Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Optimální metody dataminingu pro zpracování semistrukturovaných medicínských dat

Plzeň 2016 Mario Kamburov

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. Srpna 2016

Mario Kamburov

Poděkování

Chtěl bych poděkovat paní Doc. Dr. Ing. Janě Klečkové za vstřícný postoj, motivaci a cenný čas věnovaný vedení mojí diplomové práci. Dále bych chtěl poděkovat své rodině za trpělivosti a podpoře.

Abstract

The aim of my thesis was to propose a solution and to create a program that would allow correction of medical texts on the basis of a very large and diverse semistructured data from medical reports. The work describes the theoretical possibilities of several datamining algorithms for text classification. There is described the principle of my proposed solution, which uses a database to store the training data. In my thesis I am using fulltext search based on CouchDB and Apache Lucene for the purpose of medical spell check and text correction. The verification was applied to a selected collection of medical data. At the end there are comprehensive statistics of the data processing and comparing the obtained results. Conclusion contains an overall assessment of the work with recommendations for possible future improvements.

Abstrakt

Cílem mé diplomové práce bylo navrhnout řešení a vytvořit program, který by umožňoval korekce lékařských textů na základě velmi rozsáhlých a různorodých semistrukturovaných dat z lékařských zpráv. V práci teoreticky popisuji možnosti zpracování několik dataminingových algoritmů pro klasifikaci textů. Je zde popsán princip mnou navrženého řešení, které využívá databázi pro ukládání trénovacích dat. V práci využívám fulltextové vyhledávání pro účely navrhování oprav zkratek a celkovou korekci lékařských textů, založenou na CouchDB a Apache Lucene. Pro vylepšené hledání oprav medicínského textu jsem používal metody dataminingu. Závěr obsahuje celkové hodnocení úspěšnosti datamining algoritmů a fulltextových databází.

Obsah

1 Úvod **9**

2 Analýza **10**
2.1 Zpracování a příprava dat 12
2.1.1 Manuální příprava dat 13
2.1.2 Automatizovaná příprava dat 14
2.1.3 Pravidla pro tvorbu zkratek 14
2.1.4 Pravidla generování zkratek v latině 15
2.2 Závěr analýzy 16

3 Současný stav **18**
3.1 Popis dat 18
3.2 Klasické zkratky ukončené tečkou 18
3.3 Abreviatúry 19
3.4 Složitější zkratky 20
3.5 Další typy zkratek 21
3.6 Problémy medicínských dat 21
3.6.1 Heterogenita 21
3.6.2 Etnické nebo právní problémy 22

4 Fulltextové zpracování medicínských dat **23**
4.1 Fulltextové vyhledávání 23
4.2 Indexování 23
4.3 Modely zpracování full-textových semistrukturovaných dat . 24
4.3.1 Boolský model 24
4.3.2 Vektorový model 26
4.3.3 Rozšířený boolský model 28

5 Výběr fulltextové databáze **29**
5.1 Fulltextová podpora v MySQL 29
5.2 Fulltextová podpora v PostgreSQL 31
5.3 Fulltextová podpora v MongoDB 34
5.4 Fulltextová podpora v CouchDB 37
5.4.1 Pohledy (views) 37
5.4.2 Dotazování 39
5.5 Pokročilé technologie fulltextového vyhledávání 40
5.5.1 Apache Solr 40

5.5.2 ElasticSearch . . . 40
5.6 Apache Lucene . . . 41
5.7 Testování výkonnosti . . . 43
5.8 Souhrn . . . 46

6 Datamining **48**
6.1 Teorie . . . 48
6.2 Datamining v medicíně . . . 49

7 Výběr algoritmů **50**
7.1 Naivní bayes . . . 50
7.1.1 Teoretické základy . . . 50
7.1.2 Naivní Bayes a klasifikace textů . . . 51
7.1.3 Naivní Bayes Multinomial . . . 51
7.1.4 Optimální vylepšení algoritmu . . . 53
7.2 SMO . . . 54
7.2.1 Vznik . . . 54
7.2.2 Optimalizační problém SVM . . . 55
7.2.3 Algoritmus SMO . . . 56
7.3 J48 . . . 58
7.3.1 Rozdělení C4.5 . . . 58
7.3.2 Příklad fungování algoritmu . . . 59
7.4 IBk . . . 59
7.4.1 Princip . . . 60
7.4.2 Příklady použití . . . 61

8 Implementace řešení **63**
8.1 Volba vývojového prostředí . . . 63
8.2 WEKA API . . . 64
8.2.1 Format vstupních dat . . . 64
8.2.2 Datová část . . . 65
8.3 Rozvržení aplikace . . . 66
8.3.1 Backend . . . 66
8.3.2 Frontend . . . 67
8.3.3 Regulární výrazy a jejích role . . . 68
8.3.4 Architektura . . . 70
8.4 Uložení trénovacích dat . . . 71
8.4.1 Struktura databáze . . . 71

9 Porovnávání dataminingových metod **73**
9.1 Hodnotící kritéria 73
9.2 Další kriteria 75
9.3 Zhodnocení výsledků 76
9.3.1 Jednotkové výsledky 76
9.3.2 Celkové výsledky 78

10 Zhodnocení **80**
10.1 Dosažené výsledky 81
10.2 Úspěšnost jednotlivých algoritmů 82

11 Závěr **84**

Literatura **85**

12 Příloha A **92**

13 Příloha B **100**

1 Úvod

Problematika lékařských informačních systémů je velmi rozšířená a komplexní. Úkolem mé diplomové práce bylo korigovat nebo opravovat medicínské odborné termíny a lékařské texty v českém jazyce do takového tvaru, ve kterém by jim po prvním přečtení porozuměli nejen lékaři specialisté, ale například i ambulantní lékaři, akreditovaní Českou lékařskou komorou, jakož i specialisté z jiných oddělení.

S pokrokem internetu a moderních technologií narůstá počet dokumentů v elektronické podobě a potřeby dolování znalostí z nich. Metody, zabývající se klasifikací dokumentů, souhrnně nazýváme metody pro dolování či dobývání znalostí z dat. Metody kategorizace jsou založeny na principech pravděpodobnosti či rozhodovacích stromech atd.

V rámci této diplomové práce jsem se zabýval návrhem řešení korekce lékařských textů, založené na fulltextovém vyhledávání a datamining algoritmů. V první části práce jsem analyzoval vstupní data a připravoval slovníky pro účely detekce a opravy chyb v textu, na základě tvorby zkratek v českém jazyce a existujících standardizovaných slovníků. Následně jsem porovnával fulltextové možnosti několika open-source databází. Poté jsem implementoval základní korekce medicínských zpráv, využívající možnosti fulltextu. V druhé části své práce jsem se zabýval porovnáváním dataminingových algoritmů, zaměřené na klasifikaci textu pro lepší návrhy oprav či získávání znalosti z kontextu dané vybrané lékařské zkratky.

Výsledkem této práce je aplikace pro detekci chyb a navrhování oprav překlepů či korekci zkratek v medicínských zprávách. Na základě podrobného porovnávání jednotlivých databází a dataminingových algoritmů bylo vybráno několik vhodných a optimálních metod pro rychlou a maximálně bezchybnou korekci medicínských textů. V závěru této práce je vidět porovnávání a zhodnocení dosažených výsledků.

2 Analýza

Zadání vzniklo na popud výzkumné skupiny MRE KIV a Fakultní Nemocnice Plzeň. Cílem zadání je naimplementovat aplikaci pro kontrolu pravopisu lékařských zpráv v doméně medicíny. Vstupem byla množina dat ze specifické sféry medicíny - radiologie. Mým úkolem bylo prozkoumat možnosti korekce textu v rámci dostupných lékařských zpráv.

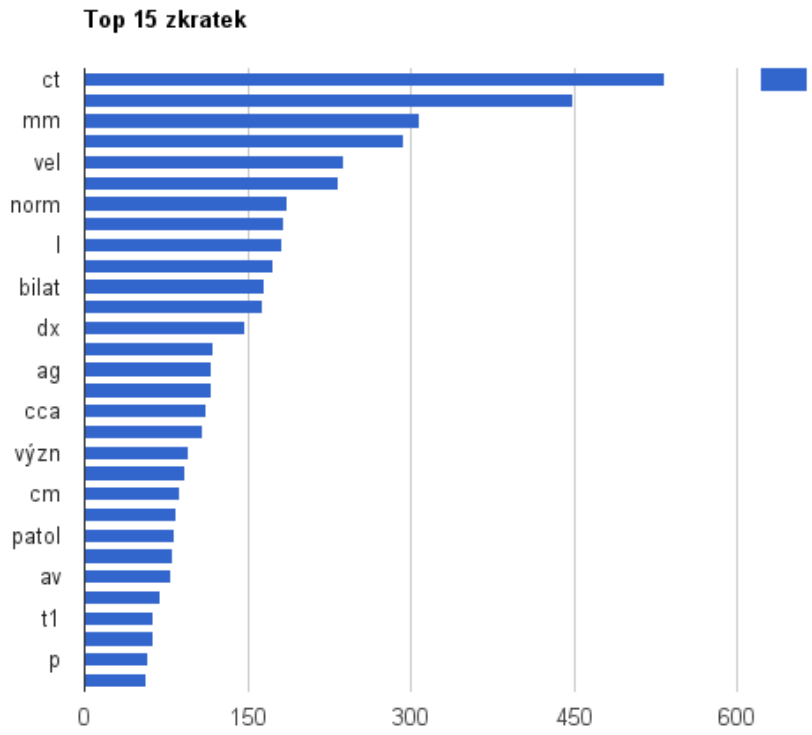
Problematika porozumění medicínských textů, tvořených psaním lékařských zpráv, objektivních a subjektivních anamnéz či dalších odborných medicínských vyjádření se vyskytuje denně v každé klinice či nemocnici. Velmi často porozumění těchto zpráv je zdlouhavá aktivita. Lékaři píší zprávy denně a zaměřuji se na pacienta, nikoliv na psaní dokumentace, tím jsou donucováni texty ve zprávách zkracovat nebo tvořit akronymy. Na druhou stranu lékařské zprávy mohou posloužit jako důležitý artefakt pro budoucí léčbu pacienta a je důležité zprávám plně rozumět. Ve své práci se zabývám porozuměním, opravováním a dolováním znalosti z medicínských zpráv a jejich fulltextovému prohledávání.

Hlavním problémem byla oprava zkrácených lékařských textů a medicínských slov v lékařských zprávách, kdy dané termíny byly uvedeny v neuniverzálních a někdy nepochopitelných tvarech.

Zpočátku jsem uvažoval o manuální korekci zkratek a ručnímu vytvoření slovníku medicínských slov, za pomocí reálného lékaře, ale to bylo nevhodné řešení, od kterého jsem později odstoupil. Důvodem bylo to, že se v lékařských zprávách objevovaly stále stejné zkratky a z toho důvodu jsem nebyl schopný vytvořit dostatečně velkou množinu slov a zkratek, která by posloužila pro účely dataminingu. Rozhodl jsem se odstoupit od původního řešení a soustředil jsem se na tvorbu a kvalitu dat. Dále jsem naprogramoval metodu pro generování zkratek ze slovníků. Nejprve bylo zapotřebí vyhledat existující vydané lékařské slovníky. Dále jsem se zabýval s tvorbou

slovníků v doméně české medicíny a medicínské latiny. Vycházel jsem z veřejně dostupných zdrojů dat. Tímto jsem nejprve vytvořil několik základních slovníků zkratek a slov v medicíně, které mi posloužily pro textovou analýzu a tvorbu generovaných zkratek.

Před samotným zpracováním bylo potřeba porozumět datům a pochopit problém blíže. Za tímto účelem jsem vytvořil histogram v prostředí Excell, obsahující 15 nejčetnějších zkratek, v lékařských zprávách:



Obrázek 2.1: 15 nejčetnějších zkratek v lékařských zprávách

2.1 Zpracování a příprava dat

Důležitou částí diplomové práce je zpracování nestrukturovaných a semistrukturovaných dat. Dále při samotné analýze bylo zjištěno, že poskytnutý dataset je rozmanitý a obsahuje velké množství různorodých zkratek, jak z pohledu lexikální semantiky českého jazyka, tak z pohledu latinského jazyka pro medicínské termíny.

Před samotnou implementací bylo potřeba připravit dostatečně velké množství dat v podobě slovníků, které by mohly posloužit jako stabilní podklad ke korekci lékařských textů. Data byla připravovaná a očišťovaná ručně a částečně automaticky v prostředí MS Excell odkud jsem tvořil CSV soubory vhodné pro import do databází. Data, se kterými jsem pracoval lze rozdělit do několik skupin:

1. **Slovníky pro kontrolu pravopisu** - Slovníky sloužíly pro kontrolu správnosti slov v lékařských textech. Jedná se o standardizované slovníky (Ispell, Aspell) pro kontrolu českého pravopisu ve většina softwarových produktů, pracující s textem. Český Ispell slovník byl ukládaný do InMemory úložiště pro rychlejší čtení.

2. **Český medicínský slovník** - Slovník sloužíl pro vyhledávání slov fulltextovými prostředky a vyhledávání významu určitých zkratek. Český medicínský slovník byl ukládaný do databáze.

3. **Latinsko-český medicínský slovník** - Slovník sloužíl opět k vyhledávání slov fulltextovými prostředky a vyhledávání významu určitých zkratek. Latinsko-český medicínský slovník byl také ukládaný do databáze.

4. **Uživatelský generovaný slovník** - Uživatelský slovník jsem vygeneroval z výše existujících slovníků, pro účely vytvoření značného množství medicínských zkratek a slov k vyhledávání

2.1.1 Manuální příprava dat

Ve fázi přípravy dat, jsem tvořil český slovník medicínských zkratek. Vycházel jsem z několika zdrojů. Hlavním byl "Velký lékařský slovník, 2008"[3], dalšími byly online zdroje Plzeňské Fakultní nemocnice, která poskytuje veřejně zkratky jednotlivých úkonů, které provádí v rámci své činnosti. Dále jsem získával data z dalších osvědčených českých online zdrojů a to především nemocnice v Plzeňském kraji nebo nemocnice jiných krajů ČR. Takto jsem vytvořil 2 slovníky - český medicínský slovník zkratek a česko - latinský slovník. Dopracoval jsem se k 3711 významům jednotlivých českých unikátních zkratek, tak jako 2116 česko-latinských unikátních slov.

Další příprava dat spočívala ve vytvoření standardizovaného slovníku českých a latinských slov za účelem korekce překlepů a chyb v české gramatice. Pro tyto účely jsem vycházel ze standardizovaných open-source slovníků Libre Office pro češtinu, které obsahovaly 166 565 slov pro češtinu a 128 624 pro latinský jazyk. Tyto jsem použil pro rychlé porovnávání neznámých slov v textu. Pomocí rychlé kontroly jsem byl schopný zjistit, zda se jednalo o slovo známé a správné, podle standardizovaných slovníků nebo naopak slovo neznámé, chybné nebo chybějící ve slovníku.

Způsob zpracování probíhal manuálně v prostředí Excell, kde jsem data očišťoval, filtroval a promazával duplicitní zkratky a významy. Dále jsem složité zkratky rozepisoval do menších podzkratek pro lepší import a následné vyhledávání. Používal jsem klasické nativní funkce Excellu pro všechny operace předzpracovávání.

Tato část diplomové práce slouží jako záchytný bod pro další zpracování jakéhokoliv automatizovaného programu.

2.1.2 Automatizovaná příprava dat

V průběhu projektu jsem dospěl k tomu, že i přes poměrně časově náročné manuálním zpracování velkého množství zkratek a medicínských slov je množství dat ve slovnících stále nedostačující. Shledal jsem, že je zapotřebí vytvořit ještě rozsáhlejší slovník. Za tímto účelem jsem vycházel z již manuálně vytvořených slovníků a naprogramoval jsem metody generování zkratek na základě vstupního řetězce.

Program pro automatické generovaní slovníků podporuje generování zkratek, jejichž rozepsaný tvar obsahuje maximálně 5 slov. Toto byl též maximum slov v souvětí, které slovníky obsahovaly. Tím jsem se dopracoval k 12,1 miliónům zkratek a jejich významů.

2.1.3 Pravidla pro tvorbu zkratek

Před automatickým generováním zkratek bylo zapotřebí nastudovat si pravidla pro tvorbu zkratek v českém jazyce. Pravidla pro tvorbu zkratek jsem verifikoval podle Ústavu pro Jazyk Český, Akademie věd ČR.

Pravidla generování zkratek v češtině

Při programování pravidel pro tvorbu zkratek jsem vycházel z učebnic české gramatiky a veřejně dostupných zdrojů. Například článek [4] uvádí, že je možno zkratky rozdělit do několik skupin:

1. **Zkracování na první písmena** - jedná se o způsob zkracování podle prvního písmena slova nebo slov daného spojení. Příklad: m., s.r.o., a. atd.

2. **Zkracování počáteční skupinou písmen** - tento typ zkracování často rozděluje slova na jednotlivé slabiky, pokud možná ukončené souhláskou nebo prostě odebíra samohlásky od slov. Například arteria -> art. nebo mln.

3. **Oficiální zkratky** - jedná se o zkraty titulů, vojenských a policejních hodností či zkratky dané veřejným orgánem státu. V medicíně se může jednat o standardy názvů léků, číselníku nemocí, aberviatúr apod.

4. **Značky** - jedná se o zkratky s ustáleným grafickým obrazem, například písmena z cizích abeced nebo zvláštních nepísmenných grafických znaků. Např. § = paragraf, š = stupeň (např. 16 šC), % = procento (např. 5 %). U těchto typů zkratek se nepíše tečka.

2.1.4 Pravidla generování zkratek v latině

Na rozdíl od českého jazyka, kde pravidla gramatiky pro tvorbu zkratek jsou v platném znění, tak v latině neexistuje gramatické pravidlo, které by stanovovalo jakým způsobem by se zkratky měly tvořit. To mi bylo potvrzeno též profesorem latinského jazyka na Univerzitě Karlově v Praze.

Praktické zkracování latinských slov se používá následovně:

1. Před slovem, ke kterému zkratka patří a upřesňuje význam dalšího slova. Naříklad a.d. 1415 - Anno Domini 1415 - "léta Páně"1415.

2. Za slovem, jehož význam zkratka doplňuje. Příklad: valeriana of. - Valeriana officinalis - Kozlík lékařský (obecný).

3. Zkratky před slovem (většinou jménem) a za slovem se současně používají i u zkratek vysokoškolských hodností. Stejně jako u pravidel českých zkratek.

4. Při tvorbě latinských zkratek platí důležitá zásada, že nová zkratka musí být tvořená takovými písmeny, aby nebylo možné ji zaměnit s jinou zkratkou, která má jiný význam. Každý vědní obor má své specifické zkratky a značky.

5. Chemické značky a zkratky se píší velkým písmenem bez tečky. Příklad: Ag-Argentum (stříbro), Au-Aurum (zlato). Podobně tak i v medicíně Aa. -> arterie (mn.č.).

6. Zkratky v literatuře se píší malým písmenem. Příklad: inf. - infinitiv, nom.-nominativ. Většinou se píší první tři počáteční písmena.

7. Při vytváření zkratek platí zásada, aby nová zkratka byla srozumitelná pro odbornou veřejnost i většinu obyvatel. Počet písmen ve zkratce není předepsán. Mohou to být první 3-4 počáteční písmena zkraceného slova nebo další kombinace 1, 3, 4, 5 nebo 1, 3, 5 písmeno a poslední písmeno zkraceného slova. Zkratka latinského slova většinou vzniká po dohodě s odborníky v daném oboru a po jejím uplatnění v praxi..

2.2 Závěr analýzy

Na základě zmíněných postupů a metod, pro řešení problému korekce medi-cínských zkratek, jsem vytvořil několik základních slovníků pro účely budoucí kontroly překlepů a detekce chyb v lékařských zprávách. Taktéž jsem vytvořil množiny textových dat - český medicínský slovník z veřejně dostupných zkratek a česko-latinský slovník. Tyto jsem potřeboval efektivně prohledávat, proto jsem se rozhodl pro zaměření na fulltextové možnosti několik databází. Nejpodstatnější částí analýzy bylo to, že jsem vytvořil algoritmus pro generování zkratek z předepsaných pravidel české gramatiky. Vedl jsem se hlavně pravidlem 2. Zkracování počáteční skupinou písmen z gramatických pravidel českého jazyka. To říká, že zkratky se tvoří tvorbou jednotlivých písmen slov daného spojení. Tím moje metoda procházela slova od konce a hledala slabiky, které osekávala tak, aby vždy byly ukončené souhláskou. Stejným způsobem jsem generoval zkratky i pro latinská slova z důvodu chybějících předepsaných pravidel. Příklad generování zratek pro slovní spojení "Konečný diastolický tlak levé komory":

```
Konečný diastolický tlak levé komor.
Konečný diastolický tlak levé kom.
Konečný diastolický tlak levé k.
Konečný diastolický tlak levé
Konečný diastolický tlak lev.
Konečný diastolický tlak l.
Konečný diastolický tlak
Konečný diastolický tl.
Konečný diastolický t.
Konečný diastolický
Konečný diastolick.
Konečný diastolic.
Konečný diastol.
Konečný diast.
Konečný dias.
Konečný d.
Konečný
Konečn.
Koneč.
Kon.
K.
```

Pro takto vygenerované zkratky jsem dále procházel slovní spojení ještě jedenkrát, abych vytvořil všechny možné permutace zkratek daného slovního spojení navzájem. Tím jsem se dopracoval k solidní množině zkratek, které jsem používal k vyhledávání textu pro účely korekce překlepů a návrhu na rozepsání lékařských zkratek.

3 Současný stav

V rámci tohoto projektu byla poskytnutá anonymizovaná množina reálných lékařských zpráv, anamnéz a medicínských textů, které byly využity jako podklad k realizaci tohoto projektu. Tato kapitola uvádí jednotlivé případy typů zkratek, které jsem měl za úkol opravit či rozepsat do základního tvaru.

3.1 Popis dat

Vstupní data byla dodána Plzeňskou fakultní nemocnicí, konkrétně rentgenovým oddělením. Jednalo se o anonymizovaná data. Celkem 375 vzorků. Šlo o vzorky zpětně dohledatelné podle URI a čistě textové .csv soubory. Aby se jednalo o kvalitní porovnání různých metod dataminingu, byly použity stejné vzorky na všech algoritmech. [13]

3.2 Klasické zkratky ukončené tečkou

Lékařské termíny potřebující korekci mohou být v různém tvaru. Jde primárně o klasické zkratky ukončené tečkou, například:

1. "Na mozku je patrná hyperdenzita v počátečním úseku a. cerebri media vlevo"
   Kde zkratka “a.“ znamená arteria. V rozepsaném tvaru:
   "Na mozku je patrná hyperdenzita v počátečním úseku arteria cerebri media vlevo"

2. "Alterace perfúzních parametrů v povodí ACM dx. s pouze drobným jádrem v bílé hmotě."⇒ "Alterace perfúzních parametrů v povodí ACM dextra s pouze drobným jádrem v bílé hmotě."

Tyto zkratky bylo nutné ručně dodefinovat ve všech možných podobách, které mohou nabývat. Například zkratka a. by mohla být použita také jako ar., art., arter., artr., Aa., A. apod. Pro efektivnější natrénování modelu byly zkratky tohoto typu definovány a sepisovány ručně s použitím stejného kontextu, za pomocí kterého lze dedukovat též stejný význam. Tím jsem se snažil dosáhnout úplnost modelu slov.

3.3 Abreviatúry

Jedná se též o abreviatury, jako například:

"CTAG:

Odstupy krčních tepen z oblouku aorty jsou volné, v oblasti jugula jsou patrny dislokační změny při zvětšené ŠŽ a uzlovité strumě vycházející z dol. pólu levého laloku, která zasahuje mírně retrosternálně. Oboustranně jsou patrny poměrně masivní kalcifikace v plátech v oblasti větvení ACC, není však patrna významnější stenoza. Intrakraniálně typické uspořádání řečiště s embolem v M1 úseku pravostranné ACM."

Kde:

- CTAG znamená počítačová tomografická angiografie,
- ŠŽ je štítná žláza,
- ACC je arteria carotis communis,
- M1 je pars sfenoidalis,
- ACM je arteria cerebri media

3.4 Složitější zkratky

Další komplikovanější případy jsou neobvyklé zkratky, kombinace velkých a malých písmen, několik teček mezi písmeny nebo v nejhorším případě, když autor lékařské zprávy zapomene přidat tečku ke zkratce, pak lze poměrně složitě naučit algoritmus na danou zkratku. Příklady:

1. "Aa. vertebrales volné. ⇒ Arteria vertebrales volné"

2. "vyš. provedeno po apl. KL i. v. dvoufázově
   ⇒ Vyšetření provedeno po aplikaci kontrastních látek intravenozně dvoufázově"

3. "CT mozku nativně: Vyjádřená ischemie levého F, T a P laloku, bez zn. krvácení. Diskr. tlak. změny na F roh levé postr. komory, střed. struktury bez lateralizace. Prosáknutí měkkých poktývek hlavy vpravo TP a v obl. pravé tváře.
   postkontrastně
   CT perfuze:
   Výpadek perfuze s minimální penumbrou FTP vlevo, zachován pruhovitý okrsek perfuze okolo centrálního sulcu vlevo."

Na příkladu 3 vidíme ukázku medicínské zkratky stejného typu ve 2 neuniverzálních podobách. Jednak lze vyjádřit ischemii laloku rozděleně pomocí popisujících písmen F - Frontální, T - Temporální, P - Parietální nebo je to možné vyjádřit přimo zkratkou FTP. Tento příklad je komplikovaný v tom, že ne vždy lze konkrétně klasifikovat danou zkratku v závislosti na jejím použití. Lékař by mohl někdy potřebovat popsat v lékařské zprávě jednu z nich vícekrát a tímto se narušuje možnost naučit algoritmus na přesnou korekci. Nicméně při nalezení sloučené a zkrácené verze FTP klasifikační algoritmus zvládá rozhodování úspěšně.

3.5 Další typy zkratek

Další typy zkratek obsahují například číslice:

1. "Uzávěr ACM dx v úrovni A1/M1. Aplazie P1 sin - plní se cestou zadní komunikanty.bez dalších patol. změn Willisova okruhu."⇒ "Uzávěr arteria cerebri media dextra v úrovni A1/M1. Aplazie P1 sin - plní se cestou zadní komunikanty. Bez dalších patologických změn Willisova okruhu."

2. "MR pánve a horních stehen: nativně a postkontrastně, 3T, sekvence T2 TSE, T2 TIRM, T1 TSE FS, a postkontrastně T1 TSE +FS"

   Tyto zkratky jsou specifické pro rentgenologické oddělení a vyžadují porozumění odborníka z rentgenologického oddělení, který je zkušený a seznámený se zkratkami, týkající se specifických zákroků při provádění rentgenového vyšetření.

3.6 Problémy medicínských dat

Na základě příkladech dat v předchozích kapitolách lze odvodit, že problematika medicínských dat je poměrně komplexní a nejednoznačná. Obecně je možné data rozdělit do dvou dílčích skupin podproblému - heterogenita dat a právní problémy dat

3.6.1 Heterogenita

Základním problémem medicínských dat je nejednotný formát a složení dat při zpracování lékařských zpráv a jejích ukládání do ruznch struktur - relačních či NoSQL databází. Dostupné data byly dodány v .csv formátu po anonymizaci, nicméně ukládání a transformace skutečných neanonymizovaných dat může mít různou formu a podobu. To je hlavním problémem

při jakékoliv další zpracování dat takovéhoto typu. Jako další vlastnost lze zdůraznit problém Big Data[1], neboli flexibilně narůstající objem dat. Známé vlastnosti jsou tzv. 4V:

- volume (objem) Objem dat narůstá exponenciálně.
- velocity (rychlost) Objevují se úlohy vyžadující okamžité zpracování velkého objemu průběžně vznikajících dat. Vhodným příkladem může být zpracování dat produkovaných kamerou.
- variety (různorodost, variabilnost) Kromě obvyklých strukturovaných dat jde o úlohy pro zpracování nestrukturovaných textů, ale i různých typů multimediálních dat.
- veracity (věrohodnost) Nejistá věrohodnost dat v důsledku jejich nekonzistence, neúplnosti, nejasnosti a podobně. Vhodným příkladem mohou být údaje čerpané z komunikace na sociálních sítích.

3.6.2 Etnické nebo právní problémy

Z pohledu vlastnictví lze říct, že je důležité mít na vědomí, že medicínská data se týkají osobních informací jednotlivých pacientů. Jinými slovy se bavíme o privátní a důvěryhodná data registrovaných pacientů. Jedná-li se o privátní data, pak je zapotřebí, s ukládáním a transformací či transportem dat, zacházet velice opatrně a řídit se specifickými zákony a pravidly, stanovené v právních vnitrostátních a mezinárodních pramenech práv, jako například zákony a normy týkající se ochrany osobních údajů pacientů.

[1]Gartner definuje Big Data jako soubory dat, jejichž velikost je mimo schopnosti zachycovat, spravovat a zpracovávat běžně používanými softwarovými prostředky v rozumném čase.

4 Fulltextové zpracování medicínských dat

4.1 Fulltextové vyhledávání

Fulltextové vyhledávání je určitý způsob vyhledávání informací, často pro-cházení velkého množství semistrukturovaných dat v databázích, které jsou obvykle předem předpřipraveny, například indexováním a tokenizací, aby bylo možno nalézt libovolné slovo, řetězec nebo souvětí v nejkratším možném čase.

Při fulltextovém vyhledávání vyhledávací algoritmus zkoumá všechna slova v každém uloženém dokumentu a pokouší se je porovnat se slovy zadanými uživatelem. V dnešní době se jedná o moderní řešení pro rychlé prohledávání, využíváno je ve většině moderních webových aplikací a portálů, umožňující fulltextové vyhledávání.

4.2 Indexování

V případech s velkým množstvím semistrukturovaných dokumentů větších než kapacita vyhledávacího algoritmu, je pro udržení rychlé odezvy důležité rozdělit vyhledávání do dvou úkolů. První je indexování a druhým vyhledávání indexovaných slov. U některých NoSQL databází je známo jako MapReduce funkce. Indexovací fáze prochází text ve všech dokumentech a vytváří seznam klíčových termínů, tzv. index. Ve vyhledávácí fázi, kdy se provádí specifický dotaz, je prohledáván pouze k tomu připravený index pro každý výraz či slovo. To vede k šetrnosti a rychlejší odezvě.

Každá databáze, ať už NoSQL nebo relační má svůj způsob indexování, pomocí tzv. Indexeru, který vytváří záznam v indexu pro každý výraz nebo

slovo, které najde v dokumentu a případně jeho pozici v dokumentu a ID dokumentu. V pokročilejších databázích Indexer umožňuje komplikované předzpracování, zahrnující textovou analýzu pro konkrétní jazykovou gramatiku. Například ignoruje tzv. stop-words, jako jsou spojky a předložky, které jsou bezvýznamné a přináší chybovost konečných výsledků. Dál umožňuje specifické jazykové úpravy, jako lemmazitaci či stemmatizaci pro nalezení kmene slova nebo derivaci pro skloňování a časování slov při hledání odvozených slov.

4.3 Modely zpracování full-textových semistrukturovaných dat

Ve své práci jsem používal několik databází, podporující fulltextové vyhledávání. S tím bylo spojeno to, že každá databáze používá jiný model zpracování textových semistrukturovaných dat.

Pro vyhledávání dokumentů je nutné stanovit, jakým způsobem bude reprezentován dotaz, jak bude reprezentován dokument a jeho index a stanovit pravidla, kdy dokument dotazu vyhoví, případně jak moc je dokument pro dotaz relevantní. Způsoby, na nichž jsou postaveny vyhledávací stroje lze popsat pomocí modelů. Existuje několik modelů, které významně modifikují implementaci vyhledávacích mechanismů. Mezi nejvýznamnějších z nich patří boolský model, vektorový model a rozšířený boolský model vyhledávání. Mezi další modely patří např. pravděpodobnostní model, fuzzy model, neuronový model, latentní sémantický model, min-max model (MMM), Paice model nebo různé modifikace Beyesovských sítí. [5] [6]

4.3.1 Boolský model

Tento model je jeden z nejstarších vůbec (uplatňovaný v letech 1950 - 1960) a v minulosti býval hojně používán v knihovnických informačních

nebo dokumentografických systémech. Na dotaz vrací jako výsledky ty dokumenty, které obsahují slova z dotazu. V základní variantě neumožňuje stanovování relevance, pracuje se zde pouze s hodnotami 0 nebo 1, respektive dokument nevyhovuje dotazu nebo dokument vyhovuje dotazu. Model podporuje následující logické výrazy:

1. term1 **AND** term2 - v dokumentu se vyskytují oba hledané termy.
   Například: "arteria **AND** cerebri"

2. term1 **OR** term2 - v dokumentu se vyskytují alespoň jeden z termů.
   Například: "arteria **OR** media"

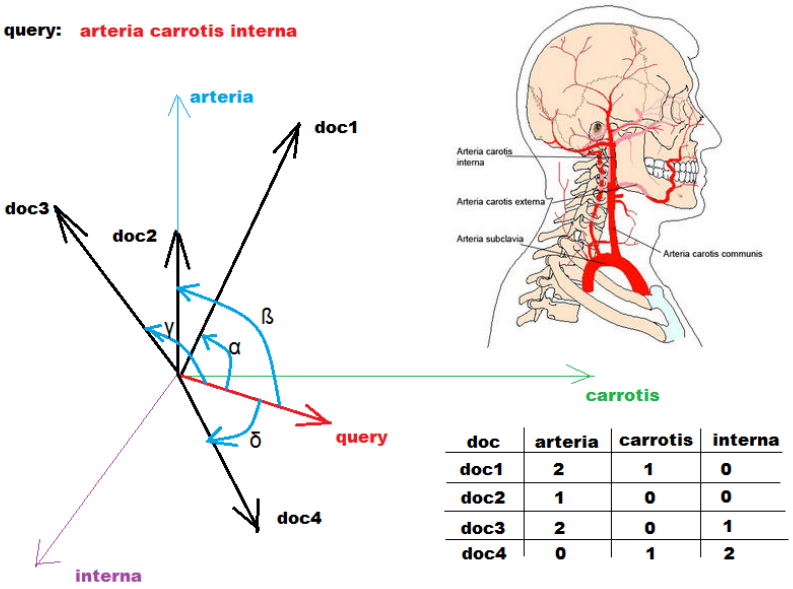
3. term1 **AND NOT** term2 - v dokumentu se vyskytuje term1, ale nikoliv term2.
   Například: "arteria **AND NOT** subclavis"

Boolský model může být obohacen také o další prvky, například o zástupné znaky v termech – "arter*"zastupuje termy "arteria", "arteriální", "arteriálních"atd. Dále podporuje specifikace vzájemné polohy dvou termů v dokumentu ve stanoveném pořadí za sebou nebo blízko sebe. Do fulltextového dotazu lze také zakomponovat omezení dle atributů dokumentu, napřiklad podmínku "AND rok vydání >= 2015".

Výhodou tohoto modelu je jeho jednoduchost, s tím související výkonnost a rychlost, nevýhodou pak potenciální existence takových dotazů, kterým vyhoví je buďto značně malá, nebo naopak rozsáhlá množina dokumentů. To je důsledkem příliš hrubého rozdělení dokumentů, u nichž se rozlišuje pouze zda dotazu vyhoví či nevyhoví. Model nemá nástroje pro uspořádání vyhovujících dokumentů podle míry relevance vůči položenému dotazu. Omezením tohoto modelu je i skutečnost, že všechny termy v dotazu i v identifikaci dokumentu jsou chápány jako stejně důležité.

4.3.2 Vektorový model

Vektorový model je novější než boolský a snaží se odstranit nebo alespoň minimalizovat nevýhody boolského modelu. Umožňuje především jemnější výpočet relevance dokumentu vzhledem k dotazu. Cíle modelu jsou založeny na tom, že každý text (textový záznam, token či dotaz) je reprezentován bodem v n-rozměrném souřadnicovém systému. Tento bod představuje zároveň i vektor začínající v počátku souřadnic.



doc	arteria	carrotis	interna
doc1	2	1	0
doc2	1	0	0
doc3	2	0	1
doc4	0	1	2

Obrázek 4.1: Ukázka tvorby vektorového modelu n-rozměrného prostoru pro n medicínských termů

Vektorový model reprezentuje jak dotaz, tak i dokument jako vektor $(v_i, ..., v_m)$. Číslo m představuje počet všech termů vyskytujících se v sadě dokumentů. Hodnoty $v_i, ..., v_m$ jsou z intervalu <0, 1>. Složka v_i vyjadřuje významnost i-tého termu pro dokument či dotaz. Nulová nebo nule blízká hodnota znamená term nevýznamný nebo málo významný. Složky blízké jedné pak vyjadřují důležité termy.

Vzhledem ke skutečnosti, že dotaz i dokument jsou reprezentovány pomocí vektorů v prostoru <0, 1> m, nabízí se měřit relevanci dokumentu pro

daný dotaz jako podobnost příslušných dvou vektorů.

Základní úvaha vychází ze známého faktu, že skalární součin dvou vektorů je největší, pokud mají tyto vektory stejný směr a jako nulový vychází, pokud mají vektory směr opačný. Tohoto jevu lze využít pro vlastní kalkulaci podobnosti. Funkce, která dotazu a dokumentu přiřadí hodnotu jejich podobnosti, se nazývá podobnostní funkce.

Nechť q je dotaz reprezentovaný vektorem termů, q_j je j-tá složka vektoru dotazu, d_i je i-tý dokument a $w_{i,j}$ je j-tá složka jeho vektoru (tj. významnost j-tého termu v i-tém dokumentu). Podobnostní funkci pak označme jako $sim(q, d_i)$. V případě využití vlastností zmiňovaného skalárního součinu by podobnostní funkce měla následující tvar:

$$sim(q, d_i) = \sum_{j=1}^{m} q_i * w_{i,j} \tag{4.1}$$

Uživatelské dotazy jsou obvykle reprezentovány množinou termů, přičemž po- čet termů je typicky velmi malé číslo. Vektor dotazu tak má téměř všechny složky nulové, pouze pro uvedené termy jsou příslušné složky rovny jedné. Tento přístup lze zobecnit tím, že pro každý term v dotazu bude umožněno uvést jeho váhu. Pro efektivní kalkulaci vah složek vektorů dokumentů v kolekci lze vektory spočítat jako:

$$w_{i,j} = TF_{i,j} * IDF_j \tag{4.2}$$

kde $w_{i,j}$ je opět j-tá složka vektoru i-tého dokumentu. $TF_{i,j}$ přestavuje frekvenci (počet výskytů) j-tého termu v i-tém dokumentu a IDF_j je tzv. inverzní frekvence j-tého termu v sadě dokumentů. Inverzní frekvence lze vyjádřit ze vztahu:

$$IDF_j = log_{\frac{n}{DF_j}} \tag{4.3}$$

Hodnota DFj vyjadřuje počet dokumentů obsahujících j-tý term, n je pak

počet všech dokumentů v kolekci. Inverzní frekvence reprezentuje důležitost termu pro indexaci v rámci celé kolekce dokumentů. Jejím hlavním smyslem je posílit vliv málo používaných termů.

Existuje řada možností, jak vektorový model dále modifikovat či rozšiřovat. Jednou z vlastností skalárního součinu například je, že pro dva dlouhé vektory (přiřazené delším dokumentům) vychází jeho hodnota větší, než pro dva kratší vektory, přestože v obou případech je směr vektorů shodný. Bezdůvodně by tak byly při výpočtu podobnosti zvýhodněny dlouhé vektory (dokumenty) před krátkými. Pro eliminaci tohoto jevu je proto vhodné vektory normalizovat na jednotkovou délku. Dále podobnostní funkce může mít i komplikovanější tvar než prostý skalární součin. Může být využita např. Kosinová míra, Jaccardova míra, Diceova míra a další. [5]

Vektorový model nabízí oproti boolskému modelu jemnější rozlišení míry relevance dokumentů. Ta nabývá hodnoty zpravidla z intervalu <0, 1>. Čím vyšší číslo je dokumentu přiřazeno, tím více je dokument relevantní vůči vyhledávacímu dotazu.

4.3.3 Rozšířený boolský model

Rozšířený boolský model se snaží o skloubení dvou výše popisovaných modelů se zachováním výhod z obou modelů. Z boolského modelu si zachovává možnost specifikace logických vazeb mezi termy v dotazu, z vektorového modelu přidává možnost určení míry relevance dokumentu vzhledem k položenému dotazu. Model jde dále rozvinout, a to zakomponováním tzv. p-normy. V p-normě se namísto druhých mocnin a odmocnin používají jejich ekvivalenty o základu p. Tento parametr p pak musí být nějakým způsobem stanoven před samotnou kalkulací podobnosti. Literatura [5] uvádí, že pro p = 2 model vykazuje lepší výsledky než model vektorový.

5 Výběr fulltextové databáze

Ve své práci jsem porovnával několik databází a jejich fulltextovou podporu vyhledávání pro účely navrhování oprav medicínských textů. Zvolil jsem relační a nerelační databáze. Širší možnosti vyhledávání nabízejí specializované fulltextové knihovny, které si v této kapitole také blíže představíme.

5.1 Fulltextová podpora v MySQL

Jako typického představitele kategorie menších databázových systémů můžeme zvolit populární, open-source databázi MySQL. Ta v rámci svých funkcionalit poskytuje s jistými omezeními i možnost fulltextově vyhledávat v textových datech.

MySQL je v této práci popisován jako ilustrační příklad typického databázového systému, který mimo ukládání dat poskytuje od verze 3.23.23 také fulltextové indexování a vyhledávání slov či slovních spojení přímo v databázi, pomocí fulltextového indexu. Existence omezení do verze 5.5 bylo podmínkou uložení textu do tabulky typu MyISAM, kde databáze umožňovala vytvoření fulltextového indexu nad sloupci s textem. Od verze 5.6+ přibyla podpora tvorby indexu i pro novější systém tabulek InnoDB. Tím se odstranila jedna z nevýhod. Tento typ indexu je dostupný pro pole datových typů text, char a varchar. Vyhledávání poté, pokud uživatel nespecifikuje jinak, probíhá ve všech indexovaných sloupcích.

Podpora fulltextu v MySQL je aktuálně možná třemi hlavními způsoby:

1. **Vyhledávání přirozeným jazykem**, jenž interpretuje vyhledávanou sekvenci jako frázi v běžném jazyce [7]. Nelze použít zástupné znaky ani pomocné booleovské operátory pro tvorbu složitějších dotazů. Ignorují se běžně vyskytující se slova uvedená v seznamu stopwords (jedná se o spojky, částice, zájmena) atd. Standardně zabudovaný seznam

stopwords v MySQL obsahuje pouze anglická slova, český seznam je možno vidět například na odkaze [8]. Příklad syntaxe:

```
SELECT MATCH (lekarska_zprava_plzen)
AGAINST ('intrakarnialni')
FROM lekarske_zpravy;
```

Pokud máme za cíl seřadit výsledky sestupně dle relevance, použijeme operátor MATCH() v klauzuli WHERE. Hodnota relevance se počítá na základě počtu slov v poli, počtu unikátních slov v poli, celkovému počtu slov v prohledávaných polích, počtu dokumentů obsahujících hledané slovo a unikátnosti slova.

```
SELECT * FROM lekarske_zpravy
WHERE MATCH (lekarska_zprava_plzen)
AGAINST ('intrakarnialni');
```

2. **Vyhledávání boolskými operátory**. Umožňuje přidávát do vyhledávání znaky a booleovské operátory se speciálním významem pro přesnější zadání dotazu. Znaky + a - (nebo jejich ekvivalenty AND a NOT) se používají pro slova, která musí být, resp. nesmí být ve výsledcích. Mezera mezi hledanými slovy funguje stejně jako spojka OR značící, že v záznamu se může nacházet pouze jedno z hledaných slov. V případě, že bychom chtěli nalézt slova začínající na určitá písmena, lze využít zástupný znak *, nahrazující 0 až n znaků napravo od znaku. Tím se dá částečně nahradit chybějící skloňování. Tedy pokud chceme nalézt všechny možné tvary vycházející ze slova art, zadáme dotaz ve tvaru:

```
SELECT * FROM lekarske_zpravy_fn_plzen
WHERE MATCH (zpravy)
AGAINST ('art*' IN BOOLEAN MODE);
```

V tomto příkladě databáze vrátí slova arteria, arterie, arteriální, artróza, atd. Obdobný účel splňuje i zástupný symbol „?“, nahrazující pouze jedno písmeno. U dotazu „arteri?“ jsou možné nalezené tvary arterie, arteria, arterii atd. V boolean módu se taktéž vynechávají slova ze seznamu stopwords, ale na rozdíl od prvního způsobu vyhledávání se do hledání zahrnují i slova vyskytující se ve více než polovině záznamů. Prohledávaná pole nemusí mít vytvořený fulltextový index, ale prohledávání pak bude pomalejší

3. **Vyhledávání rozšířeným dotazem**. Používá se zejména v případech, kdy vyhledávaná fráze je příliš krátká na to, aby byla zahrnuta do indexu. [9] Nejdříve proběhne první prohledání, ze kterého se vezme a připojí několik nejrelevantnějších dokumentů do vyhledávaného řetězce a proběhne opětovné hledání.

```
SELECT VYZNAM
FROM vyznam
WHERE MATCH (VYZNAM) AGAINST
('arteria' WITH QUERY EXPANSION);
```

Například uživatel vyhledávájící výraz "databáze"může ve skutečnosti hledat spíše výsledky typu "MySQL", "Oracle", "DB2", a "RDBMS". Jedná se o fráze, které by měly odpovídat dotazu "databáze"a měly by být též vráceny. Nazývají se implikované znalosti.

5.2 Fulltextová podpora v PostgreSQL

Jako dalšího typického představitele kategorie open-source databázových systémů můžeme zvolit databázi PostgreSQL. Ta obsahuje ve svém jádru od verze 8.3 všechny funkce potřebné pro fulltextové vyhledávání. Například implementuje modul **tsearch2**, který se používal v minulosti jako rozšíření,

které bylo nutné nejprvé nainstalovat a nakonfigurovat, pro fulltextové vyhledávání. Tento modul značně rozšířil databázi, SQL příkazy a také vnitřní funkce, pracující s textem. Zároveň umožňuje použití tzv. parserů, které slouží jako tokenizery a filtry v Apache Lucene. Mimo tsearch2 obsahuje PostgreSQL i tzv. contrib moduly. Jedná se o jednoduché funkce (lower - převedení všech písmen na malé, unaccent - odstranění dyakrytiky slov atd.), které pracují s textem a mohou zastávat funkci jednoduchého fulltextového vyhledávače. [10]. Příkladem použití může být:

```
select to_tsvector('arteria cerebri media') @@
to_tsquery('arteria');
----------
t
(1 řádek)
```

Na tomto příkladě je použitý statický řetězec "arteria cerebri media", sloužící jako vstup a pomocí symbolu @@ v PostgreSQL vyhledávám řetězec "arteria". Odpověď je true - úspěch.

```
select to_tsvector('arteria cerebri media') @@
to_tsquery('subclavis');
----------
f
(1 řádek)
```

Na dalším příkladě je podobný příklad pro hledané slovo "subclavis", které není zahrnuto ve vstupním řetězci. Odpověď databáze je negativní - false. Zde si můžeme všimnout, že symbol @@ jednak spouští fulltextové vyhledávání a současně podporuje funkce `to_tsvector(")` a `to_tsquery(")`. Tyto funkce, slouží pro konverzi textových dat do podporovaného datového typu tsvector v PostgreSQL. Funkce `to_tsvector` parsuje textový dokument na jednotlivé tokeny, redukuje tokeny na lexemy a vrácí tsvector, který vypisuje

všechny lexemy spolu se svými pozicemi v dokumentu. Zde je jednoduchý příklad:

```
SELECT to_tsvector('english', 'a fat  cat sat on a mat -
 it ate a fat rats');
to_tsvector
-----------------------------------------------------
 'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

Na příkladě je dobře vidět, že tsvector neobsahuje žádné stop slova jako spojky, částice, dokonce ignoruje interpunkční znaménka. Výsledný vektor obsahuje tokenizované, očištěné, převedené do základních tvarů slova původního vstupního řetězce. Vyhledávání uvnitř řetězců, například pro n-gramy tsearch2 neumí, musí používat další contrub modul s názvem `pg_trgm`. Toto lze zmínit jako nevýhodu a omezení databáze oproti NoSQL konkurentům.

Pro hladké fulltextové vyhledávání databáze jsou zapotřebí tzv. GiST nebo GIN indexy vytvořené nad sloupcem typu `ts_vector`. Tento typ sloupce obsahuje běžně jakékoliv hodnoty, jejichž obsahem tsearch2 pomocí parserů se snadno určuje. Dělá se to z důvodu rychlosti. Jestliže budu vyhledávat nad sloupcem typu `ts_vector`, tak výsledky budou okamžité a mnohem rychlejší v porovnání s klasickými textovými datovými typy. V praxi to funguje tak, že se vytvoří funkce nebo trigger, který aktualizuje data ve sloupcích typu `ts_vector` podle hodnot v klasickém stringovém sloupci. Tím tabulka bude mít 2 sloupce, klasický obsahující textové hodnoty a další obsahující zpracovaný tsv vektor na základě vstupu v původním textovém sloupci.

Podle [11] je GIN index lepší pro statická data, z důvodu rychlejšího vyhledávání. Pro dynamická data je lepší GiST, protože umí rychleji aktualizovat data, ale pouze má-li v indexu méně než 100 000 unikátních slov. Ve svém projektu jsem pracoval s GIN indexem z důvodu velmi častého čtení statických dat z vygenerovaných slovníků.

5.3 Fulltextová podpora v MongoDB

Jako jedním z nejznámějších představitelů NoSQL databáze jsem se rozhodl pro studování možnosti použití fulltextu v MongoDB. Jedná se o NoSQL databázi, která spadá do typů dokumentově-orientovaných databází. Je napsaná v jazyce C++. Umožňuje fragmentaci dat do více distribuovaných uzlů, tak jako replikaci dat metodou master-slave. MongoDB poskytuje tzv. kolekci dat, kde každá kolekce sdružuje data podobného typu, jedná se o databázovou entitu. Každý dokument má generovaný unikátní ID, tzv. klíč, dle kterého je možné JSON dokumenty vyhledávat. Klíč je možné taktéž generovat automaticky nebo specifikovat manuálně. Ukázka práce s mongoDB objekty v praxi:

```
db.medical.find().pretty()
{
"_id" : ObjectId("5790c555c40382db74174734"),
"zkratka" : "a.",
"vyznam" : "arteria"
}
{
"_id" : ObjectId("5790c567c40382db74174735"),
"zkratka" : "art.",
"vyznam" : "arteria"
}
{
"_id" : ObjectId("5790c574c40382db74174736"),
"zkratka" : "ACM",
"vyznam" : "arteria cerebri media"
}
{
```

```
"_id" : ObjectId("5790c583c40382db74174737"),
"zkratka" : "ACI",
"vyznam" : "arteria cerebri interna"
}
{
"_id" : ObjectId("5790c5a5c40382db74174738"),
"zkratka" : "aa.",
"vyznam" : "arterie"
}
```

V MongoDB nad jednotlivými klíči v dokumentu lze následně definovat indexy pro rychlejší vyhledávání a čtení dokumentů. Na úkor toho, že indexy zpomalují zapisování dat. Od verze 3.2 MongoDB podporuje tzv. Text Index k podpoře fulltextového vyhledávaní a dotazování. Textové indexy mohou zahrnovat jakékoli pole, jehož hodnota je řetězec nebo pole řetězcových prvků. Příklad tvorby Text indexu:

```
> db.medical.createIndex({"zkratka":"text","vyznam":"text"})
{
"createdCollectionAutomatically" : false,
"numIndexesBefore" : 1,
"numIndexesAfter" : 2,
"ok" : 1
}
```

Výsledkem takového indexování může být mnohem efektivnější a časově šetrnější operace vyhledávání, například:

```
> db.medical.find({$text: {$search: "cerebri"}},
{score: {$meta: "textScore"}}).sort(
{score:{$meta:"textScore"}
})
```

```
-------------------------------------------------------
{ "_id" : ObjectId("5790c574c40382db74174736"),
"zkratka" : "ACM",
"vyznam" : "arteria cerebri media",
"score" : 0.6666666666666666 }
{ "_id" : ObjectId("5790c583c40382db74174737"),
"zkratka" : "ACI",
"vyznam" : "arteria cerebri interna",
"score" : 0.6666666666666666 }
>
```

Na příkladě je vidět, že používám operátor $text a $search k prohledání všech textových indexů, obsahující term "cerebri". Tímto dotazem databáze vrací všechny dokumenty, obsahující hledaný výraz. To je hlavní přidaná podpora MongoDB pro fulltextové vyhledávání, která je ovšem nedostačující pro korekci medicínských zkratek. Důvodem je omezené vyhledávání celých slov. Pro korekci, opravu, náhradu či dovyplňpvání medicínských zkratek je tato funkcionalita omezující. Problém komplexnějších dotazů a vyhledávání podřetězce v textu je vyřešený použitím regulárních výrazů přímo v dotazu:

```
> db.medical.find({'vyznam': /cereb/})
{ "_id" : ObjectId("5790c574c40382db74174736"),
"zkratka" : "ACM",
"vyznam" : "arteria cerebri media" }
{ "_id" : ObjectId("5790c583c40382db74174737"),
"zkratka" : "ACI",
"vyznam" : "arteria cerebri interna" }
>
```

Dotaz vrací všechny dokumenty obsahující podřetězec "cereb". Tento řetězec je nadefinovaný tzv. vzorem (pattern). Databáze obsahuje další rozšířené

možnosti psaní komplexních regulárních výrazů. Pro řešení problému medicínských zkratek je tato varianta již vhodnější.

5.4 Fulltextová podpora v CouchDB

Jako dalším představitelem poměrně nových typů databází jsem zvolil CouchDB. Je napsaná v jazyce ErLang. Je to open-source databáze s HTTP RESTful JSON API, která klade důraz na možnosti replikace a běhu v distribuovaném prostředí. Distribuovaný systém je takový systém, který dokáže úlohy a dotazy řešit napříč počítačovou síti. CouchDB je databázový systém, který se při běhu v distribuovaném prostředí zaměřuje především na dostupnost, a to při zachování tolerance k rozdělení u tzv. CAP teoremu [1].

Ve své podstatě CouchDB též spadá do skupiny dokumentově - orientovných databází. JSON dokumenty jsou složené z libovolného počtu polí a souborových příloh. Příkladem JSON dokumentu může být:

```
{
   "_id": "63e5c848fa2211c3b063d6feccd49849",
   "_rev": "1-26d52b5096d732c631100927e460643c",
   "DATAWORKS_DOCUMENT_TYPE": "user14169_slovnik_medical",
   "vyznam": "Koincidenční emisní tomografie nádorů",
   "zkratka": "FDG-TU"
}
```

5.4.1 Pohledy (views)

Pohledy jsou primárním nástrojem použitý pro dotazování a získávání metadat o dokumentech CouchDB. Existují dva různé druhy zobrazení: trvalé a dočasné pohledy. Pohledy vyjadřují způsob jak dát nestrukturovaným

[1] CAP - Consistency, Availability, Partition tolerance. Odkaz: `https://dzone.com/articles/better-explaining-cap-theorem`

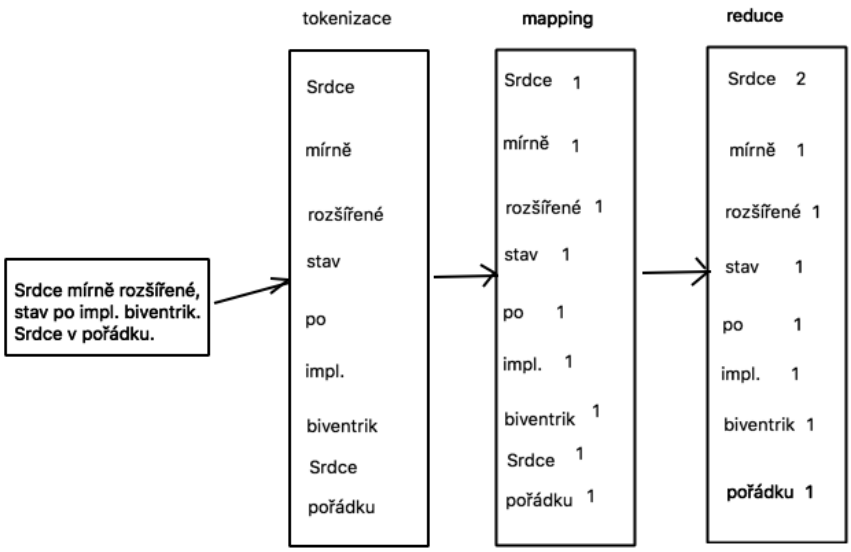
datům uloženým v dokumentech strukturu.

1. Trvalé pohledy (view) jsou uložené uvnitř speciálních dokumentů nazývaných design dokumenty, a mohou být přístupné přes HTTP požadavek.

2. Dočasné pohledy (view) nejsou uložené v databázi, ale provedené na vyžádání. K vykonání dočasného pohledu, můžeme provést požadavek HTTP POST na URI adrese: / dbname / tempview, kde tělo HTTP požadavku obsahuje kód funkce a Content-Type je nastavený na hodnotu application / JSON. Tento typ pohledů se používá pouze pro vývoj.

Kdybychom chtěli získat všechny dokumenty z databáze, funkce map by mohla vypadat následovně:

```
function(doc) {
  emit(doc._id, {"rev" : doc._rev});
}
```



Obrázek 5.1: Ukázka fungování funkce MapReduce v CouchDB

Na příkladě vidíme jednoduchou funkci v Javascriptu, která vypisuje všechny dokumenty podle klíče. Takto jednoduše lze tvořit pohledy a indexovat dokumenty v CouchDB. Těmito funkcemi, lze data v nestrukturované či semistrukturované formě reprezentovat různými způsoby, grupovat, agregovat, seskupovat atd.

Pohledy se udržují dynamicky, a nemají vliv na podobu dat uložených v databázi. V určitých aspektech se tyto pohledy podobají materializovaným pohledům v SŘBD Oracle. Samotné pohledy jsou izolované JSON dokumenty, které nemají vliv na replikaci či fragmentaci původních dat. Celkový proces Map-Reduce je definován funkcí Map, kterou jsme již popsali a následně další funkcí pro Reduce, která má spíše agregační, sumarizační a grupovací účely. Dokumenty jsou z pohledu vybírány pomocí indexu, rozsahu indexů, nebo jako celek.

5.4.2 Dotazování

Dotazování je možné přímo v browseru prohlížeče, po úspěšném nakonfigurovaní a rozběhání databáze. CouchDB nabízí jednoduché UI, pomocí kterého lze jednoduše prohlížet a obstarávat databázi dat. Dále je dotazování možno dvěmi hlavními způsoby. První je HTTP REST API, přes REST Console, CURL požadavek nebo v prohlížeči, například:

```
GET /db/_design/medical/_view/by_medical_term?key="vertebralis"
```

Dalším způsobem je pomocí API rozhraní, které podporuje řadu programovacích jazyků. Podporuje taktéž jazyk Java, který používám ve své práci. Nevýhoda tohoto způsobu je stejná jako u MongoDB, kde se v databázi vyhledávají pouze celá slova a termy. To není dostačující pro problém medicínských zkratek a opravu lékařských zpráv. CouchDB řeší tento problém stejně jako jeho konkurent MongoDB - použitím regulárních výrazů (v Javascriptu se jedná o objekt RegExp) v Map funkci.

Jako výhodou CouchDB lze upřednostnit jeho rozšířenou a prosperující komunitu uživatelů, dobrou dokumentaci a REST API. Ve své práci jsem se setkal s dobrou integrací CouchDB s vyhledávácím enginem Apache Lucene. To považuji za výhodu, ale pořád to mluví o nevyspělosti této NoSQL databáze v možnostech fulltextového vyhledávání.

5.5 Pokročilé technologie fulltextového vyhledávání

5.5.1 Apache Solr

Solr je platforma pro vyhledávání v textu, včetně fasetového vyhledávání, distribuovaného vyhledávání a vyhledávání v dokumentech typu PDF nebo ODT. Jedná se o svobodný software dostupný pod licencí Apache License, který je napsaný v Javě a vyvíjený v rámci projektu Lucene nadace Apache Software Foundation. Pro komunikaci se Solr se používá REST API, Solr podporuje ukládání dat ve formátech JSON, XML, CSV, PDF atd.

5.5.2 ElasticSearch

Elasticsearch je fulltextový vyhledávač vycházející z Apache Lucene. Pro komunikaci se používá opět RESTové rozhraní, které nabízí vysokou dostupnost, rychlost a škálovatelnost. Je vyvíjený v Javě a komunikovat s ním lze pomocí webového rozhraní. Je šířen zdarma pod licencí Apache. Jak již bylo zmíněno Solr a Elasticsearch využívájí k vyhledávání Apache Lucene, což je patrně nejvýkonnější fulltextové vyhledávání dostupné v rámci open source produktů, které jsou momentálně na trhu. Vyhledávání nabízí podporu více jazyků, vyhledávání na základě geografické polohy, vyhledávání podobných nebo příbuzných slov. Lze jej také využít k inteligentnímu automatickému doplňování formulářů na webu, pomocí Apache Lucene.

5.6 Apache Lucene

Jak již bylo mnohokrát zmíněno Apache Lucene je open-source projekt vyvíjený v Javě, pod volně šiřitelnou licenci Apache Software Foundation, který slouží pro indexaci a vyhledávání textových dat. Je velmi rychlý, protože je založený na vektorovém modelu a využívá optimální algoritmy pro hledání podobnosti dokumentů v NoSQL databázích. K organizaci dat používá invertovaný index. Lucene samotný obsahuje nízkoúrovňové příkazy pro ukládání a získávání dat. K tomu poskytuje komplexní API. Samotný projekt Lucene neřeší škálování, distribuovaný přístup a další užitečné vlastnosti pro realtime webové aplikace. Na obrázku 5.1 je vidět, jak funguje v praxi invertovaný index. Do takovéhoto typu indexu lze dále ukládat informace o pozici slova v daném dokumentu či jiných metainformací. Dalším důležitým aspektem pro datamining je ukládání těchto slov v lemmatizovaném tvaru. Má to pak přesnější morfologický význam a pozitivní dopad při fulltextovém vyhledávání.

Apache Lucene zakládá svoje vyhledávácí schopnosti na algoritmu kosinová podobnost (cosine similarity). Jedná se o míru podobnosti dvou vektorů, která se získá výpočtem kosinu úhlu těchto vektorů. Toho se dá využít pro zjištění podobnosti dvou dokumentů. V takovém případě budou vektory reprezentovat četnost jednotlivých slov. [12]. Ukázka tvorby vektorového modelu je vidět na obrázku 4.1.

Možnosti dotazování přes Apache Lucene:

1. Vyhledávání termů - Lucene podporuje vyhledávání slov, termů, souvětí atd. Příklad:

   ```
   vyznam: "Arteria Cerebri Interna"
   ```

2. Fulltextové prohledávání polí - Lucene podporuje prohledávání a dotazování jednotlivých polí v JSON dokumentech. Zde je možnost uplatnit

též boolské operátory Příklad:

```
vyznam: "Arteria Cerebri Media" AND zkratka:"cereb."
```

3. Zástupné vyhledávání - Lucene podporuje jednorázové nebo opakované zástupné vyhledávání (wildcard search) v rámci jednotlivých pojmů (ne v rámci celého výrazu dotazu). Chceme-li provést zástupné hledání jednoho znaku používáme symbol "?". Chceme-li provést několikanásobné hledání zástupných znaků používáme symbol "*". To, co řeší velké procento problémů u medicínských zkratek je právě vícenásobné zástupné vyhledávání. Příklad:

   ```
   http://127.0.0.1:5984/db/_design/med_lat/by_latin?q=art*
   ```

   Tento dotaz ?q=art* vícenásobně prohledává zástupné znaky a automaticky dovyplňuje vstupní řetězec. Tím způsobem jsme schopni vzít velké množství návrhu z databáze, seřadit je a navrhnout uživateli jako možnou korekci textu. Výsledkem dotazu může vypadat následovně:

   ```
   arteria, 0.982
   arteriální, 0.981
   arterii, 0.980
   artikulace, 0.521
   ...
   ```

4. Fuzzy prohledávání - Lucene podporuje fuzzy prohledávání založené na Levensteinové vzdálenosti. Defaultní vzdálenost slov je stanovena na délku 2. Dotazování se uskutečňuje symbolem "~". Tento typ dotazu řeší jiný problém, vyskytující se často v lékařských zprávách a to psaní překlepů. Fuzzy search podporuje nejen hledání podobnosti a vzdálenosti znaků ve slovech, ale i vzdálenosti slov navzájem v souvětích. Užití v praxi může mít v případech, kdy mu klasické dotazy

nevrací relevantní výsledky, pak je lepší použít fuzzy search a zjistit, že se jednalo o překlep. Příklad překlepu:

```
http://127.0.0.1:5984/db/_design/med_lat/by_latin?q=arterip~
```

```
Výsledky:
arteria, 1.3213
arterie  0.9213
arteria nutricia 0.8213
...
```

Dál Lucene podporuje všechny booleovy operátory, seskupování, agregování, escapování, hledání v rozmezí hodnot nebo dokonce zvyšování váh konkrétních termů pro přesnějších výsledků.

Všechny výsledky dotazů se řadí podle VSM - Vector Scoring Model, který je založený na kosinové vzdálenosti jednotlivých vektorů. Jedná se o vylepšenou verzi základního vektorového modelu, který může být počítaný např. Euklidovskou vzdáleností.

5.7 Testování výkonnosti

Za účelem tohoto projektu pro lepší rozhodování jsem zvolil porovnání výkonu databází PostgresSQL, MongoDB a CouchDB. Vytvořil jsem testy, které simulují několik základních scénářů, které se vyskytují v aplikaci mojí diplomové práce a předpokládá se, že se budou často řešit v praxi. U všech databází jsem používal multivláknový program v Javě, který testoval čtení a zápis do vybraných databází. Meřená veličina byla vždy buď počet dokumentů, které je databáze schopna přečíst za fixní čas nebo doba odezvy k přečtením nebo zápisem fixním počtu dokumentů. Tím jsem testoval stabilitu a rychlost jednotlivých způsobů indexování. Každý scénář byl opakovaný

vícekrát. Počítač, na kterém byly testy prováděné, měl následující parametry:

Hardwarová konfigurace:	Softwarové parametry:
MacBook Pro (Late 2013)	OS X El Capitan 10.11.6
Procesor: 2,6 GHz Intel Core i5	PostgreSQL 9.5.3
Paměť: 16GB 1600 MHz DDR3	MySQL 5.6
Disk: 512GB SSD	MongoDB 3.2.8
	Apache CouchDB 1.6.1
	Apache Lucene 4
	CouchDB-Lucene 1.1.0

Scénář 1 - v databázích je 1mln. dokumentů či řádků, ve 4 vláknech program čte po dobu 30 vteřin z databází dotazy vyhledávající automatické předvyplnění zkratky fulltextovými prostředky. Například dorozepsání zkratky "art."na "arteria"apod.

Pokus	Operace	PostgreSQL	MongoDB	CouchDB	CouchDB+Lucene
1	READ	4x 79 ř.	4x 23 doc	4x 72 doc	4x 1948 doc
2.	READ	4x 92 ř.	4x 22 doc	4x 65 doc	4x 1784 doc
3	READ	4x 78 ř.	4x 22 doc	4x 71 doc	4x 2273 doc

Tabulka 5.1: Scénář 1 - počet dokumentů obdržených při paralelním čtení po dobu 30 vteřin. Zkratka ř. znamená řádků, doc - JSON dokumentů

Z testu je patrné, že čtení jednotlivých databází je poměrně stejné s menšími odchylkami, například MongoDB v případě doplňování zkratek, pomocí regulárních výrazů je nejpomalejší, v porovnání s PostgreSQL, která je nejrychlejší ze všech bez použití Apache Lucene. Jako nejmocnejší je v tomto případě zvolená kombinace CouchDB s možností dotazování od Lucene.

Scénář 2 - v databázích je 1mln. dokumentů či řádků, ve 3 vláknech čtu a z 1 vlákna zapisuji po dobu 30 vteřin.

#	PostgreSQL (ř.)		MongoDB (d.)		CouchDB (d.)		Couch+Lucene (d.)	
	čtení	zápis	čtení	zápis	čtení	zápis	čtení	zápis
1.	3x 114	67	3x 22	131387	3x 93	140	3x 2528	2366
2.	3x 131	79	3x 20	131926	3x 95	138	3x 2510	2346
3.	3x 125	75	3x 17	129679	3x 101	151	3x 2495	2327

Tabulka 5.2: Scénář 2 - počet dokumentů obdržených při paralelním čtení a zápisu po dobu 30 vteřin.. Zkratka ř. znamená řádků, doc - JSON dokumentů

Jak je vidět na první pohled, nejrychlejší v zapisování do databáze je tento krát jasný lídr MongoDB, naopak opět ve čtení je nejpomalejší. Je to způsobeno tím, že podporuje pouze hledání celých slov a zpracování regulárním výrazem pro hledání automatické korekce textu zpomaluje proces čtení. Další výrazné hodnoty jsou dobré schopnosti PostgreSQL při čtení ve srovnání s NoSQL konkurenci. Opět kombinace CouchDB a Lucene je v tomto případě nejúspěšnější v počtu přečtených dokumentů za dobu běhu testu 30 s. Další zajímavostí je, že na druhou stranu CouchDB zcela selhává při zapisování, je výrazně pomalejší než MongoDB například. Dá se to zdůvodnit tím, že CouchDB komunikuje přes HTTP REST rozhraní, dokud MongoDB přes driver SDK. Jedná se tak o pomalejší protokol, tím také CouchDB při zpracování HTTP požadavku kóduje a dekóduje data v JSON formátu. Pro účely této diplomové práce jsem se snažil vytvořit program pro velmi rychlé čtení z databází i vyhledávání opravy zkratek, proto pomalejší protokol pro zápis není v tomto případě důležitý.

Scénář 3 - v databázích je 1mln. dokumentů či řádků, ve 4 vláknech čtu 1000 dokumentů či řádků, tzn. 4 vlákna krát 1000 dokumentů. Měřená hodnota je doba odezvy neboli čas zpracování.

Pokus	Operace	PostgreSQL	MongoDB	CouchDB	CouchDB+Lucene
1	READ	4x 290118	4x 1722928	4x 419672	4x 13524
2.	READ	4x 284167	4x 1718261	4x 420515	4x 13458
3	READ	4x 291284	4x 1699172	4x 419891	4x 13673

Tabulka 5.3: Scénář 3 - doba k obdržení [ms.] 4x 1000 dokumentů při paralelním čtení 4 vláknami.

Dle jednotlivých měření je možno spatřit, že nejpomalejší ve čtení je MongoDB. Důvody jsou zmíněné již výše při předchozích měření. Obdiv patří Postgresu za rychlé čtení oproti ostatním databázím. V tomto případě je lepší než CouchDB. Taktéž je znovu vidět síla nadstavby Apache Lucene pro CouchDB, kde je čtení a vyhledávání rychlejší v řádu desítkách milisekund.

5.8 Souhrn

V této kapitole jsem se zabýval porovnáváním několika typů databází pro účely problematiky této diplomové práce. Vybral jsem 2 relační databáze a 2 NoSQL databáze a porovnal jejich možnosti a podpory fulltextového vyhledávání. Aplikace pro korekci medicínských textů pracuje převážně s velkým množstvím textů a nestrukturovaných dat. Dále aplikace pracuje s poměrně velkými slovníky, které je vhodné ukládat do InMemory databáze. Ve své práci jsem zvolil použití databáze Redis pro tyto účely, neboť mám s ni v praxi nejlepší zkušenosti. V závěru lze říct, že v rámci relačních databází MySQL není vhodné řešení pro tento typ úlohy z důvodu všech omezení, které má jako například omezený Boolsky model zpracování fulltextu. Oproti tomu PostgreSQL modul tsvector2 funguje poměrně dobře a je možné ho porovnávat s Apache Lucene, tímto pro relační databáze je volba jednoznačná. Důvodem porovnání je ten, že obě metody zpracování fulltextu jsou založené na vektorovém modelu. Nevýhodou nicméně je, že není NoSQL databáze a tím ztrácí její výhody - distribuovanost, škálovatelnost atd. Dále je potřeba

instalovat a konfigurovat řadu pluginu a knihoven dodatečně do databáze, jako například modul pro n-gramy (`pg_trgm`) atd. Tento aspekt je podle mě nevýhodou z důvodu obtížné údržby a konfigurace databáze do budoucna.

Dalšími databázemi, které jsem porovnával byly MongoDB a CouchDB. Podle CAP teorému hlavní rozdíly jsou malé - MongoDB je zaměřena na plnou konzistenci, dokud CouchDB na úplnou dostupnost. Důvodem volby těchto dvou NoSQL databází je jejich jednoduchost a dobrá podpora v komunitách. případech lze tvrdit, že databáze jsou vhodné pro řešení problematiky opravy a korekce lékařského textu. Základní nastavení MongoDB není postačující a vyhovující při použití v produkčním prostředí. Vychytávka pro vícenásobné zástupné prohledávání, pomocí regulárních výrazů zpomaluje celkový proces čtení a tím je nutná integrace s jiným vyhledavacím enginem. Ve své práci jsem zvolil možnost CouchDB s integrací pro Apache Lucene. Stejná alternativa je možná i pro MongoDB, ale zůstal jsem u varianty CouchDB z důvodu podpory českého analyzátoru pro tokenizaci slov, tak jako jednoduchost používání knihovny LightCouch Java API pro komunikaci s databází přes rozhraní. Stejně tak dobře sepsané dokumentace k integraci těchto dvou open-source produktů oproti integraci MongoDB s Apache Lucene. Dalším důvodem výběru CouchDB s Lucene oproti PostgreSQL je ten, že Lucene zahrnuje v sobě řadu funkcionalit, které není třeba konfigurovat a instalovat dodatečně jako je u Postgresu, například možnost fuzzy search, kde je potřeba do Postgresu instalovat n-gramy a další slovníky. To vše Lucene v sobě už nativně obsahuje. Nezvolil jsem možnost Elasticsearch, protože mi přišlo jako těžkopádné a komplexní řešení pro tento projekt, kde si vystačím pouze s programovacím rozhraním k samotným funkcionalitám Apache Lucene. Ze subjektivního pohledu jsem bral v potaz, že Elasticsearch je vhodný k použití při tvorbě komplexních vyhledávacích realtime webových portálů, které jsou obdobou google vyhledávačů.

6 Datamining

6.1 Teorie

Datamining (Získávání znalostí z databází nebo KDD - Knowledge Discovery in Databases) [14], interdisciplinární podoblast počítačové vědy, [15], je výpočetní proces objevování vzorů ve velkých datových sadách, zahrnující metody na pomezí umělé inteligence, strojového učení, statistiky a databázových systémů.[15] Celko-vým cílem procesu dolování dat je získat informace z datových sad a transformovat je do srozumitelné struktury pro další použití. Zahrnuje také aspekty databáze a správu dat, předzpracování dat, model úvahy, zhodnocení metrik, složitost úvahy, vizualizace atd. Používají se techniky jako rozhodovací stromy, asociační pravidla, regresní, logistická analýza, neuronové sítě či shluková analýza (clustering) pro segmentaci skupin podle společných vlastnosti.

Existuje obecný postup kroků všech datamining metodologií:

1. Inicializační – formulace úlohy a porozumění problému. Často automatické vyhledávání znalostí nelze provádět zcela naslepo.

2. Datový – vyhledání a příprava dat pro analýzu. Statistické algoritmy potřebují data připravená v určité podobě, proto není možné použít přímo surových semistrukturovaných dat z operačních databází.

3. Analytický – hledání informace v datech, vytváření statistických modelů. Nejčastěji používanými metodami však jsou logistická regrese s automatickým výběrem proměnných, rozhodovací stromy a neuronové sítě.

4. Aplikační – zjištěné poznatky a modely je třeba uvést do praxe, například korekce lékařských zkratek.

5. Řízený – je třeba zajistit zpětnou vazbu (jak efektivní byl model) a v případě dlouhodobě nasazovaných modelů i kontrolovat, zda model příliš nezestárl a zachovává si svoji efektivitu.

6.2 Datamining v medicíně

Jak jsem již zmíňoval v kapitole 3.6, největším problémem je samotné před-zpracování dat, zahrnující filtrace, transformace a čištění dat od nestrukturované a semistrukturované podoby do jasně určené podoby, vhodné pro trénování datamining algoritmů. Posléze lze analyzovat dosažené výsledky a hledat vhodné korelace a úvahy pro zhodnocení. Záznamy pacientů se skládájí z klinických, laboratorních parametrů, výsledků jednotlivých vyšetření, které jsou specifické pro různá odvětví a specializace. Tato data mají většinou následující vlastnosti:

- Neúplnost: Chybí hodnoty atributů, chybí některé atributy zájmu nebo obsahují pouze souhrnná data
- Šum: Obsahují chyby nebo odlehlé hodnoty
- Nekonzistentní: Obsahují rozpory v kódech nebo názvech
- Temporální: Parametry chronických onemocnění v čase

Neexistují-li kvalitní údaje, lze tvrdit,že neexistuje ani kvalitní výsledek. Datový sklad pro dolování medicínských dat potřebuje důslednou integraci kvalitních údajů. Řešením je vytvoření rozsáhlého slovníku pojmů, jednotného rozhraní pro integraci více datových zdrojů a předávání elektronických záznamů o pacientech na úrovni mezi jednotlivými nemocnicemi celosvětově. Je dále potřeba porozumění tzv. Medical Domain, neboli v IT je nedostatek odborníků se znalostní domény v oboru medicíny.

7 Výběr algoritmů

Tato sekce popisuje výběr jednotlivých datamining algoritmů API WEKA pro klasifikaci medicínského textu. Algoritmy byly použité z vývojařské knihovny tohoto data-miningového nástroje. Ve své práci jsem zvolil datamining nástroje WEKA z důvodu širokého množství nabízených algoritmů, z důvodu otevřenosti vývoje a open-source licenci, také z důvodu předchozích zkušeností s touto knihovnou. Bylo zapotřebí se s implementací algoritmů velmi podrobně seznámit. Rozdíly konkurenčních open-source datamining nástrojů na trhu jsou zanedbatelné v kontextu řešení problematiky korekce textu v doméně medicíny.

7.1 Naivní bayes

7.1.1 Teoretické základy

Samotný algoritmus, který jsem vybral pro svou práci se zakládá na klasické bayesové větě, která je založená na pravděpodobnostním vzorečku, kterým se po celou dobu algoritmus řídí a rozhoduje, v závislosti na pravděpodobnosti výskytu daných slov, zda zařadí do konkrétní kategorie danou lékařskou zkratku (resp. testovací vzorek). Pravděpodobnostní vzoreček vypadá takto:

$$P(k|doc) = \frac{P(doc|k)P(k)}{P(doc)} \tag{7.1}$$

Kde $k \in K$ je rozepsaná lékařská zkratka z množiny všech možných kategorií, do které lze zařadit danou nalezenou neopravenou zkratku a doc je samotná zkratka, kterou potřebujeme klasifikovat. Pravděpodobnost hypotézy $k \in K$, podmíněna pozorováním medicínské zkratky doc lze tedy vyjádřit jako poměr pravděpodobností, že lékařská zkratka doc patří do dané kategorie k (rozepsaná zkratka), krát apriorní pravděpodobnost kategorie k,

vůči evidenci, což je apriorní pravděpodobnost trénovacích dat (rozepsaných zkratek). Jinými slovy algoritmus je naivní ve svém přístupu, tím že spoléhá na to, že v závislosti na hodnotě pravděpodobnostního výskytu v trénovacím modelu bude zkratka v testovací množině patřit do konkrétní kategorie k (konkrétní rozepsaná zkratka). Jinak řečeno, algoritmus spoléhá na to, že existuje rovnoměrná distribuce.[16] [17]

7.1.2 Naivní Bayes a klasifikace textů

Algoritmus Naivní Bayes je velmi rozšířený mezi klasifikačními algoritmy pro práci s textem. Je to jeden z nejpoužívanějších a nejefektivnějších algoritmů strojového učení pro práci s textem. Praxe ukazuje, že algoritmus pracuje skvěle jak s malým, tak i s velkým množstvím trénovacích dat. Toto bylo osvědčeno v rámci této diplomové práce. Důležité však je kvalitní natrénování modelu. Ohodnocení pak bude prokazovat mnohem méně chyb.

7.1.3 Naivní Bayes Multinomial

Vylepšení původního algoritmu Naivní Bayes, kterého jsem použil ve své práci zejména z důvodu, že jsem potřeboval propracovanější výsledky, je algoritmus Multinomiální Naivní Bayes. Ten se liší oproti původnímu pouze v tom, že používá Multinomcké rozdělení. Klasický Naivní Bayes používá rovnoměrné rozdělení.[16] [17] [18]

Detailní příklad

Podívejme se na detaily, jakým způsobem Multinomiální Bayes klasifikuje své vzorky do odpovídajících tříd. Za prvé je potřeba nadefinovat apriorní pravděpodobnost dané třídy [17] :

$$P(k) = \frac{Nc}{N} \tag{7.2}$$

Kde Nc je počet vzorků trénovacího modelu, popisujících třídu k a N je počet všech vzorků trénovací množiny. Hledáme-li zkratku "a."a máme-li následující testovací větu:

```
String word = "a.";
String context = "Chabější zásobení je také v nejdistálnějším
povodí a. cerebri ant. vlevo.";
```

Pomocí fulltextového vyhledávání získáváme trénovací dataset pro zkratku "a.", který vypadá následovně:

```
@relation fromJSON

@attribute training string
@attribute zkratka {a.,arter.,ACM,ACoP,a.subcl.,Ao.,PICA.}
@attribute vyznam {arteria,arteriální,'arteria cerebri media','arteria test','arteria communicans posterior',
'arteria subclavia',Aortální,'arteria cerebellaris posterior inferior'}

@data
'IC hemorhagie. Perfúze mozková. Nekrotické okrsky v povodí a. cerebri med. l. sin. , periferně místy v terénu ischemie',a.,arteria
'mm (min. 22 mm) vůči okolnímu parenchymu hyperdenzní v arter. fázi (denzity 146-181 HU),
hypointenzní ve venozní fázi (denzity',arter.,arteriální
' vyš. 9.4.2010. Rozsáhlá malacie celém povodí a. cerebri med. sin. progrese nálezu. Ostatní nález ',a.,arteria
'CTAG mozku Uzávěr levé ACM M1 úsek trvá. Fetální odstup levé ACP při',ACM,'arteria cerebri media'
'tr. Z á ě r Známky hyperakutní ischemie povodí ACM levo staré postmalatickéé změny T,P O levo',ACM,'arteria test'
' Diferencovatelná ACoP vpravo, ostatní aa. comm. nelze spolehl. diferencovat.',ACoP,'arteria communicans posterior'
'uzávěr pravé ACI od ostupu, uzávěr a. subcl. vlevo v délce 5 cm, steal sy levé vert. tepny',a.subcl.,'arteria subclavia'
'Z á ě r Známky hyperakutní ischemie povodí ACM levo staré postmalatickéé změny
T,P O levo askulární mikroléze centrum semiovale bilat CT AG yš. bolu k.l..',ACM,'arteria test'
'i ostatní ložiska, která jsou dobře diferencovatelná jen v arter. fázi. Jinak jsou játra bez
patrných nových ložisek. Dc.',arter.,arteriální
'kalcifikace v obl. Ao, odstupy volné. Uzávěr ACI dx od odstupu převážně měkkým plátem.',Ao.,Aortální
'k vyš. z 9.4.2010. Rozsáhlá malacie v celém povodí a. cerebri med. sin. , progrese nálezu. Ostatní nález se nemění,',a.,arteria
'coilingem. Pravá je pod bazí gracilní po odstupu PICA. Zvětšen ý pravý lalok štítní žlázy s
nehomogenním uzlem vel. 301x23mm.',PICA.,'arteria cerebellaris posterior inferior'
```

Obrázek 7.1: Ukázka trénovacích dat ve formátu .arff

pak naše apriorní znalosti P(k) pro rozepsané zkratky jsou:

```
P( doc | Aortální) = 0.051420399348687834
P( doc | arteriální) = 0.5810084605150269
P( doc | arteria cerebellaris posterior inferior) = 0.08891
773105693634
P( doc | arteria communicans posterior) = 0.05829038096
P( doc | arteria) = 0.04254220747182298
P( doc | arteria subclavia) = 0.04778149418255666
```

```
P( doc | arteria test) = 0.0390914561037055
P( doc | arteria cerebri media) = 0.05404002891776255
P( doc | ?) = 0.036907841436640755
```

Model je natrenovaný různými zkratkami stejného typu, získané full-textovým zpracováním a vyhledáváním. S ohledem na to, že v trénovacím modelu mám třídu '?', ke které nepatří žádný trénovací vzorek, tak tyto pravděpodobnosti algoritmus přepočítává a přiřazuje tzv. m-odhad třídy '?'. Proto v celkovém výsledku je tato výsledná pravděpodobnost o něco málo menší. Poté se vypočítávají jednotlivé podmíněné pravděpodobnosti. Pro každé slovo trénovací množiny se vypočte podmíněná pravděpodobnost s jakou může patřit do dané třídy. Můžeme použít následující vzoreček [17] :

$$P(doc|k) = \frac{count(doc, k) + 1}{count(k) + |V|} \tag{7.3}$$

Kde $P(doc|k)$ udává pravděpodobnost dat, za podmínky, že patří do třídy k. Výpočet je snadný a to, tak, že count(doc, k) vyjadřuje četnost slov testovacího vzorku, obsažené v trénovací množině. Z důvodu normalizace se přičítá jednička. Ve jmenovateli count(k) je počet všech slov, týkajících se naší konkrétní třídy k (rozepsanou lékařskou zkratku - arteriální). $|V|$ je tzv. vocabulary neboli slovník všech slov trénovací množiny.

7.1.4 Optimální vylepšení algoritmu

Z důvodu použití knihovny WEKA, bylo potřeba seznámit se podrobněji s implementací používaného algoritmu tohoto open-source produktu. Byly zjištěny malé změny v algoritmu, oproti klasickému učebnicovému vzorečku. Weka používá normalizaci za pomocí logaritmování a odlogaritmování jednotlivých pravděpodobností. Domnívám se, že důvod této implementace je

rychlost ve zpracování výsledků. Normalizace vypadá takto:

$$P(k|doc) = \frac{WP(k)}{P(doc)} \tag{7.4}$$

Kde W = $e^{(log(x)-log(y))}$. Argument x je tedy $P(doc|k)$, což už víme, že je celková podmíněná pravděpodobnost daného testovacího vzorku, patřící do konkrétní třídy k a y je vždy maximální hodnotou ze všech vypočtených x. V podstatě hodnota W se vždy rovná 1 v nejlepším případě, kdy je největší pravděpodobnost, že zkratka patří do kategorie k. Pravda je taková, že z normalizace vyplývá, že e^0=1. Jinými slovy, je-li hodnota W < 1, tak bude v každém případě menší pravděpodobnost, že patří do této kategorie. Je-li W = 1 je největší pravděpodobnost, že patří do dané kategorie. Ovšem k jiným výsledkům je možno se dopracovat v závislosti na apriorních znalostí P(k), což může být způsobeno specifickým natrénováním dat. Například v trénovací množině bude více rozepsaných zkratek v kategorii arteria cerebri media pro zkratku "a.", tím algoritmus bude spíše směřovat numericky k třídě arteria cerebri media v případech, kdy bude váhat, kterou třídu vybrat nebo jsou-li si pravděpodobnosti velmi blízké.

7.2 SMO

7.2.1 Vznik

SMO (zkratka ze Sequential Minimal Optimization) je algoritmus pro řešení problému kvadratického programování (QP), který vzniká při trénování algoritmu SVM (Support Vector Machines). Byl vynalezen Johnem Plattem v roce 1998 ve společnosti Microsoft Research. SMO je široce používán pro trénování SVM a je implementován populární knihovnou LIBSVM. Zveřejnění algoritmu SMO v roce 1998 vyvolal hodně vzrušení v komunitě SVM vývojařů, protože dříve dostupné metody pro trénování SVM byly mnohem

složitější a výpočetně náročnější. [19]

7.2.2 Optimalizační problém SVM

Uvažujme podle binární klasifikace problému s datovými sady (x_1, y_1),...,(x_n, y_n), kde x je vstupní vektor a $y_i \in$ (-1, 1) je binární název odpovídající k němu. Jemné rozpětí SVM je natrénováno k řešení problému kvadratického programování, kde je problém vyjádřen ve tvaru:

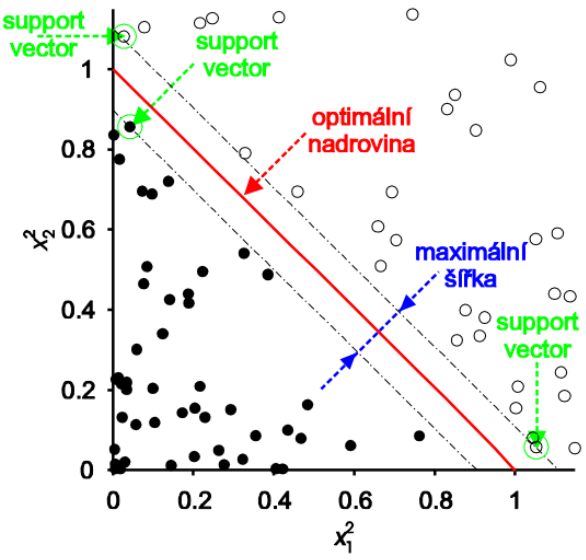
$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j K(x_i, x_j) \alpha_i \alpha_j, \tag{7.5}$$

kde platí:

$$0 \leq \alpha_i \leq C, \qquad \text{pro } i = 1, 2, \ldots, n,$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

kde C je SVM hyperparameter a K(x_i, x_j) je funkce jádra, oba dodávané uživatelem. Proměnné α_i jsou Lagrangeovy multiplikátory. [20] [21] [23]



Obrázek 7.2: Rozhodovací hranice (nadrovina) a stanovení podpůrných vektorů [22]

7.2.3 Algoritmus SMO

SMO je iterativní algoritmus pro řešení problému optimalizace popsaný výše. SMO rozděluje tento problém do série nejmenších možných dílčích problémů, které jsou pak řešitelné analyticky. Vzhledem k lineárnímu omezení rovnosti, která zahrnuje Lagrangeové multiplikátory α_i, nejjednodušší možný problém se týká dvou takovýchto multiplikátorů. Poté, pro libovolné dva multiplikátory α_1 a α_2, pak platí:

$$0 \leq \alpha_1, \alpha_2 \leq C,$$

$$y_1\alpha_1 + y_2\alpha_2 = K$$

a takto zredukovaný problém lze vyřešit analyticky. Je potřeba najít minimum jednorozměrné kvadratické funkce. K je negativní součet rovnice, který v každé iteraci klesá.

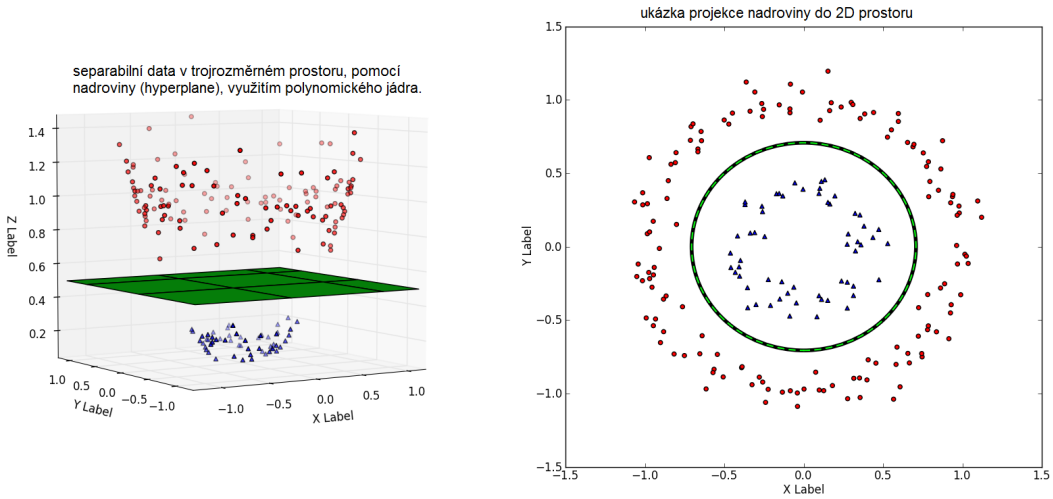
Algoritmus probíhá následujícím způsobem [23]:

1. Nalezne Lagrangeovy multiplikátory α_1, které porušují Karush-Kuhn-Tuckerovo, KKT[1] podmínky pro optimalizační úlohy.
2. Vybere si druhý násobitel α_2 a optimalizuje dvojici (α_1, α_2).
3. Opakuje kroky 1 a 2, dokud nedokonverguje.
4. Když všechny násobky Lagrange splňují podmínky KKT (v rámci tolerance uživatelem definované), problém je vyřešen. Ačkoli tento algoritmus zaručeně vždy dokonverguje, se používají heuristiky pro výběr páru multiplikátorů tak, aby se urychlil postup celého algoritmu.

[1]KKT podmínky jsou nutné podmínky pro hledání optimálního řešení úlohy nelineárního programování, za předpokladu, že i některé další podmínky jsou splněny. Je to zobecnění metody Lagrangeových multiplikátorů na omezující podmínky neobsahující rovnost (může tedy obsahovat nerovnosti)

V nejhorším případě dosahuje asymptotyckou složitost $O(n^3)$.

Optimalizace algoritmu je nastavená v konfiguraci klasifikátoru. Klasifikátor jsem ponechal tak, jak byl v základním nastavení, byla využita metoda 1 vs. 1 pro výpočet pravděpodobností. Tato metoda je lepší než jednodušší předchůdce 1 vs. All, kde se porovnávala vždy data 1 klasifikační třídy versus zbytek tříd. V metodě 1 vs. 1 se porovnávají všechny možné permutace tříd navzájem pomocí kernelů a tím se vytváří přesnější porovnání a lepší tvorba trénovacího modelu. Dalším základním parametrem nastavení byla volba jádra. Pro účely řešení problému medicínských zkratek jsem dospěl, že nastavení Polykernelu (polynomické jádro) je postačující a tím se jednalo o vhodnou optimalizaci úlohy - korekce a klasifikace medicínského textu. Na obrázku je znázorněná projekce z 3D do 2D prostoru, využitím polynomického jádra v algoritmu SMO WEKA.



Obrázek 7.3: Projekce polynomického jádra do 2D prostoru k nalezení modelu. [22]

7.3 J48

J48 je open source Java implementace C4.5 algoritmu, generující rozhodovací strom. C4.5 staví rozhodovací stromy z trénovacích dat stejným způsobem jako algoritmus ID3 [2], pomocí metody informační entropie. Je vylepšený o tzv. "pruning"(prořezávání stromu) a optimalizovaný proti přeučení (over-fitting). Trénovací data jsou vyjádřena množinou S = $s_1, s_2, ...$ již klasifikovaných vzorků. Každý vzorek s_i se skládá z n-rozměrného vektoru $(x_{1,i}, x_{2,i}, ..., x_{n,i})$, kde x představují atributy nebo vlastnosti vzorku, jakož i jako třídu, v níž s_i spadá. [24]

V každém uzlu stromu, C4.5 vybírá atribut, který nejúčinněji rozděluje trénovací sadu vzorků do podskupin posilujících jednu nebo druhou třídu. Kritériem rozdělení(prořezávání stromu) je normalizovaná informace zisku (rozdíl entropie). Atribut s nejvyšším normalizovaným informačním ziskem je vybrán do role rozhodujícího. C4.5 algoritmus se pak opakuje na poduzlech.

7.3.1 Rozdělení C4.5

Tento algoritmus má několik základních případů. [26]

1. Všechny vzorky v seznamu patří do stejné třídy. Když toto nastane, algoritmus vytváří listový uzel, který při rozhodování klasifikuje texty vždy do stejné třídy.

2. Žádný z atributů nepřináší žádný informační zisk. V tomto případě, C4.5 vytváří rozhodovací uzel abstraktně výš od kořene a používá očekávanou hodnotu třídy.

[2] ID3 - Iterative Dichotomiser 3 je algoritmus generující rozhodovací strom, vynalezeny Rossem Quinlanem

3. Nalezne třídu se kterou se nesetkal. Opět platí, že C4.5 vytváří rozhodovací uzel výše stromu pomocí očekávané hodnoty.

7.3.2 Příklad fungování algoritmu

V pseudokódu, obecný algoritmus pro vytváření rozhodovacích stromů funguje následovně: [24] [25]

1. Kontroluje pro základní případy
2. Pro každý atribut a
   (a) Vypočte jednotlivé informační zisky
   (b) Vyhledá normalizovaný podíl získané informace z prořezání stromu v a
3. Nechť a_{best} atribut je nejlepší normalizovaný informační zisk
4. Vytvoří rozhodovací uzel, který rozděluje v a_{best}
5. Opakuje na poduzlech získaných rozdělením v a_{best}, posléze přidává tyto uzly jako potomky uzlu

Ukázka vygenerovaného stromu pro rozhodování zda pacientka má rakovinu prsou, na základě několik atributů, které nejúčinněji rozdělují trénovací sadu na podskupiny lze vidět v příloze A, na obrázku 12.2. Jedná se o atributy velikost uzlu, velikost nádoru či poloha uzlu(nahoře, dole atd.). [31]

7.4 IBk

Algoritmus IBk implementuje metodu k-nejblížších sousedů. Ve strojovém učení, algoritmus k-nejbližších sousedů (nebo k-NN v krátkosti, zkrá-

cené z k-Nearest Neighbours) spadá mezi neparametrické [3] metody klasifikace. [28]

Při k-NN klasifikaci, výstupem je příslušná třída. Vstupní vzorek je klasifikován na základě hlasování svých sousedů. Testovací vzorek je přiřazován k příslušné třídě, jejíž vzorky jsou nejběžnější mezi k- nejbližších sousedů ($k \in N_+$). Pokud k = 1, pak je vzorek přiřazen třídě jediného nejbližšího souseda.

k-NN je typ učení, založené na instancích[4], nebo též lazy metoda(líná), kde funkce je aproximována pouze lokálně, a všechny výpočty jsou odloženy až do samotné klasifikace. K-NN algoritmus patří mezi nejjednodušší ze všech algoritmů strojového učení. [28] [27] [29]

Nedostatkem algoritmu k-NN je to, že je citlivý na lokální strukturu dat. Algoritmus nemá nic společného s algoritmem k-means, který je další populární metoda strojového učení.

7.4.1 Princip

Existuje několik možností výběru nejbližších sousedů. Základní metriky aplikované v algoritmu kNN jsou popsané níže v tabulce 7.1. [29]

Pro účely korekce lékařských zpráv jsem si postačil se základním nastavením tohoto algoritmu. Používal jsem euklidovskou vzdálenost jednotlivých sousedů vůči hledaného vzorku. Důvodem je malé množství trénovacích dat, tím i celková asymptotická složitost algoritmu nepřesahovala O(n).

[3] Neparametrické metody klasifikace - tyto metody jsou založeny na podstatně slabších předpokladech než metody parametrické, neboť u nich nepředpokládáme znalost tvaru pravděpodobnostních charakteristik tříd.

[4] Učení založené na instancích - buduje hypotézy přímo z trénovacích instancí. Jinými slovy, složitost hypotézy může růst exponenciálně s přibývajícími daty, v nejhorším případě, hypotéza je seznam n trénovacích vzorků. Výpočetní složitost klasifikace jedné nové instance je O(n). Jednou z výhod, které tato metoda má oproti jiným metodám strojového učení je její schopnost přizpůsobit svůj model na dosud nespatřená data.

Metrika	Matematické vyjádření
Euklidovská vzdálenost	$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$
Hammingova (Manhattan) vzdálenost	$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} \lvert a_r(x_i) - a_r(x_j) \rvert}$
prekrytí (overlap)	$d(x_i, x_j) = \sum_{r=1}^{n} (1 - \delta(a_r(x_i), a_r(x_j)))$
kosínova metrika	$d(x_i, x_j) = \frac{\sum_{r=1}^{n} (a_r(x_i), a_r(x_j))}{\sqrt{\sum_{r=1}^{n} (a_r(x_j), a_r(x_j)) . \sum_{r=1}^{n} (a_r(x_i), a_r(x_i))}}$

Tabulka 7.1: Metriky pro nalezení k nejbližších sousedů

7.4.2 Příklady použití

S narůstajícím množstvím trénovacích dat, narůstá též chybovost nebo nepřesnost tohoto algoritmu. Nejúspěšnější volba k záleží na datech. Obecně platí, že vyšší hodnoty k snižují rozptyl při klasifikaci, ale způsobují méně zřetelné hranice mezi třídami. Vhodně velké k může být zvolené různými heuristickými technikami.

Přesnost k-NN algoritmu může být vážně snížena přítomností hlučných nebo irelevantních příznaků (klasicky nepotřebná slova, spojky apod.).[28] [27]

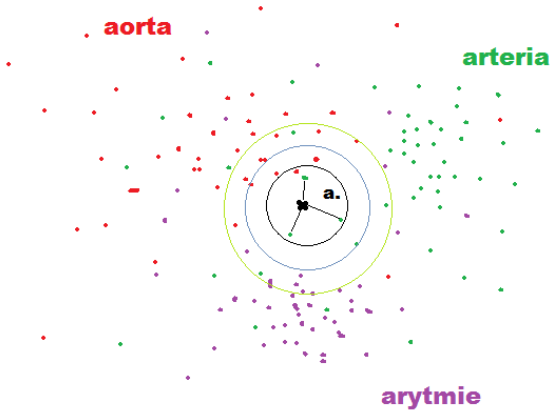
- **1-NN** - Zjistíme vzdálenosti všech prvků trénovací množiny od neznámého prvku. Vybereme daný prvek trénovací množiny, který je nejblíže a neznámý prvek klasifikujeme do stejné třídy.

- **3-NN** - Kolem neznámého prvku vytvoříme hyperkouli, která obsahuje právě tři nejbližší prvky trénovací množiny. Neznámý prvek klasifikujeme do té třídy, která je v hyperkouli zastoupena největším počtem prvků.

- **k-NN** - Při použití metod k-NN pro $k > 1$ je velmi důležitá volba k. Pro dvě třídy volíme k vždy liché (kvůli jednoznačnosti rozhodování) pro více tříd mohou nastat situace, kdy nelze jednoznačně rozhodnout.

Nejdůležitější u tohoto algoritmu bylo nastavení parametru K, nebo-li

nastavení nejbližších sousedů dle Euklidovské vzdálenosti. Na základě několika empirických pokusů jsem dospěl k tomu, že nastavení klasifikátoru kNN je nejoptimálnější pro K = 1. Toto též bylo základní nastavení klasifikátoru. Poněkud zajímavý byl fakt, při němž jsem zjistil, že čím je vyšší nastavení parametru K tím byly výsledky horší a nepřesné. Algoritmus klasifikoval špatně a nesprávně. U některých případů dokonce při nastavením K = 2 nebo K = 3 algoritmus vykazoval nesprávné výsledky. To bylo důvodem ponechat dataminingový algoritmus pro klasifikaci textu z medicínských dat s nastavením K = 1. Grafické znázornění tvorby modelu pro klasifikační třídy aorta, arteria, arytmie se vstupní testovací zkratkou "a."se může zobrazit následovně:



Obrázek 7.4: Příklad k-NN

Z obrázku vyplývá, že tento model je velmi citlivý na nastavení parametru K. Každý bod v prostoru znázorňuje trénovací slovo v trénovací množině dat. Čím více společných slov má testovací věta s trénovací množinou, tím lépe bude zklasifikována. Problém tohoto algoritmu je jeho naivní předpoklad, že ti sousedé, kteří jsou nejblíž jsou nejsprávnějšími.

8 Implementace řešení

8.1 Volba vývojového prostředí

Pro splnění účelu práce bylo nutné nejprve najít vhodný nástroj a to takový, který bude splňovat dvě hlavní kritéria. První z nich byla implementace v jazyce Java, dalším pak, aby byl nástroj open source. Na internetu existuje řada nástrojů a knihoven pro vývoj aplikací umělé inteligence. Nezvolil jsem knihovnu RapidMiner[1], protože se jedná o komerční end-to-end řešení, které je zaměřeno hlavně na cílového zákazníka a nabízí hezké GUI rozhraní. Dokumentace v Javě mi nepřišla dostatečně srozumitelná. Dalším uvažovaným nástrojem pro datamining byl Orange[2]. Jedná se o nástroj pro dataminingové modelování, který je zaměřený zejména na širokou veřejnost. Tím nebyl vhodný pro vývoj této aplikace. Jiný uvažovaný dataminingový nástroj byl KNIME[3], který poskytuje všechny potřebné možnosti pro dolování znalosti a analýzu textu. Nezvolil jsem tento nástroj, protože se jednalo o další komerční řešení na trhu, které je určeno pro jinou cílovou skupinu než pro vývojaře. Z toho důvodu se mi špatně orientovalo v jejich dokumentaci a popis možnosti integrace v Javě. Jako nejvhodnější a nejvíce používaný v komunitě Java vývojařů byly knihovny WEKA poskytnuté a vyvíjené v Novozelandské univerzitě Waikato. Předností tohoto nástroje jsou jeho obsáhlost, pokročilost a optimalizovanost. Dále se mi při programování s touto knihovnou snadno pracovalo a za půl roku jsem se naučil poměrně dobře s ní pracovat.

[1] RapidMiner - `https://rapidminer.com/`
[2] Orange - `http://orange.biolab.si/`
[3] KNIME - `https://www.knime.org/knime-analytics-platform`

8.2 WEKA API

WEKA (zkratka z Waikato Environment for Knowledge Analysis) je prostředí pro analýzu znalostí. Obsahuje balík programů strojového učení napsaný v Javě, vyvinutý na University of Waikato, Nový Zéland. Weka je svobodný software dostupný pod GNU licencí. Tyto dva předpoklady naplňují cíle této práce a to byl důvod, abych si vybral tuto knihovnu jako primární zdroj algoritmů. [30]

8.2.1 Format vstupních dat

Podporovaný formát, ve kterém jsou zpracovávaná data je **.arff**. Framework nabízí řadu metod jak manipulovat s .arff soubory nebo konvertovat data z jiných formátů (jako např.: .csv, .json, .txt apod.) do formátu .arff.

Soubor definuje 2 hlavní části, se kterými pracuje. První část obsahuje hlavičku, ve které definuje název relace a atributy, a druhá část je tělo ve které se nachází samotná data. V mém případě atributy jsou 2 typy - text a class (atribut text typu string je textová hodnota, což je samotný text daného článku a atribut class je její třída, nebo-li kategorie ke které patří z n možných). Takto strukturovaná data jsou zpracovávaná vnitřně, za pomoci speciálních metod frameworku. Ukázka formátu trénovacích dat po transformaci do formátu arff je možné vidět na obrázku 7.1 na straně 52. Kód psaný ve formátu ARFF je case-insensitive, nerozlišuje mezi velikostí písmen (např. příkazy @relation a @RELATION jsou stejné). Dále mezery mezi klíčovými slovy a mezi jednotlivými hodnotami jsou nevýznamné. [30]. Popis použitých atributů:

- **String**

  Atributy String umožňují vytváření atributů, které obsahují libovolné textové hodnoty. Je vnitřně reprezentován jako číselná hodnota (vektor), proto je potřeba použít filtr pro manipulaci řetězce (např.: String-

ToWordVectorFilter). Atributy String jsou deklarovány takto:
ATTRIBUTE zkratka string

- **Nominální atributy**
  Nominální hodnoty jsou definované jako jmenné specifikace seznamu možných hodnot.

  ```
  <nominal-name1>,<nominal-name2>,<nominal-name3>, ...
  ```

  Atribut typu nominal pak může nabývat pouze jedné z uvedených hodnot. Příkladem může být atribut dx obsahující tři nominální hodnoty reprezentující medicínskou zkratku:
  @attribute class dx,dex,dextra
  Atribut tímto definuje možné klasifikační třídy, které lékařská zkratka může nabývat. Případně to můžou být klasifikační třídy, které mají význam výstupní predikce.

8.2.2 Datová část

Datová část formátu ARFF slouží k definování jednotlivých hodnot atributů deklarovaných v hlavičce, jinými slovy k ukládání konkrétních dat pro jednotlivé hlavičkové informace. Datová část se deklaruje @data na nový řádek. Chybějící hodnoty zapisujeme pomocí znaku ?. Každý řádek reprezentuje jednotlivou instanci trénovacích dat. Podle pořadí datových hodnot se pozná, která datová hodnota patří ke konkrétnímu atributu. Záleží tedy na pořadí hodnot a také na dodržení jejich počtu, který musí být shodný s počtem atributů v hlavičce. Pokud máme 3 atributy, pak každá instance musí obsahovat tři hodnoty. Za poslední hodnotou v instanci se již čárka nepíše.[30]

8.3 Rozvržení aplikace

Tato podkapitola popisuje způsob navržení aplikace. Implementace je abstraktní a znovupoužitelná. Jinými slovy, při programování jsem přemýšlel globálně a navrhoval aplikace tak, aby zpracovávala použité algoritmy stejným způsobem a nebylo třeba měnit nic, kromě výběru samotného algoritmu. Metody pro trénování a klasifikaci dat jsou izolované, robustní a znovupoužitelné.

8.3.1 Backend

Backend této aplikace byl napsaný v jazyce Java. Jedná se o webový server v Javě, který obstarává zpracování požadavků klienta a zprostředkovává komunikaci s databází. Aplikace obsahuje několik mapování serveru, pro jednoduchou textovou korekci, založenou na fulltextových prostředcích CouchDB a Lucene. Dále obsahuje mapování, obstarávající požadavky klienta na dataminingovém zpracování. To bylo navrhnuto jako nadstavba původního řešení, využívající algoritmy dataminingu, tak jako trénovací data, uložená v databázi.

Popis balíků programu:

1. **cz.zcu.fav.kiv.mre.controllers** - obsahuje třídy kontrolerů, obstarávající klientské požadavky a mapování serveru, volané ve frontendu u jednotlivých modulů.

2. **cz.zcu.fav.kiv.mre.datamining** - obsahuje třídy pro datamining. Konkrétně tvorbu klasifikátoru, metody pro zpracování textu do podoby vektoru (tzv. StringToWordVector filtr), dynamické vytváření trénovacích a testovacích dat do tzv. instancí, připravené pro použití klasifikátoru, tak jako následné vyhodnocení výsledků a predikce, nebo-li klasifikace testovacího vzorku do konkrétní třídy dle trénovcí

hypotézy.

3. **cz.zcu.fav.kiv.mre.filters** - Filtr pro Basic Auth autorizaci
4. **cz.zcu.fav.kiv.mre.generators** - třídy pro generování zkratek ze slovníků.
5. **cz.zcu.fav.kiv.mre.JSONDocs** - pomocné třídy. Kostry entit.
6. **cz.zcu.fav.kiv.mre.listener** - inicializační naslouchávač (Listener), který importuje Ispell slovníky do Redis InMemery databáze.
7. **cz.zcu.fav.kiv.mre.text_analysis** - třídy obstarávající metody pro generování zkratek ze slov či souvětí.
8. **cz.zcu.fav.kiv.mre.utils** - pomocné metody pro připojení k datatabázi.

8.3.2 Frontend

Vývoj fontendu byla nejnáročnější část celého řešení. Rozhodl jsem se naprogramovat robustnější řešení v podobě pluginu do open-source textového editoru CKEditor v JavaScriptu. Tím jsem navrhnul 3 pluginy, které je možno libovolně vložit či vybrat jako samostatné moduly pro CKEditor:

1. **Plugin pro jednoduchou kontrolu pravopisu a překlepů** - pluginem jsem obstarával jednoduchou kontrolu textu a jeho jednoduchou opravu, navrhováním tzv. (suggestions), nebo-li oprav a korekce slov v textu, která byla označováná či podtrhnutá jako neznámá, nesprávná nebo chybná.
2. **Plugin pro kontrolu pravopisu a překlepů s nadstavbou pro datamining** - do modulu jsem navíc přidal možnost získávání kontextu kolem hledaného slova za účely dataminingu. Rozdíl je pouze backendový ve výpisu navrhovaných slov a jejich pořadí.

3. **Plugin pro plně automatizovanou korekci textu** - plugin zcela automaticky získává návrhy oprav slov fulltextovými prostředky a nahrazuje neznámá či nesprávná slova a zkratky za obdrženými návrhy ze strany serveru. Tento plugin má praktické využití pouze ve fázi vývoje.

8.3.3 Regulární výrazy a jejích role

Velmi důležitou roli při získávání kontextů kolem hledaných zkratek měly regulární výrazy, které jsem ve své práci používal pro účely dataminingu. Příklad funkce v JavaScriptu pro získávání kontextu kolem zkratky:

```
var text;
var medicalText = this.parser.getText(this.config.getText);
var reStr = "((?:[a-zA-Z'-]+[^a-zA-Z'-]+){0,10}\\b"+
                  this.parser.cleanWord(word)+
            "\\b(?:[^a-zA-Z'-]+[a-zA-Z'-]+){0,10})";

    var regex = new RegExp(reStr,'gm');
    var str = medicalText[0]
    var m, contexts = [];
    while ((m = regex.exec(str)) !== null) {
        if (m.index === regex.lastIndex) {
         regex.lastIndex++;
        }
        contexts.push(m[0]);
    }
```

Symbol `"\b"` - tzv. boundary, definuje hranici obalující hledanou zkratku a vrací pouze celá slova či písmena oddělená mezerami a dalšími interpunkčními znaménky.

Výraz `"[a-zA-Z'-]+[^a-zA-Z'-]"` prohledává slova obsahující malá písmena od a do z tak, jako velká písmena od A do Z, následované nepovinně všemi ostatními znaky (interpunkce -,;:. atd.) kromě písmen abecedy. To zajišťuje negace - symbolem `"^"`.

Dále regulární výraz využívá kvantifikátor, vajádřený symbolem `"{0,10}"`, který určije opakování nalezených shod předchozí skupiny. Tím získáváme kontext kolem hledané zkratky o vzdálenosti 10 slov před a 10 slov po hledané zkratce.

V regulárním výrazu taktéž využívám tzv. groups, neboli seskupování nalezených výsledků výrazu. Ty jsou definované kulatými závorkami na začátku a na konci výrazu. Dalé jsou definované před hledanou zkratkou a po ní. Slouží k obalení podvýrazu. Například: group(0) bude pole, obsahující 10 slov před hledanou zkratkou, dokud group(1) bude pole, obsahující 10 slov po ní.

Ve výrazu `"var regex = new RegExp(reStr,'gm');"`, který spouští regulární výraz jsem nastavil taktéž 2 parametry. Parametr "g"uplatňuje výraz globálně na celém textu, nikoliv pouze na první nalezenou shodu v pořadí. Parametr "m"umožňuje hledání začátku a konce nového řádku, nejen čistě textového řetězce. Tím jsem byl schopný detekovat nové řádky v lékařských zprávách.

K dalšímu získávání přesnějšího kontextu kolem zkratek jsem uvažoval o využití metod umělé inteligence a zpracování přirozené řeči (tzv. NLP). Pro tyto účely jsem se snažil najít vhodnou knihovnu, která by mi k nalezení přesnějších kontextů (např. přesný začátek či konec věty, kde se zkratka nachází) vyhovovala. Pracoval jsem s knihovnou Polyglot v Pythonu[4], která toto částečně umožňovala, ale v závěru jsem se rozhodl, že se jedná o příliš komplikované řešení, které by zbytečně zpomalovalo průběh zpracování, proto je vhodné pro tento typ úlohy použít řešení v Javě. Základní imple-

[4]Odkaz na dokumentace knihovny Polyglot v Pythonu s podporou češtiny - `http://polyglot.readthedocs.io/en/latest/Download.html?highlight=czech`

mentaci a integraci této knihovny jsem ve své práci zahrnul, ale v konečném výsledku jsem ho nepoužil z důvodu obtížného zpracování a zpomalení i zkomplikování celkového nasazení a údržby aplikace.

8.3.4 Architektura

Na diagramu architektury v příloze A, na obrázku 12.5 je vidět architektura nasazení celého projektu v praxi. Na diagramu jsou vidět backendová mapování serveru, která jsou následně obstarávaná v Javě. Každé volání pro korekci konkrétní lékařské zkratky vyvolává minimálně 8 fulltextových dotazů, které Apache Lucene zracovává a vyhodnocuje. V nejhorším případě, kdy zkratka neexistuje ani v databázi generovaných zkratek, čili fulltextové vyhledávání nedokáže hledaný řetězec najít, se aplikují další fulltextové metody vyhledávání, pomocí Levensteinové vzdálenosti, tzv. fuzzy search o vzdálenosti 2 znaků. V tomto případě aplikace vyhodnotí celkem 12 fulltextových dotazů, o 4 více oproti základním vyhledávání. V případě dataminingu se spouští ještě 4 fulltextové dotazy navíc. I přes takové množství dotazů pouze pro 1 hledanou zkratku je výkon aplikace poměrně slušný a použitelný pro realtime opravu textu.

Dle diagramu v příloze A, na obrázku 12.5 je možné spatřit, že v prvním kroku uživatel nejprve posílá celý text lékařské zprávy na server, kde server prochází slovníky, uložené v Redis InMemory uložišti a porovnává, zda jednotlivá slova textu jsou obsažena ve slovnících či nikoli. Na základě toho, server vrací pole nesprávných slov a ty jsou posléze barevně odlišena (podtržena červenou barvou) ke korekci. Další mapování serveru slouží pro volání metod "get_suggestions", nebo-li získávání návrhu oprav konkrétní zkratky či překlepů v prostředí CKEditor. Další důležité mapování je "get_dm_suggestions", které volá metody dataminingu navíc od klasického opravování textu a posílá je na server parametr "context", který obsahuje kontext kolem dané zkratky. Bez tohoto kontextu není možné výsledek

správně zhodnotit a získat korektní predikci (klasifikaci) vzorku do příslušné třídy.

8.4 Uložení trénovacích dat

Trénovací vzorky dat byly ukládáné do NoSQL databáze CouchDB. Odkud pak byly dotazované a používané pro trénování algoritmů strojového učení.

8.4.1 Struktura databáze

Na nasledujících příkladech je vidět ukázka uložení dat medicínských slovníků:

```
{
   "_id": "4b910e01a39154b55e1e0ea6f690a596",
   "_rev": "1-1b4d5183dc334ea4a0a4325d55c07ec6",
   "DATAWORKS_DOCUMENT_TYPE": "user14169_czech_latin_dictionary",
   "czech": "Zvýšené množství cukru ",
   "latin": "hyperglycaemia"
}
```

Na výše uvedeném příkladě je vidět česko-latinský medicínský slovník, který byl indexovaný pro fulltextové vyhledávání přes Apache Lucene nad atributů "latin"a "czech". Tímto způsobem jsme schopni snadno vyhledávat odvození nebo zkratky z textů obou atributů, například zkratku "hypergl."apod.

Dále jsem ukládál zkratky z českých nemocnic do tvaru:

```
{
   "_id": "63e5c848fa2211c3b063d6feccd585ed",
   "_rev": "1-42d5aaece38d3f3fdc0745c59d8a9827",
```

```
    "DATAWORKS_DOCUMENT_TYPE": "user14169_slovnik_medical",
    "vyznam": "Frakcionovaná radioterapie",
    "zkratka": "FRT"
}
```


V tomto příkladě se jednalo o zkratky, které zvolené nemocnice v ČR používají jako standard lokálních nemocnic a jsou veřejně vypsané či dostupné v úkonech nemocnice[5].

Dále jsem ukládál také data, která sloužila pro účely trénování dataminingalgoritmů.


```
{
    "_id": "1469447078558",
    "_rev": "1-46b94a2d0f5ac25dcff2c51ddfc0cb62",
    "VYZNAM": "Arterie",
    "ZKRATKA": "Aa.",
    "TRAINING-SENTENCE":
    "odstup rovněž uzavřen. Vpravo ACC ACE ACI volné.
    Aa. vertebrales volné.  Intrakarniálně jinak Willisův okruh"
    }
```


Jak je vidět, přibývá navíc i atribut "TRAINING-SENTENCE", kde text je očištěný od stop slova, což snižuje celkovou dobu zpracování a vyhodnocování algoritmem.

[5]Příklad zkratek převzatých z úkonů FN Plzeň pro rentgenologické oddělení. `http://old.fnplzen.cz/pracoviste/inc/rdgb/SOPRD_RDGB_0_079_00_02.docx`.

9 Porovnávání dataminingových metod

Tato kapitola popisuje matemetické tvary hodnotících kritérií pro výběr nejvhodnějšího algoritmu. Rozhraní Weka nabízí výpočet těchto metod, ze které jsem vycházel při analýze výsledků.

9.1 Hodnotící kritéria

K ohodnocení kvality natrénovaných modelů jsem použil matriky nabízené používaného frameworku pro analýzu klasifikátorů. Při porovnávání byly vypočítávány následující hodnoty[32]:

- **Průměrná absolutní chyba (Mean Absolute Error)** - Ve statistice, tato hodnota (MAE) je veličina používaná k měření, jak blízko predikce nebo předpovědi jsou k případným skutečným výsledkům. Počítá se následovně:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|p_i - a_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|\,. \tag{9.1}$$

  Z názvu lze vydedukovat, že střední absolutní chyba je průměr absolutních chyb $|e_i| = |p_i - a_i|$, kde p_i je predikce a a_i skutečná hodnota. Průměrná absolutní odchylka je většinou mírou chybné předpovědi pro analýzu časových řad [1], kde pojem "průměrná absolutní chyba"je někdy používán k záměně s více standardními definicemi střední absolutní odchylky.

[1] Časová řada stručně představuje soubor takových pozorování x_i, které jsou získány (naměřeny) ve specifickém čase t. Dále můžeme rozlišovat tzv. stochastické a deterministické časové řady nebo aditivní, multiplikativní a smíšené. [2]

- **Střední kvadratická odchylka - (Root mean square error, tzv. RMSE)** je často používaná míra rozdílů mezi hodnotami předpovídaného modelu a hodnotami skutečně pozorovanými. RMSE představuje ukázku směrodatné odchylky rozdílu mezi předpokládanými hodnotami a pozorovanými hodnotami. Tento rozdíl se nazývá rezidua. Počítá se matematickým vzorcem:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(p_i - a_i)^2} = \sqrt{MSE} \tag{9.2}$$

  Kde MSE je Mean Squared Error (Střední kvadratická chyba), p_i je predikce a a_i je skutečná aktuální hodnota. Hodnoty veličiny RMSE pro každý klasifikátor slouží jako agregátor chyb predikce v čase. Veličina RMSE je silné měřítko, používané především pro porovnávání jednotlivých algoritmů na základě naměřených chyb v modelu. `http://www.saedsayad.com/model_evaluation_r.htm`

- **Relative Absolute Error - Relativní absolutní chyba** je velmi podobná relativní čtvercové chybě v tom, že je relativní vzhledem k jednoduché předpovědi. V tomto případě odchylka je celková absolutní chyba namísto celkové čtvercové chyby. To znamená, že relativní absolutní chyba se vypočítává jako celková absolutní chyba a normalizuje se vydělením celkové absolutní chyby jednoduché předpovědi. Matematicky lze relativní absolutní chybu e_i vyjádřit:

$$\mathrm{RAE} = \mathrm{e_i} = \frac{\sum_{i=1}^{n}|P_{ij} - a_j|}{\sum_{i=1}^{n}|a_j - \hat{a}|} \tag{9.3}$$

  Kde P_{ij} je hodnota předpovědi i pro vzorek dat j (z n vzorků), a_j je cílová hodnota pro vzorek j a $\hat{a}$ je dáno vzorcem:

$$\hat{\mathrm{a}} = \frac{1}{n}\sum_{i=1}^{n}|a_i| \tag{9.4}$$

Pro dokonalé výsledky, čitatel musí být roven 0 a $e_i = 0$. Tím dosáhneme toho, že index e_i se pohybuje v rozmezí od 0 až do nekonečna, kde 0 odpovídá ideálnímu případu (dokonalá přesnost výsledků). V praxi se snažíme tuto hodnotu minimalizovat, ne vždy je to možné.

- **Root Relative Squared Error - kořenová relativní čtvercová chyba** je relativní vůči tomu, co by bylo, kdyby byla použita jednoduchá předpověď. To znamená, že relativní čtvercová chyba získává celkovou čtvercovou chybu a normalizuje ji vydělením celkové čtvercové chyby jednoduchou předpovědí. Takto získané druhé odmocniny relativní čtvercové chyby snižuje chybu do stejných dimenzí jako je samotná jednoduchá předpověď. Matematicky, kořenová relativní čtvercová chyba e_i se vyhodnocuje podle rovnice:

$$\mathrm{RRSE} = \mathrm{e_i} = \sqrt{\frac{\sum_{i=1}^{n} (P_{ij} - a_j)^2}{\sum_{i=1}^{n} (a_j - \hat{a})^2}} \tag{9.5}$$

  Kde P_{ij} je hodnota předpovědi i pro vzorek dat j (z n vzorků), a_j je cílová hodnota pro vzorek j a $\hat{a}$ je dáno vzorcem:

$$\hat{\mathrm{a}} = \frac{1}{n} \sum_{i=1}^{n} |a_i| \tag{9.6}$$

  Znovu pro dokonalé výsledky, čitatel musí být roven 0 a $e_i = 0$.

9.2 Další kriteria

Další kriteria nabízena frameworkem k ohodnocení úspěšnosti klasifikace textu byly následující metriky:

- TP Rate: Míra pravdivých pozitiv (instance správně klasifikované do

dané třídy) Příklad výpočtu:

- FP Rate: Míra falešných pozitivních (instance nesprávně klasifikovaná do dané třídy)
- Přesnost: Přesnost je podíl instancí z dokumentů získaných, které jsou relevantní pro informační potřeby uživatele.
- Kappa koeficient Cohen je míra souhlasu v rozsahu hodnot 0-1.

$$\mathrm{K} = \frac{P(A) - P(E)}{1 - P(E)} \tag{9.7}$$

  Kde P(A) je procentuální soulad mezi realitou a klasifikátorem, P(E) je podíl náhodné shody. K=1 znamená plná lineární závislost veličin, K=0 je žádná lineární závislost.

Na základě těchto koeficientů lze hodnotit nejlépe správně klasifikované-/predikované lékařské termíny a medicínské zkratky. Dále jsou tyto metriky vhodné i pro určování nesprávně klasifikovaných instancí. Ve své práci jsem se zaměřil na 4 základní kritéria - MAE, RAE, RMSE, RRSE.

9.3 Zhodnocení výsledků

9.3.1 Jednotkové výsledky

Z poskytnutých datových sad jsem zvolil jednu lékařskou zprávu, kterou jsem se snažil zcela opravit od překlepů, zkratek a chyb. Zpráva je k nalezení v příloze A, na str. 92. Vybral jsem pouze jednu, protože každá zpráva obsahuje velké množství zkratek, překlepů a odborných abreviatúr. Tím bylo porovnávání datamininingových algoritmů komplikované. Uvažoval

jsem o situaci, kdy počet zkratek ve zprávách se může lišit, tím nezáleží na tom, zda bude zvolena jedna zpráva, obsahující kolem 20 zkratek, nebo 5 zpráv, které budou obsahovat celkově kolem 20 zkratek. To bylo důvodem vybrat jednu medicínskou zprávu, pro kterou jsem hodnotil kvalitu výsledků jednotlivých algoritmů a chybovost trénovacích modelů podle jednotlivých zkratek. Hlavním kriteriém úspěchu je počet správně opravených, nebo-li správně zklasifikovaných zkratek jednotlivých algoritmů. Vybraná lékařská zpráva obsahovala 202 slov, ze kterých 23 neznámých tvarů slov či zkratek. Ostatní slova byla ve správném gramatickém tvaru, obsažena ve slovnících se standardizovanými slovy, ktera jsem ukládal do Redis databáze. Tím jsem se zaměřil nad opravou těchto 23 zkratek a neznámých slov. Jednalo se převážně o medicínské zkratky v latině, češtině nebo angličtině.

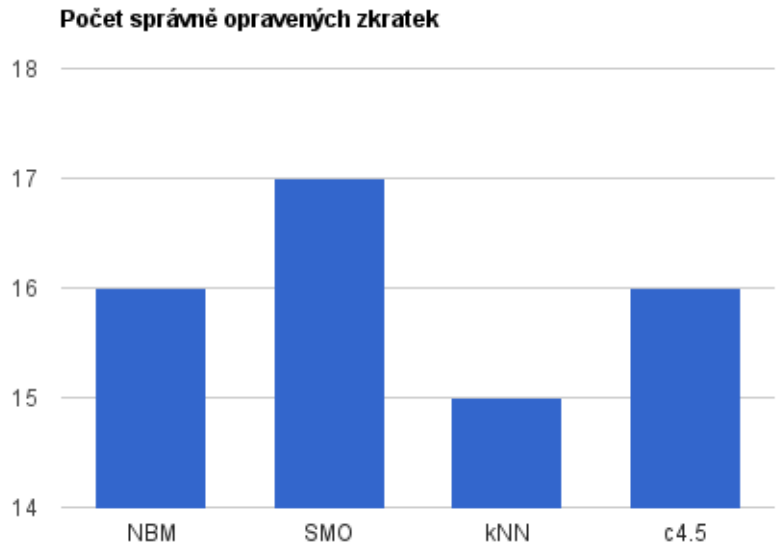
Je důležité poznamenat, že dosažené výsledky, pouze použitím fulltextového vyhledávání bez využití jakéhokoliv dataminingu, plní celkem dobře zadaný úkol a umožňují lékaři manuálně volit opravy zkratek z navrhovaných oprav přes uživatelské rozhraní. Tato část práce je zaměřená na porovnávání dataminingových algoritmů, používané k získávání přesnějších výsledků (návrhu) oprav. Datamining plugin pracuje s kontextem kolem zkratek, který slouží jako testovací data pro používané algoritmy a od toho se odvíjí výsledky dataminingových návrhů oprav. Výhodou tohoto způsobu korekce textu je zpětné učení a trénování existující množiny dat. Tím se aplikace může neustále zlepšovat a zpřesňovat.

Pro účely porovnání jednotlivých algoritmů jsem sestavil program, testující základní scenář opravy vybrané lékařské zprávy. Program byl sestavený ze 4 základních nezávislých podprogramů pro každý algoritmus. Výsledné hodnoty jsem zapisoval do souborů. Každý algoritmus vygeneroval 23 souborů, 1 pro každou zkratku s vyhodnocením proběhlé klasifikace a výpisem kriteriálních chyb. Tyto hodnoty jsem též využíval v této kapitole ke zhodnocení jednotkových výsledků jednotlivých algoritmů.

V příloze A, v tabulce 12.1 na straně 99 jsou výsledky klasifikace zkratek dle jednotlivých algoritmů. Napsal jsem 4 testovací programy, dle 4 vybraných algoritmů, na kterých jsem ověřil pravdivost klasifikovaných výsledků. Ve výsledcích jsem hodnotil zejména přesnost tvorby dataminingových modelů a pravdivost klasifikovaných zkratek.

9.3.2 Celkové výsledky

Podle dat naměřených testovacím programem, jsem dospěl k tomu, že nejvíce přesných klasifikaci vyhodnotil algoritmus SMO, nebo-li SVM s optimalizací, využívající polynomické jádro a metodu porovnávání 1 vs. 1. Je zřejmé, že algoritmus správně zklasifikoval 17 vzorků, dokud Multinomiální Naivní Bayes a prořezávané stromy C4.5 dokázaly zklasifikovat správně pouze 16. Nejhůře se představil algoritmus k-nejbližších sousedů, který dokázal správně zklasifikovat pouze 15 vzorků. Na následujícím obrázku je vidět počet úspěšně zklasifikovaných zkratek jednotlivých algoritmů:



Obrázek 9.1: Porovnávání správných klasifikací jednotlivých algoritmů

První dojem, kde je zřejmé, že žádný algoritmus nedokázal zklasifikovat správně úplně všechny zkratky. To je dáno v jejích naivních klasifikačních předpokladech, které jsou založené na statistických metodách. Hlavním jádrem správné klasifikace jsou trénovací data. Pro většinu zpráv jsem měl dostatečné množství trénovacích dat, nicméně některé algoritmy vyžadovaly ještě větší v řádě stovek či tisíců trénovacích vzorků pro konkrétní třídu. Dále je možné spatřit, že nejnižší chybovost dle všech 4 kriterií udává algoritmus Multinomiální Naivní Bayes, který si dokáže vystačit s velmi malým množstvím trénovacích dat. To byl podstata jeho největší přesnosti dle měřených odchylek. Jak je zřejmé, to však nestačí k tomu, aby byl nejdokonalejší ze všech.

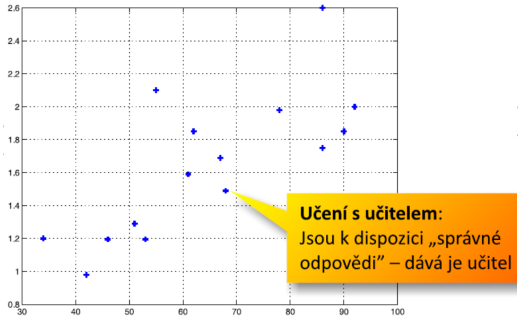
Hlavní příčinou nesprávně zklasifikovaných vzorků dat je nedostatečně kvalitně natrénovaná množina trénovacích dat. Příkladem jsou zkratky P1 a A1. Jedná se o velmi specifické termíny v rentgenologii. Pro tyto termíny jsem nenašel žádné informace ani data, tím fulltextový vyhledávač napomohl trénovacím algoritmům k vytvoření návrhu modelu z odvozených slov "pravé"nebo "Pars sfenoidalis". To prokazuje, že řešení funguje, ale je silně závislé na datech.

Další věc, kterou je možno postřehnout je, že některé odchylky jsou zcela nulové. Například u algoritmu SMO - zkratka "bilat."-> "bilaterálně". Je to dáno tím, že v trénovacích datech byla s dokonalou přesností obsažená data, která byla identická s testovacími daty. Jinými slovy, tento vzorek byl natrénovaný a posléze zklasifikovaný stejnými daty. Tím je dosažená dokonalá přesnost výsledků. V trénovacích datech se nenacházely žadné jiné eventuální klasifikační třídy, než ty které přesně vyhledáváme. V praxi se jedná o situaci, která často nenastává, ale v počátcích, kdy není dostatek učících dat pro tvorbu dokonalé hypotézy modelu, takováto situace nastat může.

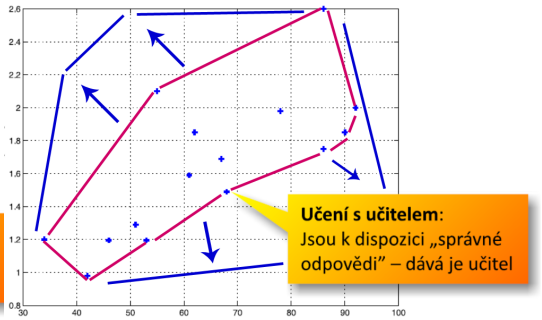
10 Zhodnocení

Úspěšnost jednotlivých algoritmů byla testovaná na dodaných datech. Jedna zpráva byla vybrána jako primární zdroj k testování. Obecně nelze říct, který algoritmus je nejlepší, protože všechny algoritmy prokázaly poměrně stejné výsledky s malými rozdíly. Jako nejpřesnější z vybraných algoritmů s optimálním nastavením v původní konfiguraci byl algoritmus SVM SMO.

To jak bude algoritmus predikovat je silně závislé na trénovacích datech. V tomto projektu jsem pracoval s odbornými termíny a zkratkami, kterým rozumí primárně specialisté z oboru medicíny v specializaci Radiologie. Ne každý atestovaný lékař zná všechny zkratky v medicíně, obvzlášť jedná-li se o velmi úzkou a specifickou doménu medicíny. To bylo důvodem, proč jsem nebyl schopný natrénovat v této práci všechny možné zkratky. Pro účely dokonalého fungování algoritmů je potřeba sbírat zpětnou vazbu a učit algoritmy, pomocí trénovacích dat správně a kvalitně. Pro lepší přehled jsem vyjádřil úplnost trénovacích dat porovnáváním následujících obázků:



Obrázek 10.1: Trénovací data [1]



Obrázek 10.2: Maximalizace úplnosti trénovacích dat

Toto porovnávání nám udává žádoucí spektrum trénovacích dat. Vizualizace na prvním obrázku ukazuje skutečnost, dokud obrázek napravo znázorňuje perfektní realitu. Ta je ovšem těžko dosažitelná, protože je potřeba mít všechny medicínské zkratky v češtině, to zahrnuje i data z jiných zdra-

votnických zařízení. Osa X grafů vyjadřuje instance testovacích dat, klasicky se jedná o medicínské zkratky, které je potřeba zařadit do dané kategorie. Kategorie jsou vyjádřené osou Y v grafech. Názorně je vidět, že čím širší je spektrum dat, tím je lépe natrénovaný klasifikátor. Úplnost modelu je klíčová pro lepší výsledky. [1]

10.1 Dosažené výsledky

Výsledná aplikace pro korekci lékařských textů byla navrhnutá jako modulové řešení pro textový editor CKEditor v Javascriptu. Ukázky vizualizace oprav a korekce textů je možné prohlédnou v příloze A, na obrázku 12.3 a 12.4. Na příkladech je vidět fungování aplikace v praxi. Frontendové řešení umožňuje ukládat data do slovníků standardizovaných slov. Zároveň umožňuje uživateli učit trénovací modely přesnějšími výsledky, přidáváním slov a zkratek s kontextem do databáze. Tato slova jsou přidávána do trénovací databáze pro tvorbu dataminingových modelů. Aplikace je jednoduchá na ovládání, rychlost databáze CouchDB s Apache Lucene a Redis jsou v produkčním prostředí dobrou kombinací pro reálný dlouhodobý provoz aplikace.

Na základě analýzy fulltextových databází jsem dospěl k závěru, že řešení je nasaditelné ve své základní podobě bez použití datamining pluginu na jakékoliv modernější open-source databázi. Nicméně výsledky testů ukazují, že vybraná kombinace byla nejvhodnější a je taktéž vhodná pro reálné použití. Plná integrace CouchDB s Apache Lucene by měla přijit s verzí CouchDB 2.0.0, kde Lucene bude plnou součástí a momentálně je vývojařská BETA verze, proto jsem zvolil stable verzi 1.6.1 s integraci od Lucene. Tím jsem byl schopný dosáhnout velmi rychlé čtení a prohledávání, pomocí fulltextových dotazů a možnosti od Lucene.

10.2 Úspěšnost jednotlivých algoritmů

Kriteriem úspěšnosti po otestování a celkovém zhodnocení byly po celou dobu 2 hlavní faktory - přesnost a úplnost.

Do velké míry je úplnost závislá na trénovacích datech. Nynější řešení je univerzální a lze ho pouze vylepšovat například o automatické učení na základě napojení na FN Plzeň a postupné rozšiřování slovníků lékařských textů. Dále tato aplikace slouží jako podklad k výzkumným pracím týmu MRE KIV. Porovnávané algoritmy fungují naprosto odlišným způsobem, nicméně se skoro všechny z nich dopracovaly k podobným výsledkům.

- SVM SMO - Algoritmus se prokázal jako nejlepší pro účely klasifikace textu. Nicméně jeho hodnoty odchylek byly nezanedbatelné, ale i přesto je vhodný k řešení zadaného problému. Algoritmus vykázal nejlepší přesnost.

- Implementace algoritmu C4.5 - Algoritmus je velice rychlý, na úkor toho je velmi citlivý při výběru prahových atributů. Z toho lze vydedukovat, že je více náklonný k chybám a nepřesným výsledkům. Tím je i přílíš závislý na trénovacích datech. V porovnáváním vykazoval dobré výsledky stejně jako Naivní Bayes. Algoritmus vykázal dobrou přesnost a minimální chybovost modelu.

- Metoda nejbližších sousedů funguje dobře, ale není vhodná pro lékařské texty z důvodu velké přítomnosti hlučných příznaků dat, které mohou v určitých případech negativně ovlivnit celkové výsledky. Například: "Krevní obraz: B-Le: 13,50 B-Ery: 4,83 B-Hb: 157 B-HTK: 0,463 B-Obj ery.: 96 B-Hb ery: 32,5 B-Hb konc: 338 B-Erytr.křivka: ".

  Zde zkratky jako B-Hb, B-Hb ery., B-Hb konc. mají vždy jiný význam a uvažování algoritmu by vykazovalo nepřesné výsledky. V práci jsem

ponechal základní optimální nastavení tohoto algoritmu, které při testování vykazovalo nejméně chyb a přesto algoritmus vyzákal nejhorší přesnost.

- Naivní Multinomiální Bayes prokázal dobré výsledky v celkovém porovnání. Jednoduchá metoda uvažování rovnoměrné distribuce funguje úspěšně a to s velmi dobrými výsledky. Kvalitní natrénování modelu s dostupnými daty by dopomohlo k jeho zdokonalení. Přesnost nebyla nejlepší, chybovost byla nejmenší.

Další možnosti pro aplikaci dataminingových algoritmů nad lékařskými texty v českém jazyce můžou být kombinace několik algoritmů například Bayes a SVM, případně zahrnout metody Itemsets a N-gramy, které je možno znovu zkombinovat a otestovat nad dostupnými daty. Další vylepšení tohoto projektu lze dosáhnout lepším předzpracováním výsledků. Například implementací pokročilejší filtrace dat před trénováním, ale na úkor delšího časového zpracování či celkové výkonnosti nynějšího řešení.

V rámci tohoto projektu jsem naimplementoval metodu pro filtraci dat, která ignoruje zkratky, které nezná. Důvodem je snížít riziko špatné klasifikace jednotlivých nalezených zkratek. Tato metoda je vhodná pro úplnost textů. Medicínské texty jsou odborné a jakákoliv chyba v reálném životě může zasáhnout zdraví a život daného pacienta.

11 Závěr

Náplní této práce bylo navrhnout, implementovat a otestovat řešení pro korekce medicínských semistrukturovaných dat, pomocí datamining metod. Metody dataminingu jsem použil pro přesnější navrhování oprav při korekci lékařských zkratek a medicínských termínů. Ze všech testovaných metod jsem zhodnotil a vybral tu nejefektivnější pro účely této diplomové práci.

Zjistil jsem, že trénovací data jsou základem dobré klasifikace. Toto byl důvod poměrně podobných výsledků jednotlivých algoritmů. Výrazné rozdíly byly časové odezvy zpracování dat, zejména na úrovni fulltextových databází. Jemnější rozdíly byly patrné v chybovosti a přesnosti jednotlivých algoritmů.

Výstupem této práce je program umožňující korekci lékařských textů v prostředí CKEditor ve formě modulů. Na základě dostupných dat jsem dosáhnul nejpřesnější a nejúplnější výsledky s algoritmem Support Vector Machines, založený na optimalizaci SMO. V nastavení algoritmu byl použitý model polynomického jádra a metoda porovnávání tříd 1 vs 1. Též bych tento algoritmus doporučoval jako nejlepší pro zpracování textů. Dalším algoritmem, který prokazoval značně dobré výsledky byl Miltinomiální Naivní Bayes.

Nelze jednoznačně říct, který dataminingový algoritmus je nejlepší pro řešení zadaného problému. Existuje skupina klasifikačních dataminingových metod, které plní účely této práce. Na základě testů a porovnávání jsem zvolil ty nejpřesnější z nich.

Základní řešení aplikace, založeno pouze na fulltextových prostředcích je taktéž vhodné pro reálné použití a umožňuje navrhování oprav chyb v textu. To je založeno na fulltextovém vyhledávání zástupných znaků a tzv. fuzzy search. Na základě zátěžových testů jsem zvolil CouchDB s integrací Apache Lucene pro rychlé vyhledávání.

Literatura

[1] Ing. EKŠTEIN, Kamil Ph.D. — přednášky Teorie Kognitivních systémů. [online] Použito na str. 80, 81, 90

[2] Ing. ŤOUPAL, Tomáš Ph.D. — přednášky Modely řízení ve firmě. [online] Použito na str. 73

[3] Hugo J, Vokurka M. Velký lékařský slovník 7. vydání, 2008. , ISBN: 978-80-7345-130-1, EAN: 9788073451301 (2008). Použito na str. 13

[4] Internetová jazyková příručka. Zpracovatel: mpra © 2008–2016 Jazyková poradna ÚJČ AV ČR, v. v. i - Zkratky čistě grafické . Dostupné na internetu: `http://prirucka.ujc.cas.cz/?id=780` . Použito na str. 14

[5] Kopecký, M. — Dokumentografické Informační Systémy [online]. Slidy k přednášce. Dostupné na internetu: `http://www.ms.mff.cuni.cz/~kopecky/vyuka/dis/`. Použito na str. 24, 28

[6] Pánek, K. — Architektury a modely webových strojů [online]. Lupa.cz . Dostupné na internetu: `http://www.lupa.cz/clanky/architektury-a-modely-webovych-stroju/`. Použito na str. 24

[7] Oracle. [online]. Full-Text Searches with Query Expansion. Dostupný z: `http://dev.mysql.com/doc/refman/5.7/en/fulltext-search.html`. Použito na str. 29

[8] Ranks NL company 2014. [online]. Stop words in Czech language. Dostupný z: `http://www.ranks.nl/stopwords/czech`. Použito na str. 30

[9] Oracle. [online]. Full-Text Searches with Query Expansion. Dostupný z: `https://dev.mysql.com/doc/refman/5.7/en/fulltext-query-expansion.html`. Použito na str. 31

[10] PostgreSQL. [online]. Chapter 12. Full Text Search. Dostupný z: https://www.postgresql.org/docs/8.3/static/textsearch.html. Použito na str. 32

[11] PostgreSQL Documentation. [online]. GiST and GIN Index Types: Dostupné z URL http://www.postgresql.org/docs/9.1/static/textsearch-indexes.html. Použito na str. 33

[12] Apache Lucene Documentation. [online]. Lucene Information Retrieval and VSM model. Dostupné z URL https://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/search/Similarity.html. Použito na str. 41

[13] KUO, Cheng-Ju; LING, Maurice HT; LIN, Kuan-Ting and HSU, Chun-Nan, "BIOADI: a machine learning approach to identifying abbreviations and definitions in biological literature", [online] DOI:10.1186/1471-2105-10-S15-S7, Online ISSN 1471-2105 (2009). Použito na str. 18

[14] USAMA Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, From Data Mining to Knowledge Discovery in Databases. [Online] DOI: http://dx.doi.org/10.1609/aimag.v17i3.1230. Použito na str. 48

[15] SOUMEN Chakrabarti, Martin Ester, Usama Fayyad, Johannes Gehrke, Jiawei Han, Shinichi Morishita, Gregory Piatetsky-Shapiro, Wei Wang, Data Mining Curriculum: A Proposal (Version 1.0), [Online] URL: http://www.kdd.org/curriculum/index.html April 30, 2006. Použito na str. 48

[16] EIBE, Frank; BOUCKAERT, Remco R. , Naive Bayes for Text Classification with Unbalanced Classes, Computer Science Department, University of Waikato, New Zealand. ISBN:3-540-45374-1 978-3-540-45374-1 DOI:10.1007/11871637 49 (2013). Použito na str. 51

[17] RAGHAVAN, Prabhakar - Text Classification : The Naive Bayes algorithm - Adapted from Lectures by Prabhakar RAGHAVAN (Yahoo and Stanford) and Christopher Manning (Stanford), Stanford University (2013). [online] URL http://cecs.wright.edu/~tkprasad/courses/cs707/L13NaiveBayesClassify.ppt Použito na str. 51, 53

[18] BARBER, David - Bayesian Reasoning and Machine Learning, [online] URL http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/240415.pdf ISSN: 0163-5700 DOI:10.1145/2636805.2636813 (2008). Použito na str. 51

[19] PLATT, John - Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, ISBN: 0-262-19416-3 (1998). Použito na str. 55

[20] CHANG, Chih-Chung; LIN, Chih-Jen. "LIBSVM: A library for support vector machines". ACM Transactions on Intelligent Systems and Technology (2011). DOI:10.1145/1961189.1961199 Použito na str. 55

[21] ZANNI, Luca. Parallel Software for Training Large Scale Support Vector Machines on Multiprocessor Systems (2006). ISSN: 1532-4435 EISSN: 1533-7928. Použito na str. 55

[22] doc. Ing. ŽIŽKA, Jan CSc - Support vector machines (SVM): Algoritmy podpůrných vektorů [online]. posl. revize 9. 12. 2004 [cit. 2012–04–25]. Vyňatek z přednášek http://is.muni.cz/el/1433/podzim2006/PA034/09_SVM.pdf. Použito na str. 55, 57

[23] RIFKIN, Ryan - "Everything Old is New Again: a Fresh Look at Historical Approaches in Machine Learning", Ph.D. thesis (2002). Použito na str. 55, 56

[24] QUINLAN, J. R., C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, (1993). ISBN 1-55860-238-0. Použito na str. 58, 59

[25] QUINLAN, J. R., Improved use of continuous attributes in c4.5. Journal of Artificial Intelligence Research, 4:77-90, ISSN 1076 - 9757. (1996). Použito na str. 59

[26] PATERA, Jan - Rozhodovací stromy. Brno: FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ ÚSTAV AUTOMATIZACE A MĚŘÍCÍ TECHNIKY, Diplomová práce (2008). Použito na str. 58

[27] ALTMAN, N. S., "An introduction to kernel and nearest-neighbor nonparametric regression". The American Statistician. ISSN 0003-1305 (Print) (1992). 60, 61

[28] COOMANS, D.; MASSART, D.L., "Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-Nearest neighbour classification by using alternative voting rules". DOI:10.1016/S0003-2670(01)95359-0 (1982). Použito na str. 60, 61

[29] Nigsch F, Bender A, van Buuren B, Tissen J, Nigsch E, Mitchell JB. "Melting point prediction employing k-nearest neighbor algorithms and genetic parameter optimization". Journal of Chemical Information and Modeling. Print Edition ISSN: 1549-9596. Web Edition ISSN: 1549-960X (2006). Použito na str. 60

[30] Weka [online]. [cit. 2012–04–11]. URL `http://weka.wikispaces.com`. Použito na str. 64, 65

[31] Mohd Fauzi bin Othman,Thomas Moh Shan Yau - "Comparison of Different Classification Techniques", Control and Instrumentation Department, Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Skudai, Malaysia. ISSN: 2231-2307 (2007). Použito na str. 59

[32] J. Scott Armstrong and Fred Collopy. "Error Measures For Generalizing About Forecasting Methods: Empirical Comparisons"(PDF). Internati-

onal Journal of Forecasting 8 (1). DOI:10.1016/0169-2070(92)90008-w. (1992). Použito na str. 73

Seznam obrázků a příkladů

2.1 Histogram 11

4.1 Vektorový model fulltextu 26

5.1 Map Reduce ukázka 38

7.1 Ukázka trénovacích dat ve formátu .arff 52
7.2 SVM 55
7.3 Polykernel 57
7.4 Příklad k-NN 62

9.1 Porovnání algoritmů 78

10.1 Trénovací data [1] 80
10.2 Maximalizace úplnosti trénovacích dat 80

12.1 Testovací lékařská zpráva 92
12.2 Rozhodovací strom algoritmu J48 93
12.3 Neopravená lékařská zpráva 94
12.4 Opravená lékařská zpráva 94
12.5 High level architecture 95

Seznam tabulek

5.1 Scénář 1 - počet dokumentů obdržených při paralelním čtení po dobu 30 vteřin. Zkratka ř. znamená řádků, doc - JSON dokumentů 44

5.2 Scénář 2 - počet dokumentů obdržených při paralelním čtení a zápisu po dobu 30 vteřin.. Zkratka ř. znamená řádků, doc - JSON dokumentů 45

5.3 Scénář 3 - doba k obdržení [ms.] 4x 1000 dokumentů při paralelním čtení 4 vláknami. 46

7.1 Metriky pro nalezení k nejbližších sousedů 61

12.1 Výsledky měření klasifikací zkratek jednotlivých algoritmů. . 99

12 Příloha A

V této příloze se nachází grafy, tabulky, ukázky a příklady

Bez intrakraniální hemorhagie, klínovitá hypodenze vel. 25x20mm na rozhraní povodí ACM a ACA vpravo, spíše starší ischemie. Hypodenze vel. 35x30mm v levé mozečkové hemisféře, starší postmalatické ložisko. Středočarové struktury bez přesunu, komorový systém a SA prostory mírně rozšířené při atrofii mozku a mozečku. VDN v zachyceném rozsahu, mastoidální sklípky a středouší bilat. bez patol. obsahu.

CT perfúze: provedeno po podání kontrastu i.v.

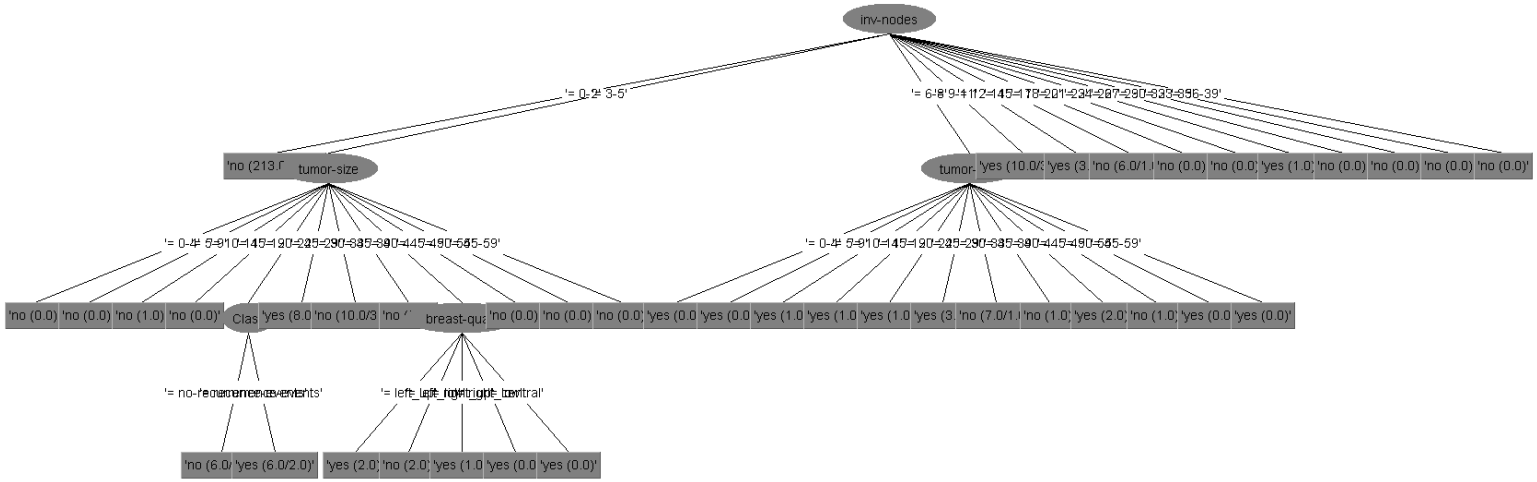
Prodloužený TTP a CBF vpravo v povodí ACM, se zachováním CBV, tedy bez vytvořeného ložiska nekrozy, kompletní výpadek perfuze pouze v místech nativně popsaných hypodenzit.

CTA krčních tepen a mozku: Provedeno po podání kontrastní látky i.v.

Ateroskleroza oblouku aorty. Odstupy z aortálního oblouku jsou volné v běžném uspořádání. Vinutý průběh ACC bilat. Výrazně vinutý průběh ACI bilat. , vlevo volná, vpravo v odstupu ateromatozní plát nevedoucí k významné stenoze. Intrakraniálně subtotální uzávěr M1 vpravo (v délce cca 6 mm), periferně se plní cévní řečiště v povodí ACM vpravo gracilně. Aplazie A1 a ACoP vpravo, aplazie P1 vlevo. AV bilat. volné, levá za odstupem s coilingem. Pravá je pod bazí gracilní po odstupu PICA. Zvětšený pravý lalok štítní žlázy s nehomogenním uzlem vel. 301x23mm.

Závěr: Akutní ischemie bez vytvořeného ložiska nekrozy vpravo v povodí ACM při jejím subtotálním uzávěru v M1.

Příklad 12.1: Text zvolené testovací lékařské zprávy.



Obrázek 12.2: Vygenerovaný rozhodovací strom

Bez intrakraniální hemorhagie, klínovitá hypodenze vel. 25x20mm na rozhraní povodí ACM a ACA vpravo, spíše starší ischemie. Hypodenze vel. 35x30mm v levé mozečkové hemisféře, starší postmalatické ložisko. Středočarové struktury bez přesunu, komorový systém a SA prostory mírně rozšířené při atrofii mozku a mozečku. VDN v zachyceném rozsahu, mastoidální sklípky a středouší bilat. bez patol. obsahu.

CT perfúze:
provedeno po podání kontrastu i.v.

Prodloužený TTP a CBF vpravo v povodí ACM, se zachováním CBV, tedy bez vytvořeného ložiska nekrozy, kompletní výpadek perfuze pouze v místech nativně popsaných hypodenzit.

CTA krčních tepen a mozku:
Provedeno po podání kontrastní látky i.v.

Ateroskleroza oblouku aorty. Odstupy z aortálního oblouku jsou volné v běžném uspořádání. Vinutý průběh ACC bilat. Výrazně vinutý průběh ACI bilat., vlevo volná, vpravo v odstupu ateromatozní plát nevedoucí k významné stenoze.
Intrakraniálně subtotální uzávěr M1 vpravo (v délce cca 6 mm), periferně se plní cévní řečiště v povodí ACM vpravo gracilně. Aplazie A1 a ACoP vpravo, aplazie P1 vlevo. AV bilat. volné, levá za odstupem s coilingem. Pravá je pod bazí gracilní po odstupu PICA. Zvětšený pravý lalok štítní žlázy s nehomogenním uzlem vel. 301x23mm.

Závěr: Akutní ischemie bez vytvořeného ložiska nekrozy vpravo v povodí ACM při jejím subtotálním uzávěru v M1.

Obrázek 12.3: Neopravená lékařská zpráva

Bez intrakraniální hemorhagie, klínovitá hypodenze ~~vel.~~ velikosti 25x20mm na rozhraní povodí ~~ACM~~ arteria cerebri media a ~~ACA~~ Arteria cerebri anterior vpravo, spíše starší ischemie. Hypodenze ~~vel.~~ velikosti 35x30mm v levé mozečkové hemisféře, starší postmalatické ložisko. Středočarové struktury bez přesunu, komorový systém a ~~SA~~ subarachnoidální prostory mírně rozšířené při atrofii mozku a mozečku. ~~VDN~~ vedlejší dutiny nosní v zachyceném rozsahu, mastoidální sklípky a středouší ~~bilat.~~ oboustranně bez ~~patol.~~ patologického obsahu.

~~CT~~ počítačová tomografická perfúze:
provedeno po podání kontrastu ~~i.v.~~ intravenozní

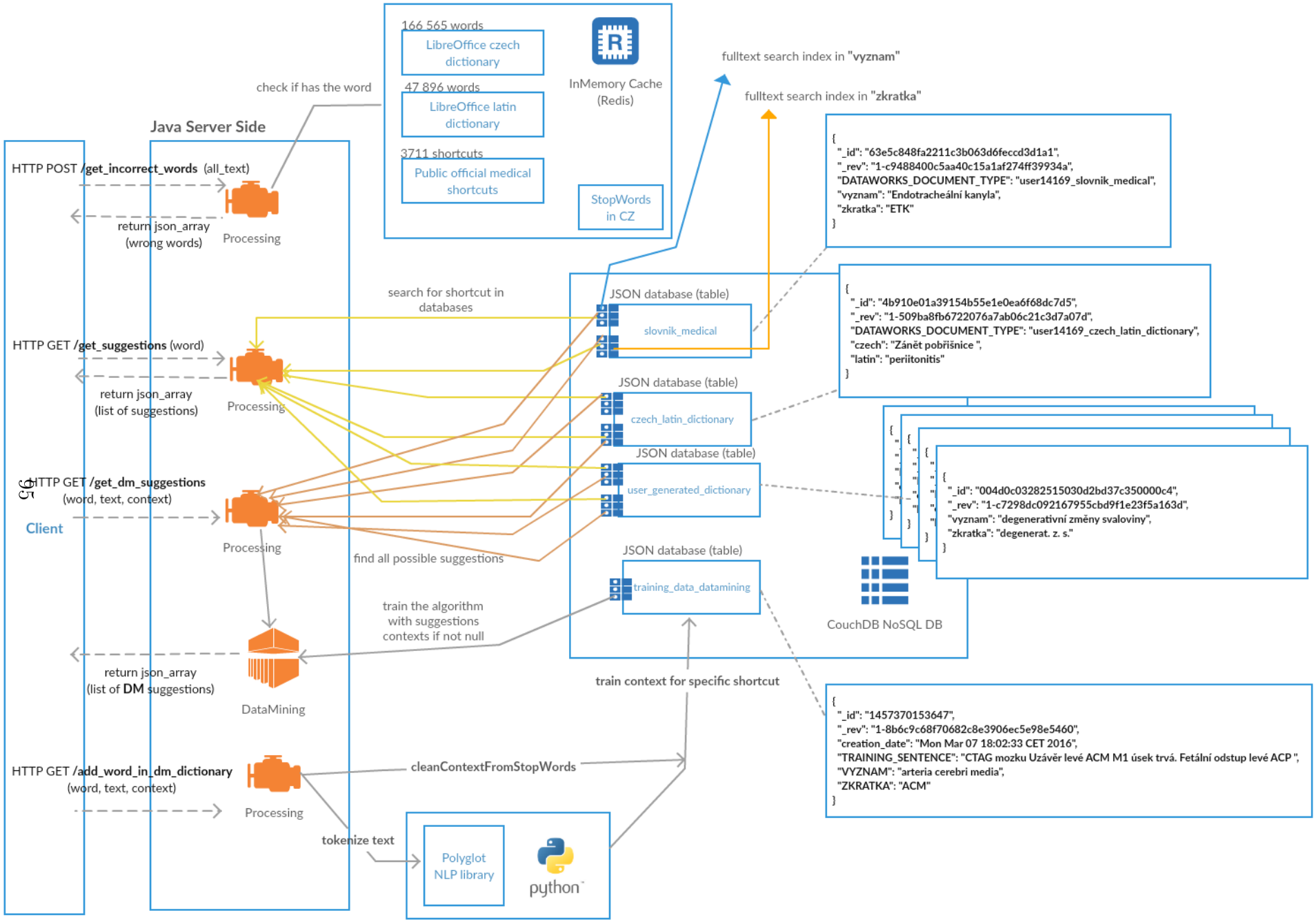
Prodloužený ~~TTP~~ Time to Peak a ~~CBF~~ Cerebral blood flow vpravo v povodí ~~ACM~~ arteria cerebri media, se zachováním ~~CBV~~ cerebral blood volume, tedy bez vytvořeného ložiska nekrozy, kompletní výpadek perfuze pouze v místech nativně popsaných hypodenzit.

~~CTA CT~~ počítačová tomografická Angiografie krčních tepen a mozku:
Provedeno po podání kontrastní látky ~~i.v.~~ intravenozní

Ateroskleroza oblouku aorty. Odstupy z aortálního oblouku jsou volné v běžném uspořádání. Vinutý průběh ~~ACC~~ arteria carotis communis ~~bilat.~~ oboustranně Výrazně vinutý průběh ~~ACI~~ arteria carotis interna ~~bilat.~~ oboustranně, vlevo volná, vpravo v odstupu ateromatozní plát nevedoucí k významné stenoze.
Intrakraniálně subtotální uzávěr ~~M1~~ pars sfenoidalis vpravo (v délce cca 6 mm), periferně se plní cévní řečiště v povodí ~~ACM~~ arteria cerebri media vpravo gracilně. Aplazie ~~A1~~ 1. krční obratel a ~~ACoP~~ Arteria communicans posterior vpravo, aplazie ~~P1~~ pravé vlevo. AV ~~bilat.~~ oboustranně volné, levá za odstupem s coilingem. Pravá je pod bazí gracilní po odstupu ~~PICA.~~ dolní zadní mozečková tepna Zvětšený pravý lalok štítní žlázy s nehomogenním uzlem ~~vel.~~ velikosti 301x23mm.

Závěr: Akutní ischemie bez vytvořeného ložiska nekrozy vpravo v povodí ~~ACM~~ arteria cerebri media při jejím subtotálním uzávěru v ~~M1~~ pars sfenoidalis.

Obrázek 12.4: Opravená lékařská zpráva



Obrázek 12.5: Vysokoúrovňová architektura

ID	Správně v praxi	Hledaná zkratka	Algoritmus	MAE	RAE	RMSE	RRSE	Predikce algoritmu	Pravdivost
1	pars sfenoidalis	M1.	NBM	0.0002	0.0552	0.0004	0.0886	malém oběhu	ne
			SMO	0.25	72.7273	0.3118	76.98	malém oběhu	ne
			kNN	0.1875	54.5455	0.2165	53.4522	malém oběhu	ne
			C4.5	0.125	36.3636	0.25	61.7213	množství	ne
2	Time to peak	TTP	NBM	0.0015	0.547	0.003	0.8258	rychlost cirkulace (time to peak)	ano
			SMO	0.2222	81.4815	0.3103	84.4617	rychlost cirkulace (time to peak)	ano
			kNN	0.1515	55.5556	0.2033	55.3283	rychlost cirkulace (time to peak)	ano
			C4.5	0.2	73.3333	0.3162	86.0707	rychlost cirkulace (time to peak)	ano
3	patologického	patol.	NBM	0.0015	0.374	0.0018	0.4237	patologického	ano
			SMO	0.2222	55.5556	0.2722	62.9941	patologického	ano
			kNN	0.2667	66.6667	0.2828	65.4654	patologického	ano
			C4.5	0.3333	83.3333	0.4082	94.4911	patologického	ano
4	arteria carrotis communis	ACC	NBM	0.0034	1.7602	0.014	4.4988	Arteria carrotis interna	ne
			SMO	0.1728	89.0228	0.2815	90.7097	Arteria carrotis interna	ne
			kNN	0.0936	48.1928	0.1489	47.977	Arteria carrotis interna	ne
			C4.5	0.0963	49.5984	0.2194	70.7182	Arteria carrotis interna	ne
5	vedlejší dutiny nosní	VDN	NBM	0.0006	0.2775	0.0012	0.3587	vedlejší dutiny nosní	ano
			SMO	0.2041	87.4636	0.301	89.1712	vedlejší dutiny nosní	ano
			kNN	0.1143	48.9796	0.1633	48.3718	vedlejší dutiny nosní	ano
			C4.5	0.125	53.5714	0.25	74.0539	vedlejší dutiny nosní	ano
6	arteria carrotis interna	ACI	NBM	0.0034	1.7602	0.014	4.4988	Arteria carrotis interna	ano
			SMO	0.1728	89.0228	0.2815	90.7097	Arteria carrotis interna	ano
			kNN	0.0936	48.1928	0.1489	47.977	Arteria carrotis interna	ano
			C4.5	0.0963	49.5984	0.2194	70.7182	Arteria carrotis interna	ano
7	arteria carrotis media	ACM	NBM	0.0018	0.7882	0.0071	2.1097	Arteria cerebri media	ano
			SMO	0.2041	88.3929	0.3012	89.7059	Arteria cerebri media	ano
			kNN	0.0952	41.25	0.1361	40.5334	Arteria cerebri media	ano
			C4.5	0.0606	26.25	0.1741	51.8505	Arteria cerebri media	ano
8	?	P1	NBM	0.0009	0.2264	0.0018	0.4265	pravé	ne

			SMO	0.2222	58.3333	0.2722	64.715	pravé	ne
			kNN	0.1905	50	0.202	48.0384	Pars sfenoidalis úseky M1-M2	ne
			C4.5	0	0	0	0	Pars sfenoidalis úseky M1-M2	ne
9	arteria carrotis anterior	ACA	NBM	0.0049	2.7264	0.0159	5.3558	Arteria carrotis interna	ne
			SMO	0.16	89.8876	0.2722	91.4874	Arteria carrotis interna	ne
			kNN	0.09	50.5618	0.15	50.4219	Arteria carrotis interna	ne
			C4.5	0.1	56.1798	0.2236	75.1646	Arteria carrotis anterior	ano
10	zadní dolní mozečková tepna	PICA.	NBM	0.0056	2.0566	0.0176	4.8007	zadní dolní mozečková tepna	ano
			SMO	0.2222	81.4815	0.3103	84.4617	zadní dolní mozečková tepna	ano
			kNN	0.1515	55.5556	0.2033	55.3283	zadní dolní mozečková tepna	ano
			C4.5	0.2	73.3333	0.3162	86.0707	zadní dolní mozečková tepna	ano
11	?	A1	NBM	0.0031	1.1709	0.0112	3.0899	angiografie	ne
			SMO	0.2222	83.105	0.3103	85.6377	angiografie	ne
			kNN	0.1282	47.9452	0.172	47.4681	angiografie	ne
			C4.5	0.1111	41.5525	0.2357	65.0465	Arterie	ne
12	intravenozně	i.v.	NBM	0.0246	17.241	0.0327	22.9232	intravenozně	ano
			SMO	0	0	0	0	intravenozně	ano
			kNN	0.1429	100	0.1429	100	intravenozně	ano
			C4.5	0	0	0	0	intravenozně	ano
13	CT angiografie	CTA	NBM	0.0165	7.8302	0.0554	17.2322	rychlost cirkulace	ne
			SMO	0.1875	88.9535	0.2912	90.5201	rychlost cirkulace	ne
			kNN	0.1029	48.8372	0.1556	48.3718	rychlost cirkulace	ne
			C4.5	0.125	59.3023	0.25	77.702	rychlost cirkulace	ne
14	milimetru	35x30mm	NBM	0.3333	100	0.3333	100	milimetru	ano
			SMO	0	0	0	0	milimetru	ano
			kNN	0.3333	100	0.3333	100	milimetru	ano
			C4.5	0	0	0	0	milimetru	ano
15	milimetru	25x20mm	NBM	0.3333	100	0.3333	100	milimetru	ano
			SMO	0	0	0	0	milimetru	ano
			kNN	0.3333	100	0.3333	100	milimetru	ano
			C4.5	0	0	0	0	milimetru	ano
16	milimetru	301x23mm.	NBM	0.3333	100	0.3333	100	milimetru	ano

			SMO	0	0	0	0	milimetru	ano
			kNN	0.3333	100	0.3333	100	milimetru	ano
			C4.5	0	0	0	0	milimetru	ano
17	pocitacova tomograficka angiografie	CT	NBM	0.0255	9.794	0.0734	20.6769	počítačová tomografická	ano
			SMO	0.2222	85.446	0.3103	87.4388	počítačová tomografická	ano
			kNN	0.1282	49.2958	0.172	48.4664	cavum douglasi	ne
			C4.5	0.1667	64.0845	0.2887	81.3408	počítačová tomografická	ano
18	bilaterálně (oboustranně)	bilat.	NBM	0.1994	79.774	0.1995	79.7912	bilaterálně	ano
			SMO	0	0	0	0	bilaterálně	ano
			kNN	0.25	100	0.25	100	bilaterálně	ano
			C4.5	0	0	0	0	bilaterálně	ano
19	arteria communicans posterior	ACoP	NBM	0.0038	2.1128	0.0138	4.6091	Arteria communis posterior	ano
			SMO	0.16	89.4118	0.2722	91.1295	Arteria communis posterior	ano
			kNN	0.0947	52.9412	0.1579	52.8681	Arteria communis posterior	ano
			C4.5	0.1111	62.0915	0.2357	78.9204	Arteria communis posterior	ano
20	velikosti	vel.	NBM	0.0006	0.2118	0.001	0.2551	velmi suspektní	ne
			SMO	0.24	82.5	0.3162	84.8668	velikosti	ano
			kNN	0.1455	50	0.1818	48.795	velmi suspektní	ne
			C4.5	0.1333	45.8333	0.2582	69.2935	vyšetření	ne
21	Central Blood Flow	CBF	NBM	0.001	0.3628	0.0029	0.7927	Mozkový krevní průtok (Cerebral	ano
			SMO	0.2222	82.7586	0.3103	85.4298	Mozkový krevní průtok (Cerebral	ano
			kNN	0.1389	51.7241	0.1863	51.2989	Mozkový krevní průtok (Cerebral	ano
			C4.5	0.1852	68.9655	0.3043	83.7708	Mozkový krevní průtok (Cerebral	ano
22	Central Blood Volume	CBV	NBM	0.0073	2.6538	0.0233	6.4836	Cerebral Blood Volume	ano
			SMO	0.24	86.8966	0.3162	88.0559	Cerebral Blood Volume	ano
			kNN	0.1333	48.2759	0.1667	46.4095	Cerebral Blood Volume	ano
			C4.5	0.1486	53.7931	0.2726	75.8947	Mozkový krevní průtok (Cerebral	ne
23	subarachnoidální	SA	NBM	0.0034	1.1076	0.0077	1.999	subarachnoidální	ano
			SMO	0.24	78.9474	0.3162	82.1995	subarachnoidální	ano
			kNN	0.16	52.6316	0.2	51.9875	subarachnoidální	ano
			C4.5	0.1867	61.4035	0.3055	79.4123	subarachnoidální	ano

Tabulka 12.1: Výsledky měření klasifikací zkratek jednotlivých algoritmů.

13 Příloha B

Obsah DVD:

Data

*.xls, .csv - datamining analýza, vstupní a předzpracovávaná data

*.pdf - Analýza slov a zkratek - Histogram

Tests

/Datamining algs testy - testy v javě, testující vybrané datamining algoritmy

/Fulltext dbs testy - testy v javě, testující základí scénáře NoSQL databází

Návod na používání testovacích programů

Import scripts

/Pg - DDL a DML skripty pro postgreSQL, včetně import dat

/CouchDB - DDL a DML bash skript pro CouchDB, včetně import dat

/MongoDB - DDL a DML bash skript pro MongoDB, včetně import dat

Prerekvizity

Java JDK, JRE, Tomcat a Eclipse - prerekvizity pro nasazení aplikace

Program

dep-jar - knihovny a závislosti, potřebné pro kompilace projektu

bin - zkompilované zdrojové kódy do spustitelné podoby

src - kompletní okomentovaný zdrojový kód programu

WebContent - Front-endová část navrženého řešení

mrekiv.war - zkomprimovaný soubor J2EE projektu

build.xml - soubour pro sestavení J2EE projektu

Text

tex - text diplomové práce ve formátu LaTeX

pdf - text diplomové práce ve formátu Adobe PDF