebej če so poizvedbe za pogled časovno zahtevne operacije).

Microsoft SQL Server

Materializiran pogled omogoča tudi Microsoft SQL Server, le da je pri njem poimenovan indeksirani pogled. Tudi konstrukt CASE je enak kot pri PostgreSQL.

5.9 CRISP-DM 3.4 (Integracija podatkov)

V tem koraku združujemo dve ali več tabel z različnimi informacijami o istih objektih (podatke povezujemo), poleg tega pa tudi **zbiramo zapise** iz več tabel v eno tabelo.

Podatki, pridobljeni iz različnih virov, so lahko med sabo nekonsistentni, zato jih moramo pred združevanjem poenotiti v **obliki** (npr. koti v stopinjah ali radianih), **kvaliteti** (npr. različno število decimalnih mest) in **agregaciji** (npr. istovrstni podatki za 10-minutni in 15-minutni interval). Nekateri podatki so lahko tudi **pomanjkljivi** (npr. manjkajoči atributi). Pri združevanju podatkov lahko omejimo izbor atributov.

Poenotiti moramo kodni nabor, obliko, kvaliteto in agregacijo. Šele nato lahko podatke združimo. Za zbiranje zapisov uporabimo stavke SELECT združene z UNION[34]. Če nam v kakšni tabeli manjka atribut, ga lahko nadomestimo z vre-

[34]Pri uporabi UNION ALL bi se lahko pojavili podvojeni zapisi.

dnostjo NULL. Pri uparjanju podatkov uporabimo SQL konstrukt FULL OUTER JOIN, ki ga poznata tako PostgreSQL kot Microsoft SQL Server.

PostgreSQL

PostgreSQL omogoča, da z uporabo funkcij *convert*, *convert_from* in *convert_to* spremenimo kodni nabor uvoženih podatkov. Pri podatkih kodiranih v binarni obliki pa lahko uporabimo funkcijo *decode*. Za poenotenje oblike in kvalitete imamo na voljo veliko različnih matematičnih in drugih funkcij, lahko pa v ta namen napišemo tudi PL/pgSQL funkcijo. Najbolj problematično je poenotenje agregacije. Pri nominalnih tipih vrednosti atributov ne moremo agregirati, razen, če bi te vrednosti združili v eno vrednost kot strukturiran atribut (npr. XML, JSON, polje (an. array) vrednosti). Kako bi izračunali povprečje med vrednostma "Ljubljana" in "Maribor", če bi atribut predstavljal ime mesta? Sicer pa lahko v PostgreSQL napišemo lastne agregacijske funkcije. Primer je v prilogi te naloge na strani 119.

Microsoft SQL Server

Pri Microsoft SQL Server možnosti naknadnega spreminjanja kodnega nabora nizov nisem našel. Omogoča pisanje lastnih agregacijskih funkcij, vendar je uporaba neprimerno težja kot pri PostgreSQL.

5.10 CRISP-DM 3.5 (Formatiranje podatkov)

V koraku formatiranja izvedemo sintaktične modifikacije nad podatki, ki ne spremenijo njihovega pomena. Takšne spremembe zahtevajo nekatera modelirna orodja.

5.10.1 Dodajanje identifikatorja

Nekatera modelirna orodja zahtevajo atribut, ki je identifikator zapisa. Če ga še ni, ga lahko dodamo.

PostgreSQL

Pri PostgreSQL to najlažje rešimo s pomočjo funkcije *row_number*, ki vrne številko zapisa znotraj particije, in kopiranja podatkov v novo tabelo, npr.:

```
SELECT row_number() OVER() AS id, *
INTO fruitfly2 FROM fruitfly;
```

Druga možnost bi bila kreiranje novega atributa, nove sekvence, nato pa prirejanje vrednosti z uporabo te sekvence (funkcija *nextval*). Problem pa je, ker pri PostgreSQL atributa ne moremo dodati na začetek ali poljubno mesto, ampak se vedno doda na koncu[35]. Pozicijo stolpca lahko spremenimo le tako, da podatke prestavimo v novo tabelo.

Microsoft SQL Server

Podobno je pri Micosoft SQL Server, kjer v orodju *Management Studio* lahko spremenimo vrstni red atributov, vendar le na praznih tabelah – v ozadju se izvede ponovno kreiranje tabele.

Enostaven način za kreiranje unikatnega atributa bi bil.

```
ALTER TABLE test.dbo.fruitfly ADD id INT;
UPDATE test.dbo.fruitfly SET id=%%physloc%%
```

Lahko pa bi tudi kreirali sekvenco in jo uporabili z uporabo konstrukta NEXT VALUE FOR.

5.10.2 Spreminjanje vrstnega reda atributov

Nekatera orodja imajo zahteve glede vrstrega reda atributov. Npr. prvo polje mora biti unikatni identifikator za vsak zapis, zadnje polje pa mora biti izhodni razred, ki ga mora model napovedati.

Niti PostgreSQL niti Microsoft SQL server ne podpirata spreminjanja vrstnega reda atributov, kar je že omenjeno v prejšnjem podpoglavju, ampak se to lahko naredi s prestavitvijo podatkov v novo tabelo. Kreiranje zadnjega atributa je enostavno, ker se vsak novo kreirani atribut avtomatsko doda na zadnje mesto.

[35]Nekateri SUPB poznajo konstrukt AFTER.

5.10.3 Spreminjanje vrstnega reda zapisov

V nekaterih orodjih je pomemben tudi vrstni red zapisov v podatkovni množici. Modelirno orodje lahko zahteva, da so zapisi sortirani glede na vrednost izhodnega atributa.

V relacijskih podatkovnih bazah lahko pravilni vrsten red enostavno dobimo s konstruktom ORDER BY. Določimo lahko več stolpcev, po katerih se sortirajo zapisi, in jim določimo naraščajoč ali padajoč vrstni red.

Če sortiranje ni izbrano, so zapisi vrnjeni v nedoločenem vrstnem redu, pogosto pa tako kot so bili dodani.

PostgreSQL

Pri PostgreSQL je dejanski vrstni red odvisen od skeniranja, plana združevanja zapisov in vrstnega reda zapisov na disku. Z uporabo konstrukta NULLS FIRST oz. NULLS LAST lahko določimo, ali se NULL vrednosti prikažejo na začetku ali na koncu. Pri sortiranju lahko navedemo tudi vzdevek (an. alias) atributa in celo številko stolpca po katerem naj sortira, začenši z 1, npr.:

```
SELECT "PARTNERS", max("THORAX")
FROM fruitfly
GROUP BY "PARTNERS"
ORDER BY 1;
```

PostgreSQL omogoča tudi urejanje celotnega rezultata množic pridobljenih z UNION, INTERSECT ali EXCEPT[36]. Omeniti velja še, da na hitrost razvrščanja zapisov lahko vpliva tudi definiran indeks na atributih, kjer sortiramo.

Microsoft SQL Server

Tudi pri Microsoft SQL Server lahko zapise poizvedbe razvrščamo po zaporedni številki atributa in po vzdevku. Enako je s konstruktom UNION, ne pozna pa konstrukta NULLS FIRST.

[36]Unija, presek in razlika množic.

5.11 Pregled primernosti PostgreSQL in Microsoft SQL server za 2. in 3. korak CRISP-DM

Kriterij	**PostgreSQL**	**Microsoft SQL**
Ekonomski in pravni kriteriji		
Cena	brezplačno	plačljivo (razen Express edition)
Odprta koda	da (PostgreSQL licenca)	ne
Razširljivost	SQL funkcije, funkcije vključenih (*PL/PgSQL, PL/Tcl, PL/Perl, PL/Python*) in dodatnih (*PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, PL/sh*) proceduralnih jezikov, interne funkcije (lastne agregacijske funkcije, tipi in operatorji, PGXS razširitve), C funkcije	T-SQL (uporabniške funkcije, kreiranje tipov, agregacijskih funkcij)
Vmesniki	tekstovni (psql) in grafični kot ločena orodja (pgAdmin III, phpPgAdmin, TeamPostgreSQL, Adminer, TOra, SQirreL SQL ...)	grafični (SQL Server Management Studio in ločena orodja drugih proizvajalcev)
Podprti formati	CSV, TSV, binarni PostgreSQL format	CSV, TSV
Dostop do drugih SUPB	dblink, DBI-link, foreign data wrappers (implementacije standarda SQL/MED)	prek .NET Framework data provider (Microsoft SQL Server, Oracle, ODBC, OLE DB prek Linked Servers)
Tehnične omejitve		
Največje število atributov	1600	1024 (pri redko posejanih vrednostih 30000 z omejitvijo 8018 bajtov na zapis)

Kriterij	**PostgreSQL**	**Microsoft SQL**
Omejitev velikosti zapisa	1,6 TB	8060 bajtov (pri varchar, nvarchar, varbinary, sql_variant in CLR uporabniških tipih lahko pride do preliva – odvečni podatki se shranijo na drugo mesto), na celico je omejitev 8000 bajtov
Omejitev količine podatkov na tabelo	32 TB (velikost podatkovne baze je neomejena)	Pri express edition je velikost celotne podtkovne baze omejena na 10 GB, sicer 524 PB.
Podprti operacijski sistemi	Linux, Windows, FreeBSD, OpenBSD, NetBSD, Mac OS X, AIX, HP/UX, IRIX, Solaris, Tru64 Unix, UnixWare	Windows

Poglavje 6

Orodja za podatkovno rudarjenje z vidika priprave podatkov

Namen poglavja ni pregled vseh možnih orodji za podatkovno rudarjenje niti ni namen primerjati orodij z vidika podpore algoritmom podatkovnega rudarjenja. Namen je primerjati orodja za podatkovno rudarjenje z vidika priprave podatkov. Nekatera izmed teh orodij vključujejo gradnike, ki jih imajo ponavadi ETL orodja. Nekatera orodja so omenjena zaradi svojih datotečnih formatov ali zaradi drugačnega načina pripreve podatkov.

6.1 Orange

Orange je orodje za podatkovno rudarjenje, ki ga omenjamo, ker uporablja prilagojeno verzijo TSV formata. Razlika je v tem, da glava datoteke ni le prva vrstica ampak prve tri. Prva vsebuje imena stolpcev, druga navaja tipe atributov, tretja pa opcijske zastave[1].

Vrednost atributa lahko pri tipu *basket* vsebuje več podatkov. Ti so navedeni v obliki *ime=vrednost* in med seboj ločeni s presledki. Če vrednost ni navedena,

[1]Tipi atributov so: discrete (d), continuous (c), string, basket. Opcijske zastave pa: ignore (i), class (c), multiclass, meta (m).

se smatra, da ima podatek vrednost 1. Tip backet uporabljamo za tekstovne ali redko posejane podatke. Primer Orange TSV datoteke iz dokumentacije[27]:

```
K       Ca      b_foo     Ba  y
c       c       basket    c   c
        meta              i   class
0.06    8.75    a b a c   0   1
0.48            b=2 d     0   1
0.39    7.78              0   1
```

V tem primeru so imena atributov K, Ca, b_foo, Ba in y navedena v prvi vrstici. V drugi vrstici so navedeni tipi atributov (vsi so zvezni razen atributa b_foo, ki je tipa *basket*). V tretji vrstici pa so opcijske zastave (*class* pove, da gre za razredno spremenljivko). Vse preostale vrstice vsebujejo podatke.

Orange podpira tudi *basket* datoteke. Primer iz Orange dokumentacije[27]:

```
nobody, expects, the, Spanish, Inquisition=5
our, chief, weapon, is, surprise=3, surprise=2, and, \
        fear,fear, and, surprise
our, two, weapons, are, fear, and, surprise, and, \
        ruthless, efficiency
to, the, Pope, and, nice, red, uniforms, oh damn
```

Orange zna brati tudi klasične CSV in TSV datoteke, le da mora pri njih tip atributa uganiti. Skrajno desni stolpec pa pri tem obravnava kot razredno spremenljivko.

Orange-ov SQL vmesnik omogoča dostop do relacijskih SUPB. Podpira MySQL, PostgreSQL, SQLite in ODBC vire. To je zelo uporabno, saj lahko po uvozu podatkov v SUPB in izvedbi transformacij beremo podatke od tam. Z verzijo 3, ki v času pisanja naloge še ni izšla, pa Orange dodaja možnost za tesnejšo povezanost s SUPB in tako odpravlja tudi težave s pomnilnikom pri večjih podatkih. To pomeni, da se podatki ne prenašajo več iz SUPB in analizirajo lokalno, ampak ostanejo v SUPB, operacije pa se prevedejo v SQL poizvedbe in shranijo v podatkovno bazo. Tak način izkorišča prednosti relacijskih SUPB in minimizira prenos podatkov med SUPB in orodjem za podatkovno rudarjenje[2].

[2]http://blog.biolab.si/2015/05/05/working-with-sql-data-in-orange-3/

6.2 Weka

Orodje Weka na tem mestu omenjamo predvsem zaradi njegove popularnosti in formata ARFF, ki je njegova posebnost. Format je opisan v poglavju podatkovnih formatov. Weka lahko uporablja tudi CSV datoteke, a z nekaj pomanjkljivostmi, ker v CSV datotekah ni opisa tipov atributov.

Tudi Weka omogoča (prek razreda DatabaseUtils in funkcije InstanceQuery) dostop do SUPB in sicer tistih, do katerih lahko dostopamo prek JDBC ali ODBC (Microsoft Access, SQL Server, MySQL, Oracle, PostgreSQL, SQLite). Nima orodij za korake priprave podatkov, a mu podporo nudi ETL orodje Pentaho Data Integration.

6.3 RapidMiner

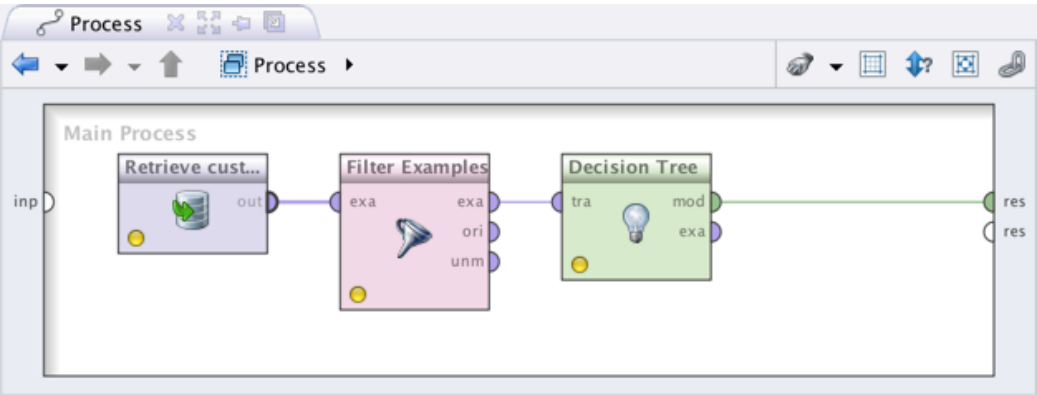
RapidMiner omenjamo, ker gre za zelo dodelano grafično orodje, ki je primerno za izvajanje procesa podatkovnega rudarjenja tudi za večje podatke. Poleg klasičnega načina, kjer se vsi podatki pred podatkovnim rudarjenjem uvozijo v pomnilnik, omogoča še izvajanje podatkovnega rudarjenja nad podatki shranjenimi v podatkovni bazi. Omogoča tudi branje in pisanje datotek v Hadoop HDFS (*Hadoop Distributed File System*) sistem[3], ki se ponavadi razteza prek več računalnikov in je zato nismo obravnavali.

RapidMiner kot vire podatkov (prek JDBC) podpira relacijske (*MySQL, PostgreSQL, Sybase, Microsoft SQL, Oracle, Microsoft Access*) in nerelacijske SUPB (*MongoDB, Cassandra*). Zna brati tudi iz tekstovnih (*CSV, ARFF, XML*) in binarnih formatov (*HDF5, Excel, SPSS*). Del vmesnika Rapid Miner studio je prikazan na sliki 6.1.

Proces zajema podatkov in ostale korake kreiramo s postavljanjem in povezovanjem gradnikov. Podpira ETL transformacije kot so normalizacija in diskretizacija atributov, filtriranje in sortiranje zapisov, transpozicija vrstic in stolpcev, obteževanje atributov idr. Po izvedbi podatkovnega rudarjenja pa lahko ustvarjene modele shrani v PMML.

Ob prvem zagonu RapidMiner moramo nastaviti mapo, ki bo služila kot re-

[3]Prek orodja RapidMiner Radoop



Slika 6.1: Izdelava procesa pridobivanja podatkov in podatkovnega rudarjenja v RapidMiner. Vir: http://docs.rapidminer.com

pozitorij za podatke, pa tudi za shranjevanje projektov. RapidMiner v ta namen uporablja SUPB, ki je del produkta. Kateri SUPB je v ozadju, ni dokumentirano.

Na žalost pa je RapidMiner v verzijah, ki omogočajo uporabo SUPB za podatkovno rudarjenje, plačljiv in zato morda ne izpolni zastavljenega ekonomskega kriterija.

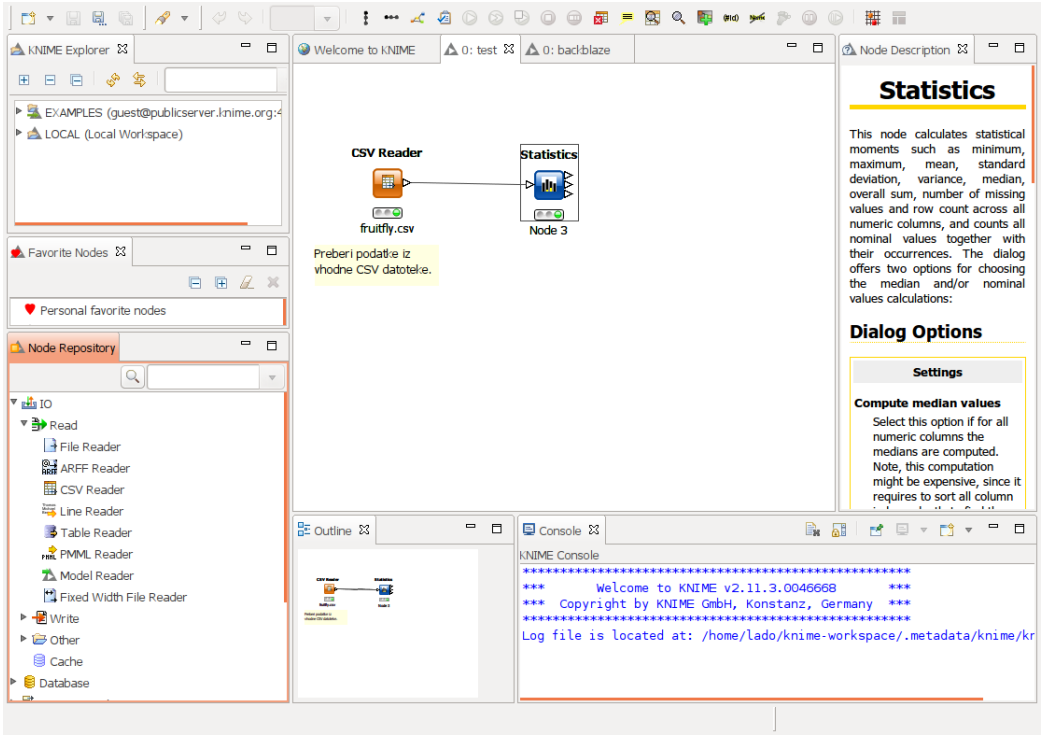
6.4 Knime Analytics Platform

Knime Analytics Platform je odprtokodno[4] grafično orodje za podatkovno rudarjenje, ki omogoča dostop do podatkov, izvajanje transformacij, vizualizacij in poročil. Sam produkt že vsebuje veliko razširitev, lahko pa namestimo tudi komercialne razširitve.

Knime omogoča uporabo tekstovnih datotek (ARFF, CSV, TSV) in različnih SUPB, do katerih dostopamo prek ODBC ali JDBC (PostgreSQL, SQLite, Microsoft SQL Server, Oracle, MySQL, IBM DB2). Pri pri vsakem podatkovnem viru lahko določimo, ali naj se podatki shranijo v pomnilnik ali na disk. Privzeto se majhne tabele shranijo v pomnilnik, večje pa na disk.

Knime omogoča tudi nekaj ETL transformacij. Agregacijo nad stolpci lahko naredimo na podlagi vzorca imena stolpcev, kar nam pride prav pri velikem številu atributov s podobnim imenom in enakim tipom podatkov.

[4]GPL licenca.

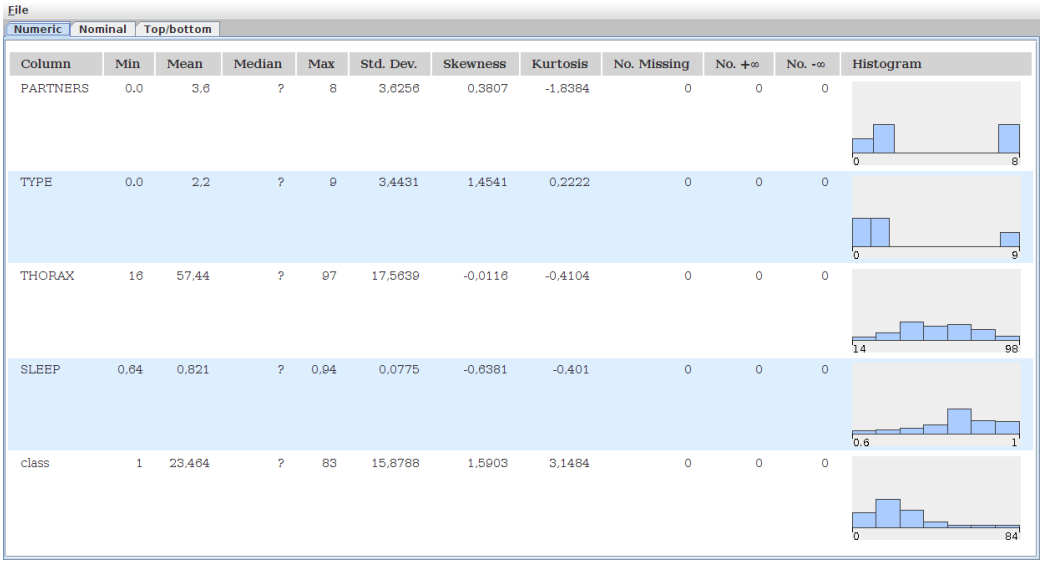


Slika 6.2: Grafični vmesnik Knime Analytics Platform.

Proces izvajanja podatkovnega rudarjenja rišemo v grafičnem okolju, ki je prikazano na sliki 6.2. Gradnike postavljamo na delovno površino, jih konfiguriramo in med sabo povezujemo. Pri vsakem gradniku lahko pregledujemo tudi njegove rezultate: pri uvozu lahko vidimo podatke, ki se bodo uvozili, pri vizualizaciji pa npr. grafe in histograme. Na sliki 6.3 je prikazan primer osnovne statistike atributov.

6.5 SUPB z vgrajeno podporo za podatkovno rudarjenje

V tem delu omenjamo SUPB, ki imajo vgrajeno podporo za podatkovno rudarjenje, a ne zahtevajo postavitve podatkovnega skladišča.



Column	Min	Mean	Median	Max	Std. Dev.	Skewness	Kurtosis	No. Missing	No. +∞	No. -∞	Histogram
PARTNERS	0.0	3,6	?	8	3,6256	0,3807	-1,8384	0	0	0	0 8
TYPE	0.0	2,2	?	9	3,4431	1,4541	0,2222	0	0	0	0 9
THORAX	16	57,44	?	97	17,5639	-0,0116	-0,4104	0	0	0	14 98
SLEEP	0,64	0,821	?	0,94	0,0775	-0,6381	-0,401	0	0	0	0.6 1
class	1	23,464	?	83	15,8788	1,5903	3,1484	0	0	0	0 84

Slika 6.3: Prikaz osnovne statistike v aplikaciji Knime Analytics Platform.

6.5.1 Oracle

Oracle ponuja funkcionalnosti podatkovnega rudarjenja kot SQL funkcije znotraj svojega SUPB. Operacije se načrtujejo v grafičnem orodju *Oracle Data Miner Workflow*, ki je razširitev za orodje SQL Developer. Pri načtrovanju procesa podatkovnega rudarjenja se načrtujejo tudi vse potrebne transformacije za pripravo podatkov. Načrtovan proces se nato shrani v strukturo, ki ji Oracle pravi supermodel, saj poleg izbranih tehnik podatkovnega rudarjenja vključuje tudi vse potrebne transformacije za pripravo podatkov.

Uporabnik lahko uporabi avtomatske transformacije (od verzije 11g orodje samo predlaga transformacije), jih dopolni ali pa jih definira sam. Vnesejo se v transformacijsko tabelo, kamor se dodajo tudi učni podatki. Podatke, ki vsebujejo transformirane atribute, lahko pregledujemo kot poglede v podatkovni bazi. Oracle podpira naslednje transformacije: diskretizacija in normalizacija atributov, obravna izstopajočih in manjkajočih vrednosti ter odstranjevanje nepotrebnih atributov. Za kategorične in numerične podatke podpira tudi vgnezdene podatkovne tipe.

Prvi korak v procesu podatkovnega rudarjenja je izdelava tabele učnih podatkov. Če so podatki že v eni sami tabeli in zapisi že vsebujejo vse informacije, ta

korak ni potreben. Če pa se podatki nahajajo v več tabelah, je potrebno izdelati pogled (an. view). Včasih so pri tem potrebne agregacije, za katere se lahko uporabijo tudi vgnezdeni stolpci. Pred podatkovnim rudarjenjem je potrebno spremeniti nekatere tipe atributov v bolj osnovne.

6.5.2 Microsoft SQL Server Analysis Services in SQL Server Integration Services

Microsoft SQL Server ima od verzije Standard naprej vključeni dve orodji, ki sta uporabni za namen naloge. To sta orodje za pripravo podatkov in orodje za samo podatkovno rudarjenje.

Microsoft SQL Server Analysis Services (SSAS) vsebuje funkcije za podatkovno rudarjenje in deluje nad OLAP ali tabelaričnimi podatki (tip instance se določi že ob namestitvi). Tabelarične podatke se kreira kot podatkovne baze z uporabo orodja *SQL Server Data Tools (SSDT)*. Lahko jih uporabljamo tudi kot razširitev za podatkovno rudarjenje za Excel. Za najpomembnejše podatke lahko uporabnik preko orodja *Migration Advisor* določi, da so vedno na voljo v pomnilniku (t. i. *in-memory* način) in so zato hitro dostopni.

Orodje za pripravo podatkov pa se imenuje *SQL Server Integration Services (SSIS)*. To je ETL orodje z grafičnim vmesnikom za Microsoft SQL Server. Vsebuje transformacije za pridobivanje in združevanje podatkov ter izvajanje agregacij. Vsebuje tudi orodja za nadzor izvajanja transformacij in aplikacijske programerske vmesnike. Podpira precej podatkovnih virov: Excel, CSV, XML, Oracle, Sybase, MySQL, PostgreSQL, Microsoft Access idr.

Izdelane transformacije se shranijo v paket transformacij. Ta se shrani v datotečni sistem, v SQL Server ali v hrambo paketov. Paket se lahko ustvari tudi s čarovnikom za uvoz in izvoz podatkov. Vsebuje *kontrolne in podatkovne elemente*. Prve izvajajo funkcije in kontrolirajo vrstni red izvajanja elementov procesa, druge pa pridobivajo podatke, jih transformirajo in shranjujejo v podatkovne ponore. Glavni kontrolni elementi so opravila, vsebniki za druge elemente in omejitve pri določanju prioritet. Glavni elementi toka podatkov pa so viri, transformacije in cilji. V paketu mora biti vsaj en kontrolni element.

Poglavje 7

ETL orodja

Glede na [28] je ETL proces pridobivanja in združevanja podatkov iz več virov v podatkovno skladišče, ki omogoča uporabnikom, da delajo na eni združeni množici podatkov. V našem primeru kot cilj uporabljamo SUPB. V tem poglavju je pregled nekaj najbolj uporabnih ETL orodij.

7.1 csvkit

Csvkit ni popolno ETL orodje, ampak je ožje usmerjeno orodje, ki ga uporabljamo iz ukazne vrstice. Izdan je pod licenco MIT. Namen orodja je delo s CSV datotekami: uvoz, procesiranje in izvoz. Brati zna bzip ali gz2 arhivirane datoteke. Csvkit ponuja naslednja uporabna orodja:

- **in2csv** – s tem ukazom uvozimo podatke iz drugih CSV datotek, datotek s podatki fiksne širine, GeoJSON, JSON, XLS, XLSX.
- **csvlook** – z njim pregledujemo CSV podatke. Izpis je podoben privzetemu izpisu v psql. Uporaben je v kombinaciji s csvcut.
- **csvcut** – s tem ukazom izberemo, kateri atributi naj se izpišejo. Imena atributov pridobimo z ukazom *csvcut -n data.csv*. Če želimo pridobiti le podmnožico stolpcev lahko npr. uporabimo *csvcut -c 2,5,6 data.csv*. Lahko navedemo tudi imena atributov, npr. *csvcut -c county,item_name,quantity data.csv*.

- **csvstat** – ukaz izvede osnovno statistiko nad podatki (tip podatkov, ali vsebujejo ničelne vrednosti, št. unikatnih vrednosti, najpogostejše vrednosti, najdaljša dolžina niza, minimalna in maksimalna vrednost, vsota, povprečje, mediana, standardni odklon, število primerov in atributov).

- **csvgrep** – podobno kot pri linux ukazu grep lahko iz CSV izluščimo ujemajoče vrstice, npr.:

```
$ csvcut -c county,item_name,total_cost data.csv |
  csvgrep -c county -m LANCASTER | csvlook
```

  Če namesto parametra *-m* uporabimo parameter *-r*, lahko podamo regularni izraz, s katerim se mora vrednost ujemati.

- **csvsort** – podatke lahko razvrstimo po stolpcih, naraščajoče ali padajoče. Podamo lahko tudi več stolpcev razvrščanje. Razvrščamo lahko tudi v obratnem vrstnem redu, vendar to velja za vse navedene stolpce. Npr.

```
$ csvcut -c county,item_name,total_cost data.csv |
  csvgrep -c county -m LANCASTER |
  csvsort -c total_cost -r |  csvlook
```

- **csvjoin** – podobno kot lahko v SQL z JOIN poizvedbo povežemo dve tabeli, lahko s csvjoin povežemo dve CSV datoteki in ju združimo. Npr.

```
$ csvjoin -c fips data.csv population.csv > joined.csv
```

  V tem primeru se atributi iz tabele population.csv dodajo na konec podatkov iz data.csv glede na ujemanje atributa fips.

- **csvstack** – pogosto so velike podatkovne množice razbite v več datotek. Ukaz csvstack omogoča, da združimo več tako razbitih datotek. Glave CSV datotek se morajo pri tem ujemati. Pri združevanju lahko uporabimo tudi opcijo *-g*, ki doda atribut, ki pove, iz katere datoteke je bil pridobljen posamezen primer.

- **csvsql, sql2csv** – omogoča shranjevanje podatkov v podatkovno bazo oz. pridobivanje podatkov iz nje. Privzeto csvsql ustvari SQL stavek za kreira-

nje tabele. S parametrom -i lahko določimo, katero vrsto podatkovne baze uporabljamo[1]. Primer:

```
$ csvsql -i sqlite joined.csv
CREATE TABLE joined (
        state VARCHAR(2) NOT NULL,
        county VARCHAR(10) NOT NULL,
        ...
);
```

Podatke iz CSV lahko uvozimo tudi direktno v podatkovno bazo, le na istem računalniku se mora nahajati. Npr.

```
$ sudo -u postgres csvsql --db postgresql:///test
  --insert joined.csv
```

Csvkit sam kreira vse potrebne atribute in vstavi podatke.

Csvkit je napisan v jeziku *python*, za dostop do različnih SUPB pa uporablja ORM SQLAlchemy.

Z ukazom sql2csv lahko podatke izvozimo iz podatkovne baze v CSV, npr.

```
$ sudo -u postgres sql2csv --db postgresql:///test
  --query "select * from joined" > joined2.csv
```

Pri manjših podatkih lahko po CSV poizvedujemo z SQL stavki, vendar se pri tem vsi podatki naložijo v pomnilnik:

```
$ csvsql --query "select county,item_name
  from joined where quantity > 5;"
  joined.csv | csvlook
```

- **csvjson** – pretvori CSV datoteko v JSON (ali GeoJSON) format.
- **csvpy** – omogoča pregledovanje CSV podatkov v python lupini. Csvkit lahko uporabljamo tudi kot knjižnico v python skriptah (import csvkit).

[1]Podprte so naslednje podatkovne baze: Microsoft Access, Sybase, SQLite, Informix, Firebird, MySQL, Oracle, Maxdb, PostgreSQL, Microsoft SQL Server

- **csvformat** – s tem ukazom lahko CSV izvozimo v prilagojen format, npr. v nestandardni CSV ali v TSV. CSV lahko pretvorimo v TSV s kratkim ukazom:

```
$ csvformat -T data.csv
```

- **csvclean** – csvkit je uporaben tudi za iskanje sintaktičnih napak v CSV datotekah.

7.2 Pentaho Data Integration

Pentaho Data Integration je odprtokodno grafično ETL orodje oz. zbirka orodij. Pred prevzemom s strani Pentaha je bilo orodje poimenovano **Kettle**[2].

Pentaho loči med izvajanjem transformacij in opravil. Pri transformacijah prenašamo in spreminjamo podatke od izvora do ponora, opravila pa se izvajajo na višjem nivoju. Izvajajo transformacije in druga opravila, preverjajo dostopnost virov, obveščajo o rezultatih po e-pošti ipd. Koraki opravila se vedno izvajajo po vnaprej določenem vrstnem redu, koraki pri transformacijah pa se lahko izvajajo tudi vzporedno.

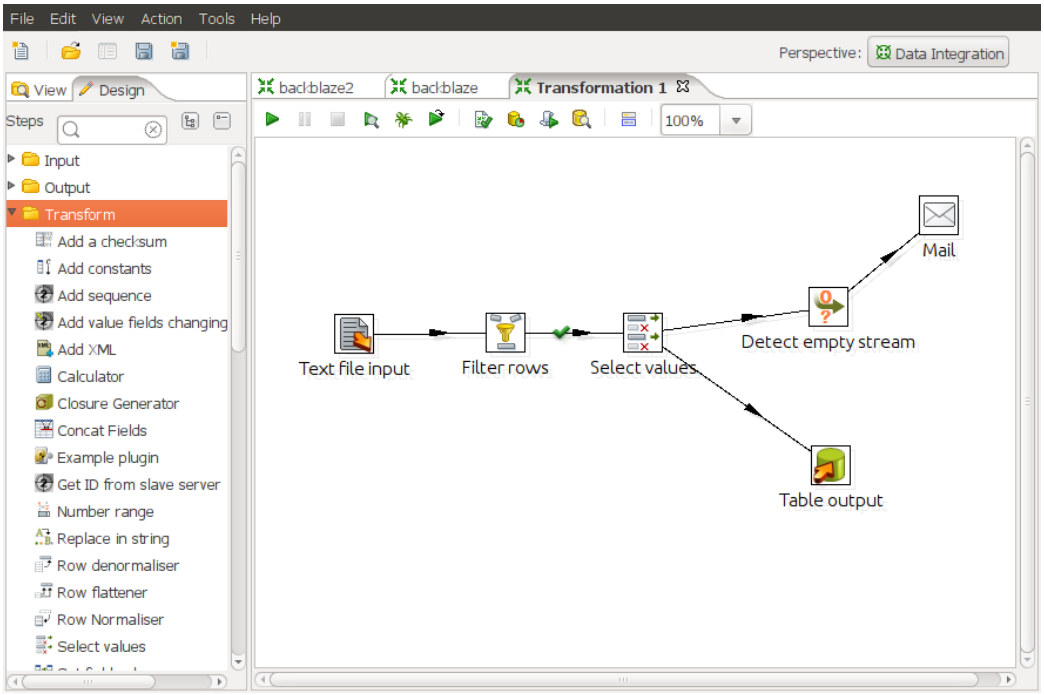
Pentaho Data Integration je sestavljen iz štirih komponent: **Spoon**, **Pan**, **Kitchen** in **Carte**. *Spoon* je grafično orodje za izvajanje transformacij in opravil. *Pan* je tekstovno orodje za zaganjanje transformacij, ki smo jih pred tem izdelali v orodju *Spoon*. *Kitchen* je tekstovno orodje, s katerim lahko izvajamo opravila. Uporabno je za izvajanje ponavljajočih operacij, npr. takšnih, ki se izvajajo dnevno. *Carte* je preprost spletni strežnik, ki omogoča, da izvajamo in nadzorujemo transformacije in opravila iz oddaljene lokacije.

Pentaho lahko opravila in transformacije (v XML obliki) shrani v datotečni sistem ali repozitorij, ki ga predstavlja podatkovna baza uporabljena v ta namen.

Pentaho ponuja kar precej gradnikov za izvajanje transformacije, lahko pa prek razširitev dodajamo tudi svoje. V orodju Spoon transormacije kreiramo tako, da gradnike premikamo na delovno površino, jih povezujemo in nastavljamo parametre. Komponente so organizirane v skupine. Za začetek transformacije ponavadi

[2]K.E.T.T.L.E je rekurzivna kratica za *Extraction Transformation Transport Load Environment*.

iščemo komponente v skupini vhodnih komponent, za konec pa v skupini izhodnih. Uporabniški vmesnik je prikazan na sliki 7.1.



Slika 7.1: Uporabniški vmesnik Pentaho Data Integration.

Preprost primer uporabe, pri katerem sem Pentaho uporabil v praksi, je bilo pridobivanje strukturiranih podatkov iz XML dokumenta v tabelo v Excel dokumentu. Na delovno površino sem dodal gradnika za vhodni XML dokument, za izhodni Excel dokument in ju povezal. Pri XML dokumentu sem določil XPath pot do ponavljajočih se elementov in atribute, ki sem jih želel pridobiti v XML. Po zagonu transformacije sem dobil izhodno Excel datoteko. Edina težava, ki sem jo imel pri tem, je bilo ločilo za decimalna mesta, ki ga je predstavljal znak pika in ne vejica.

7.3 Scriptella

Scriptella je zelo majhno odprtokodno[3] ETL orodje, veliko le okoli 2,5 MB. Grafičnega vmesnika nima, a je kljub temu uporabno orodje. Vse ETL operacije

[3]Apache licenca.

se določijo prek XML datoteke[4], transformacije pa se izvedejo z zagonom skripte *scriptella.sh*. Izvedba skripte zahteva izvajalno okolje *Java*.

Scriptella pozna tri osnovne tipe elementov: **povezava**, **skripta** in **poizvedba**. S povezavo določimo podatkovne vire, ki jih bo Scriptella uporabila. Uporabimo lahko JDBC (PostgreSQL, Microsoft SQL, Oracle, Sybase, Excel ...) ali druge vire (CSV, LDAP, MongoDB, SMPT, XPath ...). V skripti določimo kodo, ki naj se izvede pri pridobivanju podatkov. Uporabimo lahko več jezikov kot so SQL, JavaScript idr. Skripta izpisuje podatke v ciljno povezavo in se lahko izvaja za vse rezultate poizvedbe po podatkovnem viru.

Za uporabo različnih podatkovnih virov moramo JDBC gonilnike[5] prekopirati v mapo *lib*, ali pa povezavi dodamo atribut *classpath*, ki določa pot do gonilnika v datotečnem sistemu. Primer Scriptella skripte:

```
<!DOCTYPE etl SYSTEM
  "http://scriptella.javaforge.com/dtd/etl.dtd">
<etl>
  <connection id="db" driver="mysql"
    classpath="/usr/share/java/mysql.jar"
    url="jdbc:mysql://localhost:3306/knjiznica"
    user="test" password="test"/>
  <connection id="out" driver="text" url="out.txt"/>
  <query connection-id="db">
    SELECT ime, priimek
    FROM clan
    ORDER BY priimek ASC, ime ASC
    <script connection-id="out">
      $ime, $priimek
    </script>
  </query>
</etl>
```

Prednost skriptele je odprta koda, njena majhnost in zelo preprosta osnovna filozofija. Lahko bi služila kot alternativa opisanim transformacijam v PMML. Transformacija je opisana v tekstovni obliki in bi zato lahko bila prenosljiva med različnimi ETL orodji.

[4]Privzeto ime je "etl.xml".

[5]Glej: http://scriptella.javaforge.com/reference/drivers.html

Poglavje 8

Testna metodologija in rezultati poskusov

8.1 Testno okolje

Vsem testiranim orodjem je bilo potrebno zagotoviti enake pogoje za izvajanje. Enaki procesorji, enaka količina in hitrost pomnilnika ter enako hiter disk. Edino kar ni bilo možno poenotiti, je operacijski sistem. Veliko orodij je namreč vezanih na nek operacijski sistem. Za zagotovitev enakovrednih pogojev je bilo za namen naloge uporabljeno virtualizacijsko okolje *Oracle VirtualBox*[1], vsakemu virtualnemu računalniku pa je bilo dodeljeno *4 GB pomnilnika in 2 procesorja*. Vsi virtualni računalniki so imeli dostop do istih testnih podatkov prek deljene mape. Pri testiranju kombinacije večih orodij, npr. ETL in SUPB, so morala biti vsa orodja nameščena na istem virtualnem računalniku.

Ker pa je bilo testiranih orodij kar nekaj, bi bilo nepraktično ročno postavljati operacijski sistem za vsako orodje posebej. Zato je bil poleg VirtualBox uporabljen tudi *Vagrant*[2]. Slednji omogoča hitro postavitev virtualnega računalnika z uporabo osnovne postavitve in konfiguracijske datoteke, ki pogosto uporablja še namestitveno skripto. Za nekatera orodja so Vagrant konfiguracijske datoteke in namestitvene datoteke že na voljo na spletu in jih je potrebno le prilagoditi, za

[1] https://www.virtualbox.org
[2] https://www.vagrantup.com

preostala orodja pa jih je potrebno napisati.

V Vagrant konfiguracijski datoteki (*Vagrantfile*) lahko nastavimo vse parametre za virtualni računalnik. Navedemo lahko tudi vse ukaze, ki se morajo izvesti ob namestitvi. Primer konfiguracijske datoteke:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise64"
  config.vm.network "forwarded_port", guest: 5432, host: 15432
  config.vm.network "public_network", type: "dhcp"
  config.vm.synced_folder "/home/lado/VIRTUALKE/DATA", \
    "/DMDATA"
  config.vm.provider "virtualbox" do |v|
    v.customize ["modifyvm", :id, "--cpuexecutioncap", "50"]
    v.memory = 4096
    v.cpus = 2
    v.gui = true
    v.customize ["modifyvm", :id, "--usb", "off"]
  end
  config.vm.provision :shell, :path => "bootstrap.sh"
end
```

S takšno datoteko povemo, katera osnovna postavitev oz. operacijski sistem naj se uporabi, koliko pomnilnika in procesorjev naj ima, ali naj se uporabi grafično okolje ipd. Navedemo lahko tudi skripto (v našem primeru "bootstrap.sh"), ki ob namestitvi v njej izvede ukaze z administratorskimi pravicami. Z ukazom "vagrant up" spremenimo takšno konfiguracijsko datoteko v virtualni računalnik.

8.2 Testni podatki

Testnih podatkov ni bilo enostavno najti. Prvič zato, ker veliko virov podatkov že vsebuje prečiščene podatke, in drugič zato, ker sem iskal podatke v velikosti pomnilnika. Pri testiranju sem uporabil več različno velikih podatkov, a na tem

mestu omenjam le najmanjše in največje:

- **Fruitfly:** Za hitro preizkušanje orodij sem uporabil zelo znan primer "fruitfly", ki vsebuje podatke o tem ali povečana reprodukcija pri moških komarjih zmanjša njihovo življenjsko dobo. Vsebuje le 5 atributov in 125 primerov. Primeri iz teh podatkov so navedeni tudi v poglavju primerjave dveh SUPB. Vir: http://mldata.org/repository/data/viewslug/datasets-numeric-fruitfly/

- **BackBlaze:** Končni testni podatki so veliki 3,6 GB. Vsebujejo 85 atributov in 17636994 primerov. Gre za podatke o odpovedi diskov podjetja Backblaze, ki se ukvarja z arhiviranjem podatkov. Gre za podatke o 41000 diskih velikih med 1 TB in 6 TB. Za vsak disk dnevno vzamejo podatke iz S.M.A.R.T. statistike in en tak posnetek je ena vrstica v podatkih. Vsaka datoteka predstavlja podatke za en dan – podatki so za leti 2013 in 2014. Zapisani so v formatu CSV. Atributi so datum, serijska št. diska, št. modela, kapaciteta v bajtih, ali ima napako (vrednost 1 na dan, ko se disk pokvari in zamenja, sicer 0), sledi 80 S.M.A.R.T. atributov (prebrana in normalizirana vrednost za 40 statistik). V podatkih je tudi precej neznanih vrednosti, ker nekateri diski ne poročajo vseh S.M.A.R.T. vrednosti. Nekatere statistike imajo lahko pri različnih proizvajalcih in modelih tudi različen pomen. Včasih vrnejo statistike nerazumno visoke vrednosti, npr. da disk obratuje več kot 10 let. Okvarjen disk je dan po okvari zamenjan z novim diskom, ki ima drugačno serijsko številko. Poleg tega se po potrebi dodajajo tudi novi diski, zato se število diskov povečuje.
  Vir: https://www.backblaze.com/hard-drive-test-data.html

8.3 Zahtevane transformacije

Pri primeru Backblaze je bil cilj pripraviti podatke na sledeč način:

- Za vsak disk, ki je določen s serijsko številko, želimo imeti v podatkih le en zapis. Ta mora poleg podatkov o disku (proizvajalec, model) vsebovati tudi zadnji dan obratovanja, število dni obratovanja, ali je prišlo do okvare in zadnje vrednosti S.M.A.R.T. atributov.

- Odvečni S.M.A.R.T atributi se morajo odstraniti. To so npr. tisti, ki vsebujejo same nedefinirane vrednosti. Če imamo za vsako S.M.A.R.T meritev tako surovo kot normalizirano vrednost, je dovolj, da ohranimo le eno izmed njiju, saj bi med njima morala biti vrednost korelacije enaka 1.
- Vrednosti numeričnih atributov morajo biti diskretizirane.
- Pripravljeni podatki morajo biti na koncu shranjeni v SUPB.

Na podlagi tako pridobljenih podatkov bi lahko iskali znanje o tem, kateri S.M.A.R.T atributi in njihove vrednosti bi pri posameznih proizvajalcih lahko napovedovali okvaro diska.

8.4 Priprava podatkov z uporabo PostgreSQL

Prvi korak je bil uvoz podatkov. Stavek za kreiranje prazne SQL tabele sem pridobil z uporabo orodja *csvkit*, le nekatere tipe atributov sem moral popraviti. Ker ukaz COPY podpira uvažanje le ene CSV datoteke naenkrat, sem napisal zelo preprosto bash skripto (uvoz.sh) za uvoz vseh datotek iz mape:

```
#/!bin/bash
csv_dir=$(pwd)/$1/
db='vagrant'
db_table='backblaze'
for csv in $(ls $csv_dir/*.csv);
do
    psql -d $db -c "COPY ${db_table} FROM '${csv}' \
        WITH DELIMITER AS ',' CSV HEADER ENCODING 'UTF8';"
done
```

Skripto sem pognal z ukazom za merjenje časa izvajanja in sicer:

```
time (./uvoz.sh 2013 && ./uvoz.sh 2014)
```

Uvoz je trajal **11 minut in 2 sekundi**. Štetje vrstic s stavkom COUNT je trajalo 48,7 sekund. Vseh uvoženih zapisov je bilo **17636994**.

Za podatkovno tabelo sem z uporabo stavka ANALYZE VERBOSE izdelal statistiko. Ukaz sem pognal za več različnih vrednosti parametra *default_stati-*

stics_target, ki določa vzorčno množico za analizo. Rezultati so prikazani v tabeli 8.1.

Vzorec	Ocenjeno št. zapisov	Čas izvajanja	Odstotek napake
100	17652636	14s	0,089 %
200	17652451	28s	0,087 %
300	17647764	46s	0,061 %
500	17648675	1m 38s	0,060 %
1000	17638304	1m 59s	0,007 %
10000	17636994	9m 52s	0,000 %

Tabela 8.1: Ocena števila zapisov je tudi pri majhnem vzorcu dokaj natančna. Pri največjem vzorcu je ocena popolna, a je čas izvajanja analize dolg.

Z izdelavo statistike sem prišel še do drugih informacij o podatkih. Število atributov je **85**. Rezultat sem dobil s hitro poizvedbo po sistemski tabeli *information_schema.columns*. V nadaljevanju sem iskal nepotrebne atribute. Zanimalo me je ali obstajajo takšni, ki imajo pri vseh primerih nedefinirane vrednosti ali pa so pri vseh primerih enake. V ta namen sem uporabil kreirano statistiko pri vzorcu, velikem 10000 primerov, in poiskal takšne atribute. Uporabil sem takšno poizvedbo:

```
SELECT attname, null_frac, n_distinct, most_common_vals
FROM pg_stats
WHERE schemaname ='public' AND tablename ='backblaze'
  AND n_distinct BETWEEN 0 AND 1 -- st. razlicnih vrednosti
ORDER BY attname;
```

Dobil sem naslednji rezultat:

```
attname              | null_frac | n_distinct | most_common
---------------------+-----------+------------+--------------
smart_15_normalized  |         1 |          0 |
smart_15_raw         |         1 |          0 |
smart_250_normalized |   0.99829 |          1 | {1}
smart_251_normalized |   0.99829 |          1 | {1}
smart_252_normalized |   0.99829 |          1 | {1}
```

```
smart_254_raw         |  0.996277 |          1 | {0}
smart_255_normalized  |         1 |          0 |
smart_255_raw         |         1 |          0 |
```

Atributi **smart_15_normalized, smart_15_raw, smart_255_normalized in smart_255_raw** so neuporabni, ker imajo vsi zapisi nedefinirane vrednosti (null_frac). Odločil sem se, da odstranim tudi vse atribute, ki imajo delež ničelnih elementov večji kot 99,5 %. Za vsakega od njih sem pred tem preveril, ali se morda pojavlja primerih, ko je zabeležena odpoved diska:

```
SELECT failure FROM backblaze WHERE smart_250_normalized=1;
```

Nato sem odstranil odvečne pare S.M.A.R.T. atributov:

```
ALTER TABLE backblaze
DROP smart_13_normalized,  DROP smart_13_raw,
DROP smart_15_normalized, DROP smart_15_raw,
DROP smart_201_normalized, DROP smart_201_raw,
DROP smart_223_normalized, DROP smart_223_raw,
DROP smart_225_normalized, DROP smart_225_raw,
DROP smart_250_normalized, DROP smart_250_raw,
DROP smart_251_normalized, DROP smart_251_raw,
DROP smart_252_normalized, DROP smart_252_raw,
DROP smart_254_normalized, DROP smart_254_raw,
DROP smart_255_normalized, DROP smart_255_raw;
```

Brisanje je trajalo le nekaj milisekund.

Iz statistike v *pg_stats* sem dobil še eno uporabno informacijo. Ker ne potrebujemo tako surovih kot normaliziranih vrednosti, sem preveril, če se delež ničelnih elementov za surove in normalizirane vrednosti ujema. Pri S.M.A.R.T atributih 1, 5, 9, 194 in 197 se te vrednosti niso ujemale. S preprosto SELECT poizvedbo sem ugotovil, da pri teh atributih niso bile izračunane normalizirane vrednosti za vse zapise. Zato sem zavrgel vse normalizirane atribute.

V nadaljevanju sem z uporabo poizvedbe kreiral novo podatkovno tabelo, in sicer tako, da je posamezen zapis predstavljal disk z neko serijsko številko, agregirani atributi pa so vsebovali zadnje S.M.A.R.T. vrednosti. Uporabil sem funkcionalnost, ki jo omogoča PostgreSQL in sicer del SQL stavka DISTINCT ON:

```
CREATE TABLE backblaze_agr AS
```

```
SELECT DISTINCT ON (serial_number, model, capacity_bytes) *
FROM backblaze
ORDER BY serial_number, model, capacity_bytes, date DESC;
```

Pri izvajanju operecije je PostgreSQL uporabljal le enega izmed dveh dodeljenih procesorjev. PostgreSQL ne omogoča, da bi se ena poizvedba izvajala na več kot enem procesorju. Operacijo sem najprej izvajal s privzetimi nastavitvami, nato sem spremenil konfiguracijske parametre[3] na naslednje vrednosti:

```
max_connections = 10
shared_buffers = 2048MB
effective_cache_size = 2048MB
work_mem = 1024MB
```

Pred optimizacijo se je utilizacija procesorja gibala med 40 % in 70 %, po njej pa med 70 % in 95 %. Pri tem je bilo v uporabi večina od 4 GB pomnilnika. Poizvedba se je po optimizaciji izvajala **43 minut in 19 sekund**.

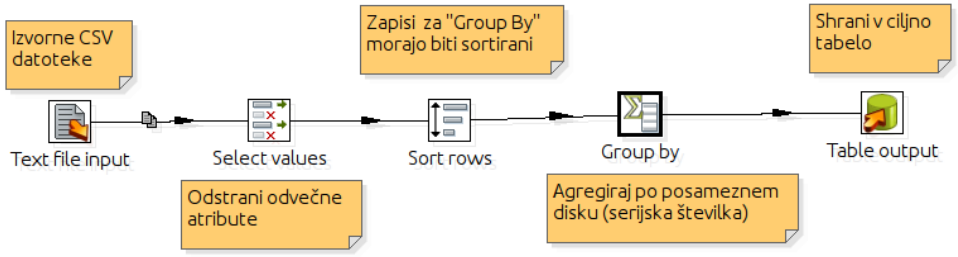
Pridobljena podatkovna množica je imela 49111 zapisov, kar je le 0,28 % zapisov prvotne množice. Poleg tega smo zmanjšali tudi število atributov iz 85 na 35. Žal pa je je PostgreSQL za dokončno pripravo podatkov zmanjkala še funkcija, ki bi izvedla diskretizacijo atributov.

8.5 Priprava podatkov z uporabo Pentaho Data Integration in PostgreSQL

V tem praktičnem poskusu sta se tako PostgreSQL kot Pentaho Data Integration izvajala na istem virtualnem računalniku. Za kreiranje procesa transformacij sem uporabil orodje Spoon, ki je del Pentaho Data Integration. Komponente so razvrščene po skupinah, a ker jih je veliko, je vseeno potrebno nekaj truda za iskanje najprimernejše, za pravilno nastavitev parametrov in za pravilno povezovanje komponent med seboj.

Na sliki 8.1 je proces, ki sem ga izdelal za izvajanje transformacij na primeru BackBlaze. Na levi strani je komponenta “Text File Input”, ki bere podatke iz vhodnih CSV datotek primera Backblaze. Pri nastavitvah komponente sem

[3] http://www.revsys.com/writings/postgresql-performance.html



Slika 8.1: Proces transformacij za primer Backblaze v Pentaho Data Integration

izbral oba direktorija s podatki (2013 in 2014) in navedel filter za izbiro vseh CSV datotek, Pentaho je sam poiskal vse CSV datoteke. Pri tej komponenti sem moral nastaviti tudi kodni nabor in določiti, da so vrednosti atributov ločene z vejico in brez narekovajev.

Naslednja komponenta ("Select Values") omogoča odstranitev atributov, ki jih v nadaljevanju ne potrebujemo. Odstranil sem jih na podlagi poznavanja pogojno uporabnih atributov iz prejšnjega poglavja. Komponente za izločanje neuporabnih atributov nisem našel, našel pa sem produkt *DataCleaner*[4], ki to omogoča in zna za to celo ustvariti Pentaho transformacije.

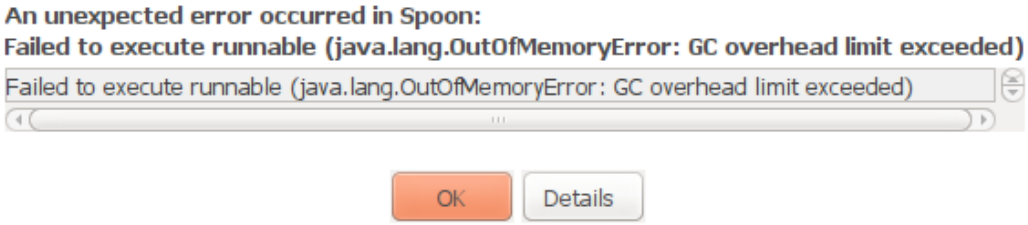
Komponenta na sredini procesa ("Sort Rows") sortira vhodne primere glede na datum. To zahteva naslednja komponenta ("Group by"). Pri tej sem nastavil agregacijo po diskih, kar pomeni, da se za vsak disk poleg njegovih osnovih informacij pridobi le zadnje vrednosti S.M.A.R.T. (vrednost na dan ko disk odpove oz. zadnja znana vrednost).

Agregirane podatke sem preusmeril v končni objekt "Table output". Prek čarovnika sem najprej nastavil povezavo do PostgreSQL podatkovne baze. Pentaho ne zna sam kreirati sheme podatkovne tabele na podlagi podatkov, kot jo za CSV zna csvkit, ampak jo moramo predhodno ustvariti ročno. Zna pa predlagati SQL stavek za kreiranje tabele. Na primeru Backblaze ni pravilno ugotovil vseh tipov atributov. Za S.M.A.R.T. atribute je predlagal tip *timestamp* namesto *bigint*, za datum *timestamp* namesto *date*, za atribut *failure* pa *bigint* namesto *boolean*. Poleg tega je za tip *varchar* predlagal prekratko vrednost, zaradi česar se

[4] http://datacleaner.org

je prvotni poskus uvoza podatkov ponesrečil. Z ročnim popravljanjem SQL stavka za kreiranje tabele sem pomanjkljivost odpravil.

Po zagonu se transformacije niso izvedle do konca, saj je – verjetno zaradi sortirne komponente – prišlo do pomanjkanja pomnilnika in uvoz se je ustavil, kar je prikazano na sliki 8.2. Med delovanjem sta imela oba procesorja utilizacijo blizu 100%. Sicer pa je Pentaho od začetka izvajanja procesa do prekinitve (po 32 minutah izvajanja) prebral približno 1 GB podatkov.



Slika 8.2: Prekinitev uvoza podatkov v Pentaho zaradi omejitve pomnilnika.

8.6 Povzetek

V tem poglavju smo na primeru Backblaze prikazali pripravo podatkov za podatkovno rudarjenje. Žal priprava ni bila popolna. Tako pri PostgreSQL kot pri Pentaho Data Integration sem pogrešal možnost diskretizacije. Pri Pentaho sem pogrešal tudi možnost iskanja neuporabnih atributov. Pri PostgreSQL so se transformacije uspešno izvedle, pri Pentahu pa je zaradi sortirnega koraka prišlo do pomanjkanja pomnilnika. Pri uspešni transformaciji s PostgreSQL smo zmanjšali število atributov iz 85 na 35, število zapisov pa iz 17636994 na 49111.

Poglavje 9

Zaključek

Razumevanje in priprava podatkov za podatkovno rudarjenje zahteva v celotnem procesu CRISP-DM največ vloženega časa in truda. Delo nam pri tem olajšajo različne SUPB in ETL orodja. Pogosto potrebujemo pri pripravi podatkov več orodij, še posebej, če jih moramo najprej iz ene vrste tekstovne (npr. ARFF) ali binarne oblike (npr. HDF) pretvoriti v drugo tekstovno ali binarno obliko.

9.1 Vloga tekstovnih datotek in SUPB v procesu priprave podatkov

Priprava podatkov se začne z njihovim zbiranjem. Ker jih pogosto dobimo v eni ali več tekstovnih datotekah, ki jih je enostavno prenašati med računalniki, smo v začetnem delu naloge naredili pregled nad najpogostejšimi tipi tekstovnih in binarnih formatov. Tekstovni formati so tudi pri večjih količinah podatkov še vedno primerni za hranjenje in izmenjavo in so zato še vedno v zelo široki uporabi. Verjetno se bodo zaradi svoje enostavnosti v ta namen tudi še dolgo obdržali. Tudi testni podatki v tej nalogi so bili v CSV obliki, kljub temu da so bili veliki skoraj 4 GB.

Po drugi strani imamo na voljo veliko vrst različnih SUPB (relacijskih in nerelacijskih), ki so primernejše za hranjenje in organizacijo večjih količin podatkov. Zaradi vseh funkcionalnosti in boljšega upravljanja z resursi – prek mnogo pa-

rametrov lahko določimo, koliko pomnilnika lahko porabijo ipd. – so primerne tudi za vsako nadaljnjo manipulacijo s podatki. Sortiranje po atributih, izbira atributov, uporaba vgrajenih ali lastnih agregacijskih funkcij, okenske funkcije in indeksiranje so le nekatere izmed funkcionalnosti, ki jih v tekstovnih datotekah ne moremo uporabljati. To še posebej velja za relacijske SUPB. Nekatere nerelacijske SUPB so zelo primerne za hranjenje in pridobivanje podatkov, manj pa za izvajanje transformacij.

9.2 Vloga ETL orodij

Med tekstovnimi datotekami in SUPB je še eno področje orodij za pripravo podatkov. To so ETL orodja. Ponavadi jih uporabimo za pridobivanje podatkov iz podatkovnih virov, izvajanje transformacij in shranjevanje podatkov v končni podatkovni vir. Ta orodja so z vidika porabe pomnilnika, procesorja in vhodno-izhodnih enot manj dodelana kot nekatere robustne SUPB. Pentaho Data Integration je tako pestro in uporabniku prijazno orodje, pri izvajanju slabo načrtovanih transformacij pa lahko postane neodziven ali pa se transformacija prekine. Nekateri njegovi gradniki izvajajo opravila v pomnilniku in so zato za velike podatke neprimerni.

9.3 ETL ali ELT

Pogosto poleg izraza ETL srečamo še izraz ELT (an. Extract, Load, Transform), kar pomeni, da namesto, da bi izvorne podatke transformirali na njihovi poti, jih najprej prenesemo v njihov ponor (ponavadi SUPB) in transformacije izvajamo tam. Kot je v nalogi prikazano na primeru dveh SUPB, PostgreSQL in Microsoft SQL Server, so v mnogo primerih tudi SUPB odlično orodje za izvajanje transformacij. Seveda pri tem vsa SUPB niso enako dobra. PostgreSQL ima, poleg velikega števila vgrajenih funkcij in razširitev, možnost enostavnega dodajanja lastnih in agregacijskih funkcij v več jezikih, kot smo pokazali na primeru iskanja najvišjih korelacij med atributi in izstopajočih vrednosti (an. outliers). Tudi dva na videz enaka stavka se lahko v dveh različnih SUPB izvedeta drugače, kot smo prikazali na primeru, kjer je bil v WHERE delu stavka funkcija *random*.

Vprašanje, ali podatke naložiti v SUPB in tam izvajati transformacijo (ELT), ali pa podatke transformirati že na njihovi poti (ETL), ni trivialno. Vsak način ima svoje prednosti in slabosti. Na podlagi primerov in podane teorije bi lahko podal naslednja priporočila.

ETL način je primernejši:

- ko nočemo po nepotrebnem tratiti prostora na disku, saj se nepotrebni podatki odstranijo med izvajanjem transformacij,
- ko nam čas izvajanja transformacij ni na prvem mestu, vseeno pa se mora izvesti v sprejemljivem času,
- ko želimo vse transformacije najprej načrtovati in jih izvesti kot celoto.

ELT način pa je primernejši:

- ko nam je pomembna hitrost izvajanja in je uporabljen SUPB dovolj funkcionalen,
- ko nas ne moti dodatna poraba prostora (podatke najprej v celoti prenesemo v SUPB),
- ko si želimo transformacije izvajati po korakih in med koraki odkrivati dodatne značilnosti v podatkih.

V nekaterih primerih bi bila smiselna tudi kombinacija obeh načinov.

9.4 Orodja glede na kriterije ocenjevanja

Pri izbiri primernega orodja moramo upoštevati več kriterijev. Če bomo transformacije izvajali na enem samem računalniku, se izbira začne že pri finančnem kriteriju – verjetno ne bomo zapravili za orodje nekajkrat več kot za računalnik, sicer bi raje razmislili o nakupu več računalnikov in vzporednem procesiranju. Pomemben kriterij pri izbiri je tudi maksimalno število atributov, s katerim so na žalost pogosto omejene relacijske SUPB. 1000 atributov v svetu podatkovnega rudarjenja ni nič nenavadnega, še posebej če vemo, da gre ponavadi za denormalizirane podatke. Pri kriteriju enostavnosti uporabe ima pogosto prednost dober

grafični vmesnik, vendar so lahko včasih pri zahtevnejših operacijah prednost tudi tekstovni vmesniki. Pentaho Data Integration oz. orodje Spoon ima lepo oblikovan vmesnik, a se med izvajanjem transformacij zatikal. Takšnega občutka pa ni bilo pri uporabi tekstovnega orodja *psql*, s pomočjo katerega lahko upravljamo PostgreSQL. Pri izbiri orodja z vidika enostavnosti uporabe bi lahko omenili tudi dobro dokumentacijo. Nič nam ne pomaga, če ima orodje funkcionalnost, ki jo iščemo, a je nikakor ne moremo najti, ali pa sploh ni dokumentirana.

Pogosta pomanjkljivost ETL orodij in SUPB pri uvažanju podatkov iz zunanjih virov je njihova nezmožnost avtomatskega kreiranja ciljne podatkovne tabele. Zamislite si, koliko klikanja ali pisanja SQL kode bi bilo potrebno pri uvažanju podatkov s 1000 atributi, samo zato, da bi kreirali primerno strukturo tabele v SUPB. Pri tem me je pozitivno presenetilo orodje **csvkit**, ki je z vidika podprtosti tekstovnim datotekam zelo ozko usmerjeno, a svoje delo opravi zelo dobro. Ciljno podatkovno tabelo kreira glede na vsebino v podatkih in nam s tem prihrani precej časa.

V svetu odprtokodnih in brezplačnih SUPB pogrešamo takšne SUPB, ki bi bile narejene prav za namen podpore podatkovnemu rudarjenju. Nekateri produkti so obetavni, a so še vedno v zgodnji fazi razvoja (npr. BayesDB). Upajmo, da se razvijejo v zrela orodja.

9.5 Splošni zaključki

V povprečnem procesu priprave podatkov potrebujemo le manjši del opisanih CRISP-DM opravil, odvisno od vrste podatkov in zahtevanih transformacij. Pogosto podatke pridobimo iz tekstovnih ali binarnih datotek. Za pripravo podatkov so nekatere relacijske SUPB zelo primerna orodja, manjka pa jim nekaj funkcij za pripravo podatkov, kot je mediana in drugi kvantili, odstranjevanje izstopajočih vrednosti, diskretizacija ipd. Takšne funkcije bi lahko z malo truda tudi sami implementirali. Glede nadzora porabe resursov so robustni relacijski SUPB bolj izpopolnjeni kot ETL orodja, ki sem jih preizkusil. Grafična ETL orodja so navidez enostavna za uporabo, vendar vseeno zahtevajo nekaj znanja, pri načrtovanju poizvedb pa moramo proces pogosto popravljati na podlagi napak.

9.6 Možnosti za nadaljne delo

Ker je področje precej široko, je možnosti za nadaljnje delo precej. Težava celovitega pregleda nad orodji je v tem, da zahteva vsako orodje veliko učenja preden lahko o njemu sodimo. Zraven sodi tudi veliko preučevanja dokumentacije in iskanje možnih rešitev na praktičnih primerih. Če funkcije ne najdemo v dokumentaciji, to še ne pomeni, da ne obstaja. Zato bi bila ena od možnosti za nadaljnje delo primerjava orodij, ki v tem delu niso bila omenjena. Smiselno bi bilo tudi bolje preučiti smer razvoja orodij za podatkovno rudarjenje z vidika priprave in uporabe podatkov, saj se v nekaterih primerih zdi, da ta orodja konvergirajo. Pentaho vključuje podporo za Weka komponente, Knime ponuja veliko gradnikov, ki so sicer domena ETL orodij, Orange pa naj bi tesno sodeloval s SUPB. Po drugi strani pa bi lahko tudi utemeljevali smisel "šibke sklopljenosti" med temi orodji. Dodatne raziskave so potrebne tudi na področju povezanosti orodij za podatkovno rudarjenje in SUPB orodji. Katere SUPB uporabiti in na kakšen način izvajati pripravo podatkov? Ali bi bil smiseln razvoj namenske SUPB, ali bi bilo bolje dopolniti obstoječe SUPB. Verjamem, da daje vsebina te naloge bralcu nove ideje za raziskovanje.

Dodatek A

Priloge

A.1 Omejitve št. atributov pri različnih SUBP

SUPB	Omejitev	Opombe
Firebird	16384 za tip INTEGER (4 bajti)	Odvisno od tipa podatkov. Največja velikost vrstice je 65536 bajtov.
MariaDB	(enako kot MySQL)	
Microsoft SQL server	1024	30000 pri redko posejanih vrednostih (max. 8019 bajtov za vrstico)
MonetDB	Neomejeno	Vsak stolpec je v svoji datoteki.
MySQL community server 5.6	4096, InnoDB 1000	max. velikost vrstice 65535 bajtov, pri InnoDB 8000
Oracle 12c	1000	
PostgreSQL	1600	
SQLite	32767	Nastavi se pri prevajanju s parametrom SQLITE_MAX_COLUMN, privzeto je omejitev 2000 stolpcev. Pri večjem št. stolpcev je hitrost branja/pisanja vprašljiva.
Sybase ASE	1024	

A.2 PL/PgSQL skripta za preverjanje omejitve števila stolpcev

Funkcija testira omejitev števila stolpcev glede na podan tip. Ob prekoračitvi se prikaže številka atributa, kjer se je to zgodilo, sicer se kreira shema tabele s takšnim številom atributov. Prvi atribut je za identifikator.

```
/**
 * Testiraj najvecje mozno st. attributov
 * Primer klica: SELECT pg_max_attrs(2000, 'text');
 */
CREATE OR REPLACE FUNCTION
    pg_max_attrs(max integer, testtype varchar)
    RETURNS integer AS $$
DECLARE
    i integer := 1;
BEGIN
    -- Kreiraj novo testno tabelo
    DROP TABLE IF EXISTS test_table;
    CREATE TABLE test_table (id SERIAL PRIMARY KEY);

    -- Vstavljaj stolpce dokler lahko
    WHILE i <= max LOOP
        EXECUTE ('
          ALTER TABLE test_table
          ADD COLUMN attr' || i || ' ' || testtype
        );
        i := i + 1;
    END LOOP;

    RETURN max;
EXCEPTION
    -- Vrni stevilko, kjer je prislo do izjeme
    WHEN too_many_columns THEN RETURN i;
END;
$$ LANGUAGE plpgsql;
```

A.3 PL/PgSQL skripta za izračun korelacij med vsemi atributi

Spodnja funkcija uporablja PostgreSQL funkcijo *corr* za izračun korelacije med vsemi kombinacijami atributov. Kot parameter ji podamo ime podatkovne tabele.

```
-- Ustvari nov podatkovni tip
CREATE TYPE corr_type AS (
    attribute1 varchar, attribute2 varchar, correlation float
);

/**
 * Izracunaj korelacijo med vsemi kombinacijami atributov
 * in vrni kot tabelo. Primer poizvedbe:
 *   SELECT * FROM maxcorr('fruitfly')
 *   ORDER BY correlation DESC LIMIT 10;
 */
CREATE OR REPLACE FUNCTION maxcorr(_table varchar)
    RETURNS SETOF corr_type AS $$
DECLARE
    out varchar := '';
    cr float;
    X information_schema.columns.column_name%TYPE;
    Y information_schema.columns.column_name%TYPE;
    f corr_type%ROWTYPE;
BEGIN
    -- Za vsak atribut
    FOR X IN (
        SELECT column_name
        FROM information_schema.columns
        WHERE table_schema=current_schema()
            AND table_name=_table
        ORDER BY column_name ASC
    )
    LOOP
        -- Za vsak preostali atribut
        FOR Y IN (
            SELECT column_name
```

```
            FROM information_schema.columns
            WHERE table_schema=current_schema()
                AND table_name=_table
                -- Izracun v obe smeri ni potreben
                AND column_name > X
            ORDER BY column_name ASC
        )
        LOOP
            -- Izracunaj korelacijo med atributoma
            EXECUTE 'SELECT corr("' || X || '", "' || Y ||
                '") FROM ' || _table INTO cr;
            f.attribute1 = X;
            f.attribute2 = Y;
            f.correlation = cr;
            RETURN NEXT f;
        END LOOP;
    END LOOP;

    RETURN;
END;
$$ LANGUAGE plpgsql;
```

A.4 PL/PgSQL agregacijska funkcija za izračun mej izstopajočih vrednosti po metodi interkvartilnih razmikov

V spodnji kodi je agregacijska funkcija, ki na podlagi vrednosti atributa izračuna mejne vrednosti za izstopajoče vrednosti po metodi interkvartilnih razmikov. Ker funkcija nima vključenega sortiranja jo kličemo z uporabo ORDER BY, npr:

```
SELECT outliers("THORAX" ORDER BY "THORAX") FROM fruitfly;
```

```
/**
 * Pridobi vrednost srednjega elementa v polju.
 * Pri sodem št. elementov vrni povprečje sosednjih el.
 */
CREATE OR REPLACE FUNCTION arr_get_mid(arr anyarray)
    RETURNS float AS $$
DECLARE
    len integer; halflen integer;
BEGIN
    len := array_length(arr, 1);
    halflen := floor(len / 2);

    IF len % 2 THEN --liho
        RETURN (arr[halflen + 1])::float;
    ELSE -- sodo; vrni povprecje
        RETURN (arr[halflen] + arr[halflen + 1])::float / 2;
    END IF;
END;
$$ LANGUAGE plpgsql;

/**
 * Izracunaj meje za izstopajoce vrednosti (an. outlier)
 * z interkvartilnim razmikom
 */
CREATE OR REPLACE FUNCTION get_outlier_boundaries(arr anyarray)
    RETURNS float[] AS $$
DECLARE
```

```
    len integer; halflen integer;
    Q1 float; Q3 float; IQR float; L float; H float;
BEGIN
    len := array_length(arr, 1);
    halflen := floor(len / 2);

    -- Izracunaj prvi in tretji kvartil
    Q1 := arr_get_mid(arr[1:halflen]);
    Q3 := arr_get_mid(arr[len - halflen + 1:len]);

    -- Interkvartilni razmik
    IQR := Q3 - Q1;

    -- Izracunaj zgornjo in spodnjo mejo
    L := Q1 - 1.5 * IQR;
    H := Q3 + 1.5 * IQR;

    RETURN ARRAY[L, H];
END;
$$ LANGUAGE plpgsql;

/**
 * Agregacijska funkcija
 */
DROP AGGREGATE IF EXISTS outliers(anyelement);
CREATE AGGREGATE outliers(anyelement)
(
    SFUNC = array_append,          -- pri vsaki vrstici
    STYPE = anyarray,              -- tip za stanje
    FINALFUNC = get_outlier_boundaries, -- koncni izracun
    INITCOND = '{}'                -- zacetno stanje
)
```

Stvarno kazalo

Literatura

[1] E. Dumbill, "Getting Up to Speed with Big Data; What Is Big Data?" v *Big Data Now* (Current Perspectives from O'Reilly Media), izd. 2012, Sebastopol: O'Reilly Media, 2012, poglavje 2, podpoglavje 1., str. 3.

[2] P. Katsen. (2013, 22. Nov.). *An Ode to Little Data* [Spletni vir]. Dostopno na: http://katsenblog.com/post/67763100374/an-ode-to-little-data

[3] P.Vagata, K. Wilfong. (2014, 10. Apr.). *Scaling the Facebook data warehouse to 300 PB* [Spletni vir]. Dostopno na: https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/

[4] W. Zhou (2015, 8. Feb.). *Moving STATS for Large Database during Exadata Migration* [Spletni vir]. Dostopno na: https://weidongzhou.wordpress.com/2015/02/08/moving-stats-for-large-database-during-exadata-migration/

[5] P. L. Barreiro, J. P. Albandoz, "Population and sample. Sampling techniques", Sevilla, Španija, 2001, str. 4

[6] S. Džeroski, "Towards a General Framework for Data Mining," v *KDID'06 Proceedings of the 5th international conference on Knowledge discovery in inductive databases*, Berlin, Nemčija, Springer, 2006.

[7] C. Clifton, B. Thuraisingham, "Emerging standards for data mining" v *Computer Standards & Interfaces*, 23. izd., Bedford: Elsevier, 2001, str. 187–193.

[8] I. H. Witten, E. Frank, "Preparing the input," v *Data Mining* (Practical Machine Learning Tools and Techniques), 2. izd. San Francisco: Elsevier, 2005, pog. 2.4, str. 52–60.

[9] P. Chapman et al., *CRISP-DM 1.0* (Step-by-step data mining guide), SPSS Inc., 2000.

[10] KDnuggets. (2007, Aug.). *Data Mining Methodology (Aug 2007)* [Spletni vir]. Dostopno na: http://www.kdnuggets.com/polls/2007/data_mining_methodology.htm

[11] A. Azevedo, M. F. Santos, "KDD, SEMMA AND CRISP-DM: A PARALLEL OVERVIEW," v *Proceedings of the IADIS European Conference on Data Mining*, Amsterdam, Nizozemska: IADIS, 2008.

[12] U. Fayyad, G. P.-Shapiro in P. Smyth,"From Data Mining to Knowledge Discovery in Databases" v *AI Magazine*, 17(3):37-54, jesen 1996

[13] H. Blockeel et al., "Mining Views: Database Views for Data Mining" v *Data Engineering, ICDE 2008, IEEE 24th International Conference*, 2008, str. 1608–1611

[14] Open Knowledge Foundation, "An Overview of File Formats" v *Open Data Handbook Documentation*, izd. 1.0.0, Open Knowledge Foundation, 2012, pog. 1.7.1, str. 16–18

[15] Biolab (2014, 30. Maj). *Orange and SQL* [Spletni vir]. Dostopno na: http://blog.biolab.si/2014/05/30/orange-and-sql/

[16] A.Nayak et al., "Type of NOSQL Databases and its Comparison with Relational Databases" v *International Journal of Applied Information Systems (IJAIS)*, izd. 5, št. 4, Foundation of Computer Sci ence FCS, 2013

[17] L. Candiller et al., "Mining XML Documents" v *Data Mining Patterns: New Methods and Applications*, New York: Information Science Reference, 2008, pog. IX, str. 198–219

[18] A.Bertrand. (2015, 11. Maj). *SQL Server 2016 : JSON Support* [Spletni vir]. Dostopno na: http://blogs.sqlsentry.com/aaronbertrand/sql-server-2016-json-support/

[19] R.R. Bouckaert, "ARFF," v *WEKA Manual for Version 3-6-2*. Hamilton, Nova Zelandija: University of Waikato, 2010, pog. 9, str. 161–166.

[20] J. Korpela. (2005, 12. Feb.). *Tab Separated Values (TSV): a format for tabular data exchange* [Spletni vir]. Dostopno na: https://www.cs.tut.fi/~jkorpela/TSV.html

[21] *MAT-File Format*, The MathWorks, Inc. (Marec 2014), MATLAB, MAT-File format, izd. 2015a, The MathWorks Inc., Natick, MA, 2015.

[22] J. F. McGowan. (2011, 24. Jan.). *Using Octave, a Free MATLAB Alternative* [Spletni vir]. Dostopno na: http://math-blog.com/2011/01/24/using-octave-a-free-matlab-alternative/

[23] S. Goessner. (2006, 21. Feb.). *JSONPath - XPath for JSON* [Spletni vir]. Dostopno na: http://goessner.net/articles/JsonPath/

[24] M. Eriksson, V. Hallberg, "Comparison between JSON and YAML for data serialization,' ' B.S. Thesis, KTH, Theoretical Computer Science , Stockholm, Swe., 2011

[25] W. L. Martinez et al., "Exploratory Data Analysis with MATLAB", Second ed. Boca Raton: CRC Press, 2011, pog.1, str. 3–7

[26] A. Yal. (2009, 11. Jan.). *Beyond SoundEx - Functions for Fuzzy Searching in MS SQL Server* [Spletni vir]. Dostopno na: http://anastasiosyal.com/POST/2009/01/11/18.ASPX

[27] University of Ljubljana. (2015). *Loading and saving data, Orange's tab-delimited format* [Spletni vir]. Dostopno na: http://orange.biolab.si/docs/latest/reference/rst/Orange.data.formats.html

[28] W. Eckerson, C. White, "Evaluating ETL and Data Integration Platforms". Seattle: TDWI Report Series, 2003, str. 6