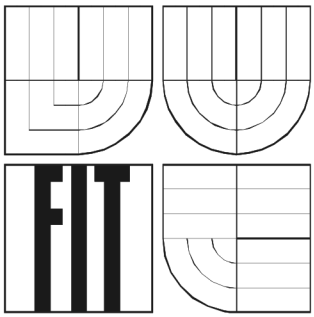
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Metody získávání znalostí z textových dat

Diplomová práce

2007 **Luděk Smékal**

Zde má přijít originál zadání

2

Metody získávání znalostí z textových dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D.
Další informace jsem čerpal z literatury a z konzultací u vedoucího diplomové práce.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Luděk Smékal
23.01.2007

3

Abstrakt

Tato diplomová práce pojednává o tzv. „dolování dat“, tedy získávání takových informací z databází, které nejsou přímo viditelné, ale které lze pomocí speciálních algoritmů z databází získat. Zaměřuje se na klasifikaci dokumentů do tříd v rámci digitální knihovny, vybranou metodou, která je založena na množinách položek tzv. „metoda itemsets“. Tato metoda rozšiřuje oblast aplikace Apriori algoritmu určeného původně ke zpracování transakčních databází a generování frekventovaných množin položek. Práce se rovněž zabývá možností modifikace vlastního Apriori algoritmu pro potřeby klasifikace metodou itemsets.

Klíčová slova

dolování dat, transakční databáze, množina položek, klasifikace, digitální knihovna, Apriori algoritmus, metoda itemsets, frekventovaná množina položek

Poděkování

Srdečně děkuji panu Ing. Vladimíru Bartíkovi, Ph.D., za jeho odbornou i pedagogickou pomoc při řešení projektu.

4

Abstract

This diploma thesis handles with so-called data mining. Data mining is about obtaining some data or informations from databases, where these data or informations are not directly visible, but they are accessible by using special algorithms. This diploma thesis mainly aims documents clasifying by selected method in scope of digital library. The selected method is based on sets of items called "itemsets method". This method extends Apriori algorithm application field originally designed for transaction databases processing and generation of sets of frequented items.

Keywords

data mining, transaction databases, itemset, classification, digital library, Apriori algorithm, method Itemsets, frequent Itemsets

5

Obsah

6

7

1 Úvod

Tato diplomová práce se zaměřuje na proces klasifikace dokumentů do tříd metodou itemsets [9]. Jejím cílem je podrobnější popis samotné metody, než jak je tomu v [9] a obdobných dokumentech, její implementace spolu s novou modifikací Apriori algoritmu [8], jež tato metoda využívá, a rovněž testování a zhodnocení výsledků na trénovací kolekci dokumentů Reuters-21578 [RC]. Jedná se tedy o srovnání vlivu nové modifikace této metody s jejím základním konceptem popsaným v [9].

Metoda itemsets vychází z Apriori algoritmu, který pracuje s tzv. transakční databází. Tato data představují předzpracované abstrakty dokumentů do databáze podobné transakční databázi, ze které lze Apriori algoritmem zjistit potřebné informace, nezbytné pro vlastní klasifikaci dokumentů do patřičných tříd, čímž vznikne tzv. digitální knihovna, jejíž tvorba se pro člověka (experta) v době velkých objemů textových dat s velkým množstvím nejrůznějších dokumentů a neustále se měnícím obsahem a průběžným doplňováním, s čímž souvisí i vznik úplně nových kategorií, stala prakticky nemožnou, neboť je tato činnost velice časově náročná.

V druhé kapitole je o problematice získávání znalostí z textových dat obecně pojednáno a dále jsou vysvětleny důležité pojmy potřebné pro pochopení dalšího textu.

Ve třetí kapitole je popis klasifikace jako takové a je zde vysvětlen pojem digitální knihovna.

Ve čtvrté kapitole je celkový popis metody itemsets včetně Apriori algoritmu [8] a jeho modifikace, který je v metodě itemsets použit pro generování vstupních dat, včetně demonstračního příkladu.

Pátá kapitola slouží jako dokumentace k programové části. Obsahuje také kapitolu sloužící jako manuál k programové části, která se nachází na přiloženém médiu.

V šesté kapitole jsou představeny výsledky experimentů aplikace programové části na kolekci dokumentů Reuters 21578 [RC]. Zde je také srovnání obou metod mezi sebou, tedy metody itemsets a dále modifikace této metody použitím proměnné minimální podpory.

V závěru jsou stručně shrnuty získané poznatky a možnosti dalšího výzkumu metody.

8

2 Dolování v textových datech

Dolování v textech (v textových databázích), známé pod zkratkou "text mining", definujeme jako proces získávání znalostí, který má za cíl identifikovat a analyzovat předem neznámé a potencionálně užitečné pro uživatele zajímavé, informace obsažené ve velkém množství textových dat. Takovými informacemi máme na mysli informace, které nejsou implicitně viditelné. Problémem při dolování v textových datech je fakt, že texty nejsou navrženy pro zpracování počítačem. Jsou velmi obsáhlé, mají velký šum, často se měnící obsah, mají špatnou strukturu a velmi složitou sémantiku textu. Také se v nich často objevují dvojsmysly a jsou psány v mnoha jazycích.

Dolování v textech chápeme jako činnost skládající se ze tří částí, a to předzpracování dokumentů (kapitola 2.1), dolování dat (kapitola 2.2) a prezentace získaných znalostí (kapitola 2.3).

2.1 Předzpracování dokumentů

Ještě před klasifikací je nutno textové dokumenty předzpracovat, tedy nalézt jednotlivé stavební prvky textu a oddělit informace podstatné pro další zpracování od údajů, jež nejsou důležité pro klasifikační metody a jejichž zařazení do klasifikačního procesu by vedlo k nepřesným výsledkům, tedy k nepřesnému zařazování dokumentů.

Takovéto přípravě dat pro klasifikační metody říkáme indexace. V kolekci dokumentů se vyhledají jednotlivá témata, dokumenty a slova. Vytváří se tzv. indexové soubory, které obsahují vazby mezi dokumenty a třídami a vazby mezi dokumenty a slovy způsobem, který je vhodný pro následné zpracování. Vlastní slova dokumentů se zpravidla v indexových souborech nevyskytují. Místo nich jsou většinou používány jejich jednoznačné identifikátory. Podobná situace je i v případě evidence dokumentů a jednotlivých tématických okruhů. Součástí výstupu indexace bývá také slovník tříd, jenž zvoleným identifikátorům přiřazuje skutečné názvy tříd. Indexaci dělíme na 3 fáze a nyní si je vysvětlíme v 2.1.1. až 2.1.3. .

2.1.1 Lexikální analýza

Lexikální analýza je prvním krokem předzpracování dokumentů. Jejím hlavním cílem je nalezení hranic mezi slovy ve vstupních datech.

Problém lexikální analýzy spočívá ve správném stanovení oddělovačů slov. Typicky jsou jako oddělovače zvolena interpunkční znaménka, mezery a znaky konce řádku. Dále je možnost přidat k oddělovačům i další nepísmenné znaky. Avšak může se stát, že narazíme na symboly, u kterých není zcela jednoznačná situace, zdali je lze považovat za oddělovače či nikoliv. S takovými problémy se setkáváme většinou v technické literatuře, kde máme problémy s interpretací číslic. Zde

9

se totiž objevují identifikátory, a ty číslice obsahují. Obvykle se respektují slova s číslicemi, která nezačínají číslicí.

Další problém představují spojovníky. Ty chápeme jako oddělovače různých slov, přestože někdy můžou být součástí názvů nebo indikací dělení slov na konci řádků.

Problém je také s rozlišováním velkých a malých písmen. Běžně se tento problém obchází převodem na stejnou velikost. Také je nutné hlídat způsob, jakým je zpracovávaný text kódován (například čeština a další složitější jazyky).

2.1.2 Odstraňování bezvýznamných slov

V textových datech se hojně vyskytují slova, která nemají k účelu indexace žádný význam. Jsou to slova, která nenesou téměř žádnou informaci a mají pouze gramatický význam. Tato slova tvoří seznam a ten je označován jako slovník nevýznamových slov. Slovník nevýznamových slov lze vytvářet jak ručně, tak i s použitím tzv. frekvenčního slovníku, jenž obsahuje určité procento nejčastěji se vyskytujících slov. Taková slova se totiž vyskytují ve většině zpracovávaných dokumentů. Jako perspektivní se ukázal způsob automatického vytvoření frekvenčního slovníku s následnou ruční korekcí jeho obsahu.

Odstranění nezvýznamových slov zrychluje a zpřesňuje klasifikaci, a také snižuje paměťové nároky na klasifikační algoritmus.

2.1.3 Lemmatizace

Pojmem lemmatizace rozumíme proces, při kterém převádíme morfologické tvary slov do jejich základního tvaru. Tento převod je pro klasifikaci velice výhodný, neboť oproti nejrůznějším morfologickým tvarům jsou slova v základním tvaru naprosto jednoznačně identifikovatelná, a tím pádem mnohem snadněji dohledatelná. Lemmatizaci lze realizovat dvěma základními přístupy. Konkrétně viz. 2.1.3.1 a 2.1.3.2.

2.1.3.1 Slovníkově orientovaný

Slovníkově orientovaná lemmatizace používá pro svůj chod velmi obsáhlé slovníky, které pro každé slovo v jeho základním tvaru definují (uvádějí) jeho všechny (pokud možno) podoby v různých tvarech. Tento přístup je značně časově náročný, a s tím jde ruku v ruce i finanční stránka, a používání takovéhoto slovníku není ani triviální záležitostí. Bohužel obecně je platné pravidlo, že čím je složitější jazyk a obsahuje více různých tvarů slov, tím je slovník pro tento jazyk rozsáhlejší. Výhodou je, že tento přístup je přesnější než algoritmický popsaný v 2.1.3.2.

2.1.3.2 Algoritmický

Oproti slovníkově orientovanému přístupu představuje algoritmický přístup mnohem levnější jednodušší a rychlejší variantu lemmatizace. Jeho hlavní nevýhodou ovšem je, že je oproti použití slovníků mnohem méně kvalitní.

Většinou se používá technika odtrhávání konců slov. Samozřejmě tu vyvstává problém jakým způsobem určit, co je tím správným koncem slova a co se má ve slově zachovat, to znamená co je koncovka a co je ještě základ slova. Je zřejmé, že tu opět budeme mít velké problémy se složitějšími jazyky, u kterých se nám ve většině případů nepodaří nalézt správné kmeny slov.

2.2 Dolování dat

Fáze dolování dat představuje aplikaci příslušného algoritmu na již předzpracovanou databázi. Existuje mnoho typů dat, které lze pro dolování použít, a také existuje několik typů znalostí, které lze „data miningem“ získat.

Mezi nejpoužívanější typy dat (myšleno strukturované typy dat) patří relační a transakční databáze nebo datové skladby. Je možné použití také objektově orientované, prostorové, časové, multimediální a další databáze. V této práci se zaměřím na transakční databáze, které budou podrobněji vysvětleny.

2.2.1 Typy znalostí

Existuje mnoho znalostí, které lze dolovacími algoritmy získat z textových databází. Ze základních typů znalostí uvádím a podrobněji vysvětlím alespoň 4 nejzákladnější (nejpoužívanější):

2.2.1.1 Klasifikace

Klasifikace znamená rozdělení objektů nebo dokumentů do několika tříd na základě některých jejich atributů. Pro klasifikaci se používají nejvíce algoritmy ID3 (využívá rozhodovacích stromů), metody využívající neuronových sítí nebo Bayesovské klasifikace. Metoda itemsets, kterou se zabývá tato práce, bude vysvětlena v 4. Podrobněji o klasifikaci v 3.

2.2.1.2 Predikce

Predikce je předpovídání hodnot v budoucnosti na základě hodnot z minulosti. Musí jít o hodnoty, které lze modelovat statistickými funkcemi. Využívá se k tomu jedna z následujících typů regrese:

lineární, vícenásobná, polynomiální, exponenciální nebo logaritmická

2.2.1.3 Shlukování

Shlukování je roztřídění skupiny objektů do skupin. Tyto skupiny nejsou na rozdíl od klasifikace předem stanoveny. Jediné, co musí pro všechny shluky platit je, že rozdíly mezi objekty uvnitř jednoho shluku musí být minimální a rozdíly mezi jednotlivými shluky musí být maximální. Metody pro shlukování jsou například: Hierarchické metody (postupně rozdělují větší nebo slučují menší shluky). Vznikne hierarchická struktura shluků. Rozdělovací metody rozdělí vstupní množinu objektů na určitý předem daný počet podmnožin a dále optimalizují podle daného kritéria. Algoritmy založené na hustotě prvků a na mřížkách nebo metody využívající neuronové sítě.

2.2.1.4 Asociační pravidla

Asociační pravidla jsou jedny z nejpoužívanějších typů znalostí. Jsou výhodná tím, že jsou srozumitelná i pro uživatele, který není odborníkem v oblasti statistiky nebo jiných teoretických oblastech. Tento typ znalosti byl původně navržen pro transakční data používaná především pro analýzu nákupního košíku. Obecně jsou však asociační pravidla použitelná pro libovolný typ dat.

2.2.2 Transakční databáze

Transakční databáze se nejčastěji používají v obchodní sféře pro ukládání prodejních dat. Typickým příkladem jsou databáze nákupů v supermarketech. Každý nákup představuje transakci a každá transakce se skládá z položek.

Definice 2.1: Transakční databáze

Nechť $\boldsymbol{I} = I_1, I_2, \ldots, I_m$ je množina binárních atributů, které nazveme položkami. Nechť $\boldsymbol{T}$ je databáze transakcí. Každá transakce t je množinou položek takovou, že $t \subseteq I$. Takovou transakci lze reprezentovat binárním vektorem, kde $t[k] = 1$, jestliže t obsahuje položku I_k a $t[k] = 0$ v opačném případě. Pro každou transakci je v databázi jedna taková n-tice. Nechť $X \subseteq I$ je množina položek z $\boldsymbol{I}$. Pak řekneme, že transakce t odpovídá množině X, jestliže pro všechny položky $I_k \in X$ platí, že $t[k] = 1$.

ID transakce	Položky
1	{hodiny, baterie, ryba, krumpac}
2	{hodiny, ryba, lepidlo}
3	{hodiny, baterie, truhlik}
4	{ryba, krumpac}
5	{hodiny, baterie, ryba, krumpac, lepidlo}
6	{truhlik}
7	{krokodyl, ryba, polstar}
8	{krokodyl, salam, jar}
9	{jar, krumpac}

Obr. 2.1 : Příklad transakční databáze

2.3 Prezentace získaných znalostí

Získané znalosti prezentujeme takovým způsobem, aby byly srozumitelné běžnému uživateli. Dále pak zhodnotíme, zda jsou výsledky dolování v praxi využitelné.

3 Klasifikace dokumentů

S rozvojem informačních technologií a internetu je přímo spjat i vývoj databází. Bohužel daní za množství dostupných informací je i stále narůstající nepřehlednost a zmenšující se schopnost se v datech orientovat, což vede k nemožnosti nalézt v rozumně krátké době relevantní informace, a hledání v nestrukturovaných datech je dnes prakticky nemožné. Proto je nezbytné data (dokumenty) nějakým rozumným způsobem zařadit (klasifikovat) do příslušných tříd, k čemuž nám slouží nejrůznější algoritmy, které jsou v současné době předmětem výzkumu, a jejichž zdařilé implementace jsou ceněny pro svojí užitečnost.

Klasifikaci obecně rozdělujeme na dvě části. Konkrétně na *fázi učení* a na *fázi vlastní klasifikace*.

Fáze učení:

Fázi učení ještě rozdělujeme do následujících kroků:

- Odborník (odborníci) orientující se v dané problematice nadefinuje příslušnou hierarchii tématických oblastí, čímž nám vzniká množina klasifikačních tříd, jejíž mohutnost je rovna počtu listových témat v dané hierarchii.
- Expert zařadí určitý, pro každé z témat statisticky významný, počet dokumentů do stanovených témat. Vzniknou tak tzv. trénovací množiny.
- Provede se automatické natrénování klasifikátoru. Tím je myšleno, pro metodu itemsets, vytvoření charakteristických množin položek každého tématu (viz. 4.1.2).

Fáze klasifikace:

Po fázi učení nastupuje *fáze klasifikace*, během níž zařazujeme dokumenty do příslušných tříd, k čemuž využíváme poznatků získaných během fáze učení (trénování).

Výsledky klasifikačních algoritmů lze samozřejmě vyhodnotit jak z hlediska přesnosti P (jako anglické precision), tak z hlediska úplnosti R (jako anglického slova recall), a dále zavádíme pojem úspěšnost. Definice těchto pojmů je v 3.2.

V této kapitole je zaveden pojem digitální knihovna viz 3.1, která může být výsledkem procesu klasifikace.

3.1 Digitální knihovna

Pod tímto pojmem si můžeme představit systém, který shromažďuje nejrůznější informace v elektronické podobě, zajišťuje k nim snadný přístup a taktéž se stará o ochranu jejich obsahu. Veškeré materiály obsažené v takovéto knihovně tvoří součást vhodně zvolené organizační struktury.

Vytvoření a správa takovéto digitální knihovny představuje v současné době velice náročný úkol jak finančně tak časově. Každý příspěvek, který chceme do digitální knihovny vložit, musí nejdříve projít procedurou klasifikace, která nám zajistí co nejkvalitnější uspořádání obsahu knihovny. Pokud dojde během rozšiřování obsahu knihovny k rozšíření o nové tématické okruhy, pak je přímo nutností provést u veškerých dokumentů novou klasifikaci, neboť je možné, že jejich obsah patří také do nově vzniknuvších tříd.

Pokud bychom opomněly neustálou údržbu organizační struktury knihovny pak by vznikaly problémy se zařazením dokumentů, které by komplikovaly a zcela znemožňovaly efektivní vyhledávání požadovaných informací v knihovně. Snahou je tedy možnost provádět klasifikaci zcela automaticky, čímž by se samozřejmě snížily celkové vynaložené prostředky a došlo by k zpřístupnění takovéto knihovny většímu počtu uživatelů, nehledě na možnost prakticky neomezeného toku nových dokumentů, omezeného pouze paměťovou kapacitou a výkonem serveru.

3.2 Parametry vyhodnocování klasifikace

Přesnost (P) je pravděpodobnost, že vybraná třída je relevantní, a vypočítáme ji podle vzorce:

$$\mathbf{P = p / q}$$

Kde p – udává počet správně automaticky určených tříd.
q – je celkový počet automaticky určených tříd.

Úplnost (R) je pravděpodobnost, že relevantní třída je vybrána a vypočítáme ji podle vzorce:

$$\mathbf{R = p / r}$$

Kde p – udává počet správně automaticky určených tříd.
r – udává počet relevantních tříd stanovených ručně odborníkem.

Úspěšnost je dána parametry přesnosti a úspěšnosti a vypočítáme ji jako:

U = aritmetický průměr hodnot P a R pro všechny kategorizované dokumenty

4 Metoda itemsets

Pro tuto metodu představují množiny položek, tedy itemsets (nebo také frekventované vzory nebo také frekventované množiny prvků), naprosto klíčovou roli. (Samozřejmě, že nejen pro tuto metodu.) Tyto množiny položek představují společně se svojí četností (někdy označované jako podporou - support) základní popis zpracovávaných dat a lze je úspěšně uplatnit i v klasifikačních úlohách. Více o množinách položek je v 4.1.1. Metoda itemsets používá tzv. charakteristické množiny položek, jež jsou popsány v 4.1.2 . Pro získání častých množin položek se používá Apriori algoritmus, podrobně popsaný v [8], o němž se zmíním v 4.1.3. Vlastní metoda itemsets se dělí na 2 fáze a je popsána v 4.2 . V podkapitole 4.3 je pak předvedena metoda itemsets na jednoduchém příkladě.

Apriori algoritmus, popsaný v dostupné literatuře, používá výhradně pevnou hodnotu minimální podpory (uživatelem zadanou). Modifikace tohoto algoritmu, popsaná v 4.1.4, naproti tomu používá proměnnou hodnotu minimální podpory, čímž lze dosáhnout jednak 100% funkčnosti programu i za předpokladu nevhodně zadané hodnoty minsup (minimální podpory), která se mění dle potřeby správné funkčnosti, a také lze tímto způsobem ovlivnit celkový čas výpočtu bez ohledu na ostatní parametry a mohutnost trénovací databáze při fázi učení.

4.1 Generování vstupních dat

Jak již bylo naznačeno v úvodu, zásadním úkolem pro metodu itemsets je vygenerování množiny charakteristických položek. Pro vygenerování této množiny je potřeba znát množiny častých položek (popsány v 4.1.1.), které mají četnost (podporu) větší než minimální. Popisem charakteristických položek se zabývá kapitola 4.1.2 . Algoritmus apriori, pomocí něhož generujeme časté itemsety, je popsán v 4.1.3 .

4.1.1 Množiny častých položek

Množiny častých položek (itemsets nebo také frekventované množiny prvků) jsou tím hlavním vstupním parametrem pro metodu itemsets. Pro jejich získání lze použít nejznámější Apriori algoritmus (viz. 4.1.3), který byl původně určen k získávání frekventovaných množin položek z transakčních databází, ze kterých je možné dalším algoritmem získávat asociační pravidla.

Pro určení, které množiny prvků jsou frekventované (časté), si zavedeme metriku četnost (podpora-support).

Četnost=Podpora (*support*): množina položek(itemsets) má podporu *s* (četnost), jestliže *(s*100)%* řádků v tabulce obsahuje všechny položky z dané množiny. Jde tedy o frekvenci výskytu množiny v databázi.

Znamená to, že každá množina prvků má určitou četnost (většinou se nepočítá v % ale jen jako počet výskytů) podle toho, v kolika řádcích transakční databáze (řádkem je myšlena jedna transakce) je tato množina obsažena. Pokud je tato množina (itemset) obsažena v databázy vícekrát nebo stejněkrát, než je zadaná (uživatelem) minimální četnost, pak hovoříme o této množině jako o časté množině (frequent itemset). Množiny položek obsahující k prvků se označují jako k-množiny.

4.1.2 Charakteristické množiny položek

Pro účely klasifikace je nutné zjištění vazeb frekventovaných itemsets (častých množin položek) na dokumenty a k jednotlivým třídám. To znamená, že hledáme charakteristické itemsets. Následující označení množin a samotný vzorec převzat z [9].

Ke každé množině položek, označme si ji jako Π_j, lze nalézt množinu dokumentů, budeme ji značit jako $D\Pi_j$, která obsahuje Π_j, přičemž je nepodstatné, jak velká je frekvence výskytu množin položek v dokumentu, ale podstatným se stává pouze fakt, že daná množina položek se v dokumentu vyskytuje. Pro každé téma T_i existuje množina dokumentů DT_i zařazených do této třídy.

Pro účel zjištění, do jaké míry itemset Π_j charakterizuje třídu T_i si zavedeme váhový koeficient $w_{\Pi ji}$. Tento se vypočítá podle následujícího vzorce.

$$\mathbf{w_{\Pi ji} = | D\Pi_j \cap DT_i | \ / \ | DT_i |}$$

Vzorec nám udává v kolika procentech dokumentů třídy se daná množina položek vyskytuje. Bohužel nám tento vzorec nezohlední případy itemsetů, které jsou charakteristické pro větší počet tříd, nebo dokonce pro všechny třídy. V těchto případech nám totiž takové itemsety neposkytují žádnou informační hodnotu a stávají se nepříznivými pro účel klasifikace. Z tohoto důvodu se používá spíše vzorec uvedený v 4.2.1, který zohledňuje i výskyt itemsetů v jiných třídách.

Podle hodnot získaných váhových koeficientů se vybere pro každou třídu soubor charakteristických množin položek. Výběr můžeme uskutečnit například tak, že za charakteristické množiny položek označíme takové, u kterých váha přesahuje pro danou třídu jistou prahovou mez.

4.1.3 Apriori algoritmus

Apriori algoritmus představuje účinnou metodu při získávání asociačních pravidel, tedy nejpoužívanější formě znalosti získávané z databází. Najde však uplatnění i v kategorizaci dokumentů. Původní apriori algoritmus se aplikuje na transakční databáze a je tedy nutné zavést obdobu této databáze, která představuje množinu dokumentů reprezentovaných množinami významových termů. Množinám termům říkáme množiny položek (itemsets) a termům říkáme položky.

Základem tohoto algoritmu je apriori vlastnost, která zajišťuje efektivní generování kandidátních množin položek. V každém kroku algoritmu probíhá generování tzv. kandidátů a poté kontrola minimální podpory. V prvním kroku jsou kandidáty všechny jednoprvkové množiny. V dalším kroku vznikají kandidáti o velikosti 2 atd. Pokud žádný z kandidátů o dané velikosti není frekventovaný itemset, pak algoritmus končí. Množinu všech kandidátů o velikosti *k* označíme jako C_k a množinu všech frekventovaných množin o velikosti k označíme jako F_k.. Dále označme T jako množinu klasifikačních tříd a Π jako konkrétní itemset.

Algoritmus můžeme zjednodušeně popsat takto:

```
// pro 1-itemsety
C1 = všechna významová slova;
F1 = ∅;
For ∀ Πi ∈ C1 do
  For ∀ tj ∈ T do
        If  (podpora Πi je ve třídě tj větší než daný minimální práh)
              Then begin
                    Přidej Πi do F1
                    Break;  // není nutné zkoušet další třídy
              End;
// pro k-itemsety, k ≥ 2
For ∀ Πi ∈ Fk-1 do
  For ∀ Πj ∈ Fk-1 , Πi ≠ Πj do
        If  (shoda prvních k-2 položek v Πi a Πj   &&   poslední položky se liší)
              Then begin
                    c = Πi join Πj ;
                    // nyní se prověří apriori vlastnost – častost všech podmnožin c
                    if (existuje podmnožina s , s ⊂ c mající k-1 prvků, kde s ∉ Fk-1)
                          then break;
                          // c patří do Ck , a je nutné prověřit jeho četnost
                          else for ∀ tm ∈ T do
                                if (podpora c je v tj větší než daný minimální práh)
                                      then begin
                                            přidej c do Fk ;
                                            break;
                                      end;
              end;
```

Všimněme si druhé fáze algoritmu, kde se vylučují ty množiny, jejichž některá podmnožina není frekventovaná. Tento krok plyne z faktu, že podpora k-množiny nemůže být vyšší než podpora její podmnožiny. Tuto vlastnost nazýváme apriori-vlastnost. Je to nejpomalejší část algoritmu a existují některé efektivnější metody, které ale nejsou předmětem této práce.

4.1.4 Modifikace Apriori – proměnná minimální podpora

Ačkoliv je původní Apriori algoritmus nejvíce využívaným algoritmem pro získávání častých množin položek, je jeho aplikace na reálné databáze (jakou je například testovací Reuters 21578) v jistých ohledech naprosto nevhodná.

Podle literatury je běžně nalezeno kolem **16 000** frekventovaných položek v reálné databázi (myšleno trénovací databázi, tedy soubor zařazených trénovacích dokumentů). Pokud bychom chtěli vygenerovat najednou všechny dvouprvkové kandidátní množiny, tak musíme mít na paměti, že podle vzorce **(n^2 + n) / 2**, kde n + 1 je počet frekventovaných položek, budeme potřebovat paměť pro:

((15 999 ^2) + 15 999) / 2 = **127 992 000** kandidátních 2-prvkových množin.

Řešení, jak se vyhnout tak velikým paměťovým nárokům, se při zachování apriori vlastnosti nabízejí celkem dvě, a to:

1. Multipass Apriori algoritmus, který zpracovává F_1 po částech, čímž dosáhneme zmenšení paměťových nároků na max n-1, kde n je počet množin v F_1, dvouprvkových množin v každém kroku. Popsaný podrobně v [8].
2. Zvětšit hodnotu minimální podpory na takovou úroveň, aby bylo algoritmem generováno správné množství frekventovaných položek.

První způsob je sice velice elegantní, viz [8], ale na druhou stranu neřeší problém časové složitosti algoritmu, nehledě na problémy dále, z paměťových nároků dalších nezbytných výpočtů, vyplývající.

Druhý zbůsob je nejčastěji používaný, ale určení správné hodnoty minimální podpory nemusí být vždy jednoduché, neboť implementace algoritmu je téměř vždy omezena maximálním počtem prvků, a kolik k-prvkových kandidátních a častých množin položek bude v dalším kroku, je možno zjistit až během výpočtu. Problém je v tom, že při velmi malé hodnotě minimální podpory je počet kandidátních množin položek příliš velký již v prvním kroku, a dále se může v dalších krocích algoritmu tento počet ještě zvyšovat. Takto dojde snadno k přetečení paměťových možností počítače. Naopak při příliš vysoké hodnotě minimální podpory se může snadno stát, že počet frekventovaných množin položek získaných Apriori algoritmem bude příliš nedostatečný pro klasifikaci a čas potřebný pro samotnou klasifikaci bude sice mnohem nižší, naproti tomu klasifikace nebude uspokojivě

úspěšná a přesná. Jelikož je vhodná hodnota minimální podpory jiná pro každou konkrétní trénovací databázi, může být hledání této hodnoty značně časově náročné.

Jelikož ani jeden z těchto dvou způsobů jsem nepovažoval za dostatečně efektivní pro získání častých množin položek pro účely klasifikace, modifikoval jsem Apriori algoritmus takovým způsobem, aby se hodnota minimální podpory měnila v každém kroku algoritmu podle potřeby. Samozřejmě tento způsob implementace zcela ignoruje apriori vlastnost využívanou během generování kandidátních množin. Předpokládá se tedy, že i když některá nefrekventovaná podmnožina kandidátní množiny, která není frekventovaná pro aktuální hodnotu minimální podpory, není vyřazena a je zapsána jako kandidátní, tak to přesto výpočet příliš nezdrží (při hledání počtu výskytů kandidátních množin v transakcích se taková množina stejně následně vyřadí, pokud nebude mít dostatečnou podporu).

Kroky modifikovaného Apriori algoritmu můžeme velice zjednodušeně popsat takto:

1) najdi jedinečné položky
2) spočti výskyty jedinečných položek v transakcích
3) změň hodnotu minimální podpory tak, aby počet frekventovaných položek z množiny jedinečných položek byl ≤ maximální hodnotě
4) podle podpory utvoř množinu frekventovaných jednoprvkových množin
5) opakuj, dokud má množina kandidátních prvků více než 0 prvků:

    k++;

    vytvoř množinu kandidátních Ck množin prvků;

    spočti výskyty množin z Ck;

    nastav hodnotu minimální podpory tak, aby počet frekventovaných množin položek vybraných z Ck byl ≤ maximální hodnotě;

    podle podpory utvoř množinu frekventovaných jednoprvkových množin;

Tento zápis samozřejmě předpokládá hlubší znalost principu původního Apriori algoritmu popsaného v 4.1.3, nebo ještě lépe v [8]. Ve stručnosti jde o to, že v každém kroku algoritmu, kdy na základě četnosti kandidátní k-prvkové množiny vybíráme takové množiny jejichž četnost výskytu je minimálně stejná jako je minimální hodnota minimální podpory, nastavíme tento mimimální práh (minimální podporu) tak, aby počet vybraných frekventovaných množin byl co možná nejbližší, ale ne větší, než je maximální zvolená hodnota. Nejjednoduší implementace spočívá v nastavení minimálního prahu na úroveň zadanou uživatelem a následnému spočtení, kolik frekventovaných

množin takto získáme, a následná inkrementace hodnoty minimálního prahu, pokud je počet frekventovaných větší než maximální hodnota, atd…, dokud nedosáhneme počtu frekventovaných množin položek vyhovujícímu, tedy ≤ maximální hodnotě.

Výhoda oproti jiným řešením spočívá v tom, že počet množin frekventovaných položek bude téměř stejný pro různé databáze, křivka závislosti počtu frekventovaných množin na počtu položek v množině bude mít tvar „vrhu vodorovného“ (ze začátku spíše lineární průběh…) a ne vrhu pod úhlem, kde nemůžeme s jistotou určit meze potřebné paměti, a hlavně lze snadno nastavit maximální počet výsledných itemsetů, čímž lze dosáhnout zrychlení samotné klasifikace (zařazování dokumentů do tříd). Výsledné experimenty vlivu použití modifikace změny minimální podpory jsou popsány v kapitole 6.

4.2 Fáze metody itemsets

Vlastní metoda itemsets je oproti Apriori algoritmu, sloužícímu pouze pro generování vstupních dat v podobě častých itemsetů, původní a dělíme ji, dle [9], na dvě fáze.

4.2.1 Fáze učení

Pro každou množinu frekventovaných položek Π_j, můžeme nalézt množinu dokumentů, budeme ji značit jako $D\Pi_j$, která obsahuje Π_j. Pro každé téma T_i existuje charakteristická množina dokumentů DT_i zařazených do tohoto tématu. Tématu T_1 odpovídá množina DT_2, tématu T_2 odpovídá množina DT_2, atd. až vytvoříme L množin dokumentů.

Úkolem je zjistit určitý soubor charakteristických množin položek pro každý okruh témat, přičemž každá množina položek je asociována s podmnožinou množiny všech témat. Množina položek Π_j je asociována s těmi tématy T_i, kde hodnoty váhy $w_{\Pi j}$ přesahují definovanou prahovou hodnotu. Efektivní vzorec pro výpočet $w_{\Pi j}$ oproti 4.1.2 je (vzorec převzat z [9]):

$$\mathbf{w_{\Pi j} = | D\Pi_j \cap DT_i | \ / \ | DT_i | \ x \ [1 + \ | D\Pi_j | - | D\Pi_j \cap DT_i |]}$$

kde i = 1, 2, 3, ….

V tomto vzorci jmenovatel normalizuje váhy s ohledem na počet dokumentů patřících do T_i a také s ohledem na výskyt množiny položek i v jiných tématech. Tímto způsobem se potlačí důležitost termů, které se často vyskytují i v jiných dokumentech než DT_i.

4.2.2 Fáze klasifikace

Během této fáze musíme vzít v úvahu také různou mohutnost množin položek. Ze statistického hlediska je totiž významné odlišit shodu dvojic od shody ve trojicích atd. Pro tento účel nám poslouží váhový faktor w_{fj}, jenž odpovídá mohutnosti množiny položek. Je to tedy pro jedince w_{f1}, pro dvojice w_{f2}, atd.

Teď lze provést zařazování dokumentů do tématických okruhů. Předpokládejme, že soubor C_j obsahuje položky Π_{j1}, Π_{j2}, Π_{j3} , …, $\Pi_{|Cj|}$. Vypočítáme váhu odpovídající přesnosti asociace dokumentu D s tématem T_j podle vzorce:

$$W_{Tj}^{D} = \sum_{i=1}^{|Cj|} wf_{|\Pi i|} \times w_{\Pi i}$$

Kde $(\Pi_j \in C_j) \wedge (\Pi_i \subseteq D)$ pro všechna j = 1, 2, 3, …, L

To znamená že, váha klasifikace je určená součtem součinů vah $w_{\Pi i}$ s váhovými faktory $wf_{|\Pi i|}$ pro všechny množiny položek daného tématu.

Dokument D zařadíme do tématu T_j, jemuž odpovídá nejvyšší váha odpovídající přesnosti asociace dokumentu.

Pokud bychom chtěli dokument zařadit do více tříd (témat), pak dokument D zařazujeme do všech tříd, pro které váha odpovídající přesnosti asociace přesáhne určité procento hodnoty maximální dosažené váhy. Kolik procent si v takovémto případě zvolíme je klíčovým parametrem pro výslednou přesnost a úplnost, neboť s rostoucími procenty klesá přesnost klasifikace, ale úplnost naopak roste.

4.3 Metoda itemsets podrobně v příkladu

Teoreticky popsaný náznak metody itemsets je natolik nedostatečný, že nejlepším způsobem, jak pochopit zcela a do nejmenších detailů, jak metoda itemsets funguje, bude uvést konkrétní jednoduchý příklad s komentářem 4.3.1.

Jelikož byl algoritmus Apriori popsán v [8], nebudeme se jím již nyní dopodrobna zabývat a pouze uvedeme výsledný výstup množin kandidátních a častých položek, které byly získány po jeho aplikaci na zvolenou databázi.

4.3.1 Příklad funkce metody itemsets

Zadání příkladu je v podkapitole 4.3.1.1, fáze učení v 4.3.1.2 a fáze klasifikace je popsána a předvedena v 4.3.1.3.

4.3.1.1 Zadání demonstračního příkladu

Př. 4.1: (ukázka činnosti metody itemsets s komentářem)

Mějme celkem 2 klasifikační třídy, do kterých budeme dokumenty řadit. První třídu označme CPU (procesory) a druhou třídu označme GRAF (grafické karty). Dále mějme předzpracované trénovací dokumenty do formy trénovací transakční databáze viz obr. 4.1, dle definice 2.1, a 2 dokumenty určené ke klasifikaci (obr. 4.2). Nechť hodnota minimální podpory (minsup) je rovna 2.

ID transakce	Položky	Třída (téma)
1	{celeron, 2000, mhz, mmx, dualcore, cache}	CPU
2	{pentium, 3, mmx, cache, 1, mb}	CPU
3	{ati, pameť, 256, mb, agp}	GRAF
4	{matrox, 128, mb, agp}	GRAF

Obr. 4.1 : transakční (trénovací) databáze

ID	Položky dokumentu
1	{celeron, 3000, mhz, mmx, sse3, dualcore, cache, 2, mb}
2	{ge-force, 256, 32, mb, agp, opengl, directx}

Obr. 4.2 : dokumenty určené ke klasifikaci (forma transakční DB)

4.3.1.2 Příklad - fáze učení

Cílem této fáze je získání množin častých položek (itemsetů) pomocí Apriori algoritmu a jejich následné váhové ocenění z hlediska jednotlivých tříd. Takto získáme charakteristické množiny položek pro každé téma (třídu). Prakticky to znamená, že větší vahou, pro dané jedno téma, oceníme každý jeden itemset, který danou třídu lépe charakterizuje. Naproti tomu váha pro ostatní témata bude pro tento konkrétní itemset menší než váha v rámci třídy, pro kterou daný itemset je charakteristický.

Výsledkem fáze učení je tedy tabulka (viz obr. 4.8), jež obsahuje všechny itemsety a jejich váhy pro každé téma (třídu).

Nejprve uveďme výsledky aplikace Apriori algoritmu na transakční DB (obr 4.1), (podrobně v [8]) tabulka na obr. 4.3 až 4.7 :

ID	Položka	Počet výskytů
1	Celeron	1
2	2000	1
3	Mhz	1
4	Mmx	2
5	Dualcore	1
6	Cache	2
7	Pentium	1
8	3	1
9	1	1
10	Mb	3
11	Ati	1
12	Paměť	1
13	256	1
14	Agp	2
15	Matrox	1
16	128	1

Obr. 4.3 : tabulka jedinečných položek a počet jejich výskytů v transakcích

Z této tabulky vybereme frekventované jednoprvkové itemsety. (počet výskytů ≥ minsup) tj. položky 4, 6, 10 a 14 (viz obr 4.4).

ID	Položka
1	mmx
2	Cache
3	Mb
4	agp

Obr. 4.4 : tabulka frekventovaných položek

Z tabulky 4.4 utvoříme dvouprvkové kandidátní množiny prvků a spočteme jejich výskyty v databázi. Výsledek udává tabulka na obr 4.5.

ID	Množina	Počet výskytů
1	{mmx, cache}	2
2	{mmx, mb}	1
3	{mmx, agp}	0
4	{cache, mb}	1
5	{cache, agp}	0
6	{mb, agp}	2

Obr. 4.5 : tabulka kandidátních dvouprvkových množin a počet jejich výskytů v transakcích

Z této tabulky vybereme frekventované dvouprvkové itemsety. (počet výskytů ≥ minsup) tj. množiny 1 a 6 (viz obr 4.6).

ID	itemset
1	{mmx, cache}
2	{mb, agp}

Obr. 4.6 : tabulka frekventovaných dvouprvkových množin

Z tabulky 4.6 nelze již vytvořit přirozeným spojením n-1 prvků žádnou kandidátní množinu a algoritmus Apriori tímto končí. Celkem jsme tedy zíkali 6 itemsetů (viz obr 4.7).

ID	Itemset
1	Mmx
2	Cache
3	Mb
4	agp
5	{mmx, cache}
6	{mb, agp}

Obr. 4.7 : tabulka všech frekventovaných množin prvků (itemsetů)

Nyní je potřeba spočítat pro každý itemset a každé téma váhy podle vzorce z 4.2.1., což představuje množství rovné počtu itemsetů krát počet témat, tedy celkem 12 vah:

Váha itemsetu {mmx} pro téma CPU:

$$W_{[\{mmx\}]CPU} = \frac{|\, a \cap b \,|}{|\, a \,| \times [1 + |\, a \,| - |\, a \cap b \,|\,]} = \frac{|\, \{1, 2\} \cap \{1, 2\} \,|}{|\{1, 2\}| \times [1 + |\{1, 2\}| - |\, \{1, 2\} \cap \{1, 2\} \,|\,]} =$$

$$= \frac{|\, \{1, 2\} \,|}{|\, a \,| \times [1 + |\, a \,| - |\, a \cap b \,|\,]} = \frac{2}{2 \times [1 + 2 - 2\,]} = \frac{2}{2} = 1$$

Kde: a = množina dokumentů obsahujících itemset {mmx}

b = množina dokumentů zařazených do tématu CPU

Váha itemsetu {mmx} pro téma GRAF:

$$W_{[\{mmx\}]GRAF} = \frac{|\, \{1, 2\} \cap \{3, 4\} \,|}{|\{3, 4\}| \times [1 + |\{1, 2\}| - |\, \{1, 2\} \cap \{3, 4\} \,|\,]} = \frac{0}{6} = 0$$

Váha itemsetu {cache} pro téma CPU:

$$W_{[\{cache\}]CPU} = \frac{|\, \{1, 2\} \cap \{1, 2\} \,|}{|\{1, 2\}| \times [1 + |\{1, 2\}| - |\, \{1, 2\} \cap \{1, 2\} \,|\,]} = \frac{2}{2} = 1$$

Váha itemsetu {cache} pro téma GRAF:

$$W_{[\{cache\}]GRAF} = \frac{|\, \{1, 2\} \cap \{3, 4\} \,|}{|\{3, 4\}| \times [1 + |\{1, 2\}| - |\, \{1, 2\} \cap \{3, 4\} \,|\,]} = \frac{0}{6} = 0$$

Váha itemsetu {mb} pro téma CPU:

$$W_{[\{mb\}]CPU} = \frac{|\, \{2, 3, 4\} \cap \{1, 2\} \,|}{|\{2, 3, 4\}| \times [1 + |\{2, 3, 4\}| - |\, \{2, 3, 4\} \cap \{1, 2\} \,|\,]} = \frac{1}{6}$$

Váha itemsetu {mb} pro téma GRAF:

$$W_{[\{mb\}]GRAF} = \frac{|\, \{2, 3, 4\} \cap \{3, 4\} \,|}{|\{2, 3, 4\}| \times [1 + |\{2, 3, 4\}| - |\, \{2, 3, 4\} \cap \{3, 4\} \,|\,]} = \frac{2}{4} = \frac{1}{2}$$

Váha itemsetu {agp} pro téma CPU:

$$w_{[\{agp\}]CPU} = \frac{|\{3,4\} \cap \{1,2\}|}{|\{3,4\}| \times [1 + |\{3,4\}| - |\{3,4\} \cap \{1,2\}|]} = \frac{0}{6} = 0$$

Váha itemsetu {agp} pro téma GRAF:

$$w_{[\{agp\}]GRAF} = \frac{|\{3,4\} \cap \{3,4\}|}{|\{3,4\}| \times [1 + |\{3,4\}| - |\{3,4\} \cap \{3,4\}|]} = \frac{1}{1} = 1$$

Váha itemsetu {mmx, cache} pro téma CPU:

$$w_{[\{mmx,\ cache\}]CPU} = \frac{|\{1,2\} \cap \{1,2\}|}{|\{1,2\}| \times [1 + |\{1,2\}| - |\{1,2\} \cap \{1,2\}|]} = \frac{2}{2} = 1$$

Váha itemsetu {mmx, cache} pro téma GRAF:

$$w_{[\{mmx,\ cache\}]GRAF} = \frac{|\{1,2\} \cap \{3,4\}|}{|\{3,4\}| \times [1 + |\{1,2\}| - |\{1,2\} \cap \{3,4\}|]} = \frac{0}{6} = 0$$

Váha itemsetu {mb, agp} pro téma CPU:

$$w_{[\{mb,agp\}]CPU} = \frac{|\{3,4\} \cap \{1,2\}|}{|\{3,4\}| \times [1 + |\{3,4\}| - |\{3,4\} \cap \{1,2\}|]} = 0$$

Váha itemsetu {mb, agp} pro téma GRAF:

$$w_{[\{mb,agp\}]GRAF} = \frac{|\{3,4\} \cap \{3,4\}|}{|\{3,4\}| \times [1 + |\{3,4\}| - |\{3,4\} \cap \{3,4\}|]} = \frac{2}{2} = 1$$

Výsledek fáze učení je přehledně zpracován do tabulky na obr 4.8, kde jsou nejenom zaneseny výsledné váhy, ale taktéž sloupec kolikaprvkový itemset je. Tento údaj slouží k výpočtu vah při fázi klasifikace viz 4.3.1.3.

ID	Itemset	Kolikaprvkový itemset	Váha pro téma CPU	Váha pro téma GRAF
1	Mmx	1	1	0
2	Cache	1	1	0
3	Mb	1	1/6	½
4	agp	1	0	1
5	{mmx, cache}	2	1	0
6	{mb, agp}	2	0	1

Obr. 4.8 : tabulka vah itemsetů – výsledek fáze učení

4.3.1.3 Příklad - fáze klasifikace

V této fázi již probíhá samotné řazení dokumentů do tříd. Nejprve je potřeba vypočítat výslednou váhu pro každou třídu, a poté určit podle největší váhy, do které třídy dokument patří.

Váha pro každé téma je rovna součtu všech vah vynásobených faktorem mohutností itemsetu (viz 4.2.2), které odpovídají itemsetům, jež jsou v dokumentu obsaženy (tabulka na obr 4.9).

Dokument 1 z tabulky z obr 4.2 obsahuje itemsety {mmx}, {cache}, {mb} a {mmx, cache}. Vypočítáme váhu odpovídající přesnosti asociace dokumentu 1 s tématem CPU. K tomu použijeme data z tabulky na obr 4.8:

$$W^1_{CPU} = wf_1 * w_{[\{mmx\}]CPU} + wf_1 * w_{[\{cache\}]CPU} + wf_1 * w_{[\{mb\}]CPU} + wf_2 * w_{[\{mmx, cache\}]CPU} =$$
$$= 1 * 1 + 1 * 1 + 1 * 1/6 + 2 * 1 = 4 \text{ a } 1/6$$

Vypočítáme váhu odpovídající přesnosti asociace dokumentu 1 s tématem GRAF:

$$W^1_{GRAF} = wf_1 * w_{[\{mmx\}]\,GRAF} + wf_1 * w_{[\{cache\}]\,GRAF} + wf_1 * w_{[\{mb\}]\,GRAF} + wf_2 * w_{[\{mmx, cache\}]\,GRAF} =$$
$$= 1 * 0 + 1 * 0 + 1 * 1/2 + 2 * 0 = 1/2$$

Druhý dokument z tabulky z obr 4.2 obsahuje itemsety {mb}, {agp} a {mb, agp}. Vypočítáme váhu odpovídající přesnosti asociace druhého dokumentu s tématem CPU:

$$W^2_{CPU} = wf_1 * w_{[\{mb\}]CPU} + wf_1 * w_{[\{agp\}]CPU} + wf_2 * w_{[\{mb, agp\}]CPU} =$$
$$= 1 * 1/6 + 1 * 0 + 2 * 0 = 1/6$$

Vypočítáme váhu odpovídající přesnosti asociace druhého dokumentu s tématem GRAF:

$$W^2_{GRAF} = wf_1 * w_{[\{mb\}]\,GRAF} + wf_1 * w_{[\{agp\}]\,GRAF} + wf_2 * w_{[\{mb, agp\}]\,GRAF} =$$
$$= 1 * 1/2 + 1 * 1 + 2 * 1 = 3 \text{ a } 1/2$$

Nyní nám vznikla tabulka (obr 4.9), ze které je snadné určit, do které třídy který dokument patří. Samozřejmě je jasné, že pokud by dokument hovořil jak o tématu 1, tak o tématu 2, pak by výsledné váhy mohly mít stejnou, nebo podobnou hodnotu.

Protože tento jev není v reálných databázích nijak neobvyklý, je dobré řadit dokumenty ne pouze do jedné, ale do několika tříd, pakliže získané váhy pro dané téma přesahují jisté % maximální dosažené hodnoty váhy. Více v kapitole 6.

Id dokumentu	Váha pro téma CPU	Váha pro téma GRAF
1	4 1/6	½
2	1/6	3 ½

Obr. 4.9 : tabulka vah – přesnost asociace

Prvnímu dokumentu nejvíce odpovídá téma CPU a druhému téma GRAF, což je 100% správný výsledek klasifikace, kterého lze dosáhnout pouze na specifických příkladech a speciálně upravených databázích, nikoli v reálné databázi jakou je například Reuters 21578 [RC] (viz výsledky experimentů v 6).

5 Dokumentace a implementace

Tato kapitola rozebírá samotné řešení, volbu programových prostředků samotné implementace, a také pojednává o formátu vstupních dat, výstupních dat o chování a ovládání programu (5.5).

V kapitole 5.2 je pojednáno o předzpracování dat. Kapitola 5.3 vysvětluje jakým způsobem vznikl slovník nepotřebných slov. Kapitola 5.1 popisuje obsah a formát vstupních a výstupních souborů, keré program využívá. Kapitola 5.4 slouží jako manuál k programu. Kapitola 5.5 se zmiňuje o užitých programových prostředcích.

5.1 Vstupní a výstupní soubory programu

V této kapitole jsou popsány formáty a obsah jednotlivých souborů, které slouží jako vstupní (kapitola 5.1.1) nebo výstupní (kapitola 5.1.2), ze kterých program čte nebo je sám generuje.

5.1.1 Vstupní soubory

Vstupní soubory jsou takové, ze kterých program načítá data. (Jsou otevřeny pouze pro čtení.)

Podkapitola 5.1.1.1 popisuje konfigurační soubor config.txt, ve kterém je možné měnit nastavení použitá při běhu algoritmu. Kapitola 5.1.1.2 popisuje formát slovníku nepotřebných slov. V kapitolách 5.1.1.3 a 5.1.1.5 se dozvíme o formátu souborů, jež obsahují data s čísly dokumentů a příslušnosti k tématům, neboli informaci o trénovací databázi. Soubory, obsahující dokumenty jak trénovací databáze, tak nezařazené dokumenty, jsou popsány v kapitolách 5.1.1.4 a 5.1.1.6.

Soubory, které jsou zmíněny v kapitole 5.1.1.7, jsou pak podrobněji popsány v 5.1.2.2 a kapitole 5.1.2.3.

5.1.1.1 Soubor config.txt

Soubor config.txt – soubor s nastavením programu. Jednotlivá možná nastavení popsaná níže:

minsup: 2

hodnota která udává minimální (uživatelem zadanou) podporu při hledání itemsetů

metoda_zmeny_minsup: 1

možná hodnota 1 (true), 0 (false) – udává, zda použít při hledání množin častých položek proměnou hodnotu minimální podpory

kolik_souboru_reuters_zpracovat: 22

v případě že má být použito kolekce Reuters 21578 [RC], tak toto číslo udává na kolik z 22 souborů má být metoda itemsets aplikována (myšleno od prvního, následné soubory budou ignorovány)

minimalni_mozna_vaha: 50

udává (v miliontinách) jaká je hodnota minimální výsledné váhy pro řazení. V případě že největší dosažená váha ze všech bude menší než tato hodnota, pak dokument zůstane nezařazený.

pouzit_stejne_k_razeni: 1

možná hodnota 1(true), 0(false) – pro potřebu validace programu z hlediska korektnosti zařazování jsou v případě true zařazovány ty dokumenty, které byly použity ke klasifikaci

test: 0

namísto kolekce Reuters 21578 [RC] je použita databáze test.sgm a za cvičné dokumenty jsou brány ty, které jsou v souboru test.read_d

min_%_z_max_vahy: 75

dokumenty budou zařazeny do každého z tématů, jehož váha odpovídá % z hodnoty váhy maximální.

max_polozek_v_transakci_omezeni_apriory: 4

toto číslo udává, kolikaprvkové (maximálně) itemsety bude generovat Apriori algoritmus

predzpracovavat_databazi: 0

jelikož jsou jako vstupní databáze brány trénovací databáze (soubor Database.txt) a databáze nezařazených dokumentů (soubor nezarazene.txt), jsou při hodnotě 1 (true) tyto soubory vytvořeny ze souborů klekce Reuters 21578 [RC] (v případě nastavení test: 0, jinak se zpracuje cvičná databáze, která má stejný formát jako kolekce Reuters 21578 [RC]), jinak se předpokládá, že tyto soubory již existují a neproběhne fáze předzpracování.

V případě chybné hodnoty, mimo stanovené meze, program nahlásí chybu a automaticky přidělí příslušné proměnné výchozí hodnotu. To znamená že špatně „vyplněný“ konfigurační soubor programu nevadí, ale nepovede nejspíše k požadovaným výsledkům.

5.1.1.2 Soubor slovnik_nepotrebnych.txt

Soubor slovnik_nepotrebnych.txt obsahuje slova, která jsou z dokumentů vyřazena během fáze předzpracování dat. Jedná se o taková slova, která nemají žádný nebo negativní vliv na klasifikaci. Buď jsou to slova, která se vyskytují v naprosté většině dokumentů, nebo se naopak vyskytují pouze v jediném dokumentu, a proto nemohou být častá. Formát souboru:

Slovo
Slovo
…
slovo
prázdný řádek
EOF (konec souboru)

Kde - Prázdný řádek na konci souboru je důležitý. Program jinak nepozná, že je na konci souboru. Každé slovo je samostatně na řádku a není před ním ani za ním žádný z oddělovacích znaků (mezera, tab, …).

5.1.1.3 Soubor cate90_train_rel.read_d

Soubor obsahující čísla dokumentů Reuters 21578 [RC] a příslušnost k tématům. V tomto souboru jsou tedy označeny trénovací dokumenty. Formát souboru:

c90_train.rel has 7769 documents:

Číslo: téma téma … téma

Číslo: téma téma … téma

…

Kde – číslo představuje identifikační číslo dokumentu z kolekce Reuters 21578 [RC], a téma představuje název tématu, do kterého tento dokument patří. Pokud dokument patří do více tříd (témat), jsou jednotlivá témata oddělena mezerou. Na každém řádku je jeden údaj. První řádek je při načítání ignorován, neboť má pouze informativní charakter. (Soubor je součástí kolekce Reuters 21578 [RC]).

5.1.1.4 Soubory kolekce reuters 21578 - reut2-000.sgm až reut2-021.sgm

Formát souborů je vysvětlen (anglicky) v dokumentaci k reuters 21578 [RC]. V podstatě pro naše účely stačí vědět že číslo dokumentu je označeno jako NEWID="číslo dokumentu", dokument začíná tagem <REUTERS …. . samotný text dokumentu je mezi tagy <BODY> text dokumentu </BODY>. Dokument uzavírá tag </REUTERS>.

5.1.1.5 Soubor test.read_d

Soubor obsahující čísla dokumentů z testovací mikrodatabáze a příslušnost k tématům. V tomto souboru jsou tedy označeny trénovací dokumenty. Formát souboru:

test.read_d has 12 documents:

Číslo: téma

Číslo: téma

…

Kde – číslo představuje identifikační číslo dokumentu z testovací databáze a téma představuje název tématu, do kterého tento dokument patří. Na každém řádku je jeden údaj. První řádek je při načítání ignorován, neboť má pouze informativní charakter. Tento soubor má význam pro demonstraci a ladění programu, neboť velikost testovací databáze je malá, přehledná a výpočetně nenáročná.

5.1.1.6 Soubor test.sgm

Tento soubor slouží jako testovací vstupní databáze a je vytvořen na základě prvního souboru z kolekce reuters 21578 (soubor reut2-000.sgm), ve kterém byly odmazány nadbytečné dokumenty (soubor jich obsahoval původně 1000) a obsah těchto dokumentů (text dokumentů) byl nahrazen za jiný.

Jedná se tedy spíše o demonstrační a ladící soubor používaný při vytváření minipřekladače pro kolekci reuters 21578. Obsahuje celkem 20 dokumentů, z toho poslední 4 nemají žádný obsah (0 položek), a proto jsou programem ignorovány. Prvních 12 slouží jako trénovací databáze a další 4 jako neznámé nezařazené dokumenty z oblasti (témat) grafických karet a procesorů. Soubor je použit místo kolekce reuters v případě nastavení test: 1 v config.txt.

5.1.1.7 Soubory Database.txt a nezarazene.txt

Soubory Database.txt (kap. 5.1.2.2) a nezarazene.txt (kap. 5.1.2.3) sice patří mezi vstupní soubory (pokud neprobíhá fáze předzpracování dat viz konfigurační soubor kap. 5.1.1.1), ale vzhledem k tomu, že jsou tyto soubory výsledkem fáze předzpracování, tak jsou popsány v kapitole o výstupních souborech programu 5.1.2.

5.1.2 Výstupní soubory

Výstupní soubory jsou ty, které program sám generuje, nebo mění jejich obsah (v případě že již existují, jsou přepsány novými daty).

Podkapitola 5.1.2.1 popisuje soubor vysledek.txt, který obsahuje výsledek klasifikace a také kontrolní výpisy proměnných pro potřeby statistického vyhodnocení. Kapitola 5.1.2.2 rozebírá soubor Database.txt, který slouží jako trénovací transakční databáze. V kapitole 5.1.2.3 je pak popis souboru nezarazene.txt, který představuje nezařazené dokumenty ve formě transakční databáze.

5.1.2.1 Soubor vysledek.txt

Tento soubor obsahuje veškeré výsledky řadícího algoritmu včetně časů výpočtu jednotlivých fází. V souboru je také výpis načteného konfiguračního souboru pro snažší vyhodnocení klasifikační metody (tedy výsledky řazení při konkrétním nastavení parametrů). Ukázka obsahu souboru:

```
minsup zadany je = 2
pouzit metodu zmeny minsup = 1
kolik_souboru_reuters_zpracovat = 22
minimalni_mozna_vaha = 0.000050
pouzit stejne dokumenty k razeni jako ke klasifikaci = 1
Patri do tridy pokud ma 75 procent max vahy
Max polozek itemsetu 4
```

```
 Predzpracovat databazi = 1
databaze zpracovana OK za  77.47 sekund.                    (pozn. 1)
1 krok, 9667 Ck mnozin, 149 Fk mnozin, minsup je nyni = 98
2 krok, 11026 Ck mnozin, 147 Fk mnozin, minsup je nyni = 173
3 krok, 216 Ck mnozin, 149 Fk mnozin, minsup je nyni = 153
4 krok, 85 Ck mnozin, 85 Fk mnozin, minsup je nyni = 2
Faze uceni dokoncena za  47.63 sekund pro minsup = 2 ....
Radim....pouzivam k tomu 530 itemsetu
******************
 1: EARN                                                    (pozn. 2)
 9: EARN
…
14751: nezarazeno
14805: COCOA HOUSING 14805: COCOA HOUSING
***************
dokumentu k razeni: 2996                                    (pozn. 3)
nezarazenych: 434
uspesne zarazenych: 2562
***************
dokumentu shodne zarazenych: 2225                           (pozn. 4)
dokumentu jinak zarazenych: 337

Klasifikace za  23.19 sekund.
```

Jako první věc je vypsána každá proměnná z konfiguračního souboru. Poté, v případě že je zadáno předzpracování databáze, je vypsán čas, za který byla databáze zpracována, a tedy vytvořeny soubory Database.txt a nezarazene.txt (pozn. 1). Následuje výpis výsledků činnosti Apriori algoritmu na trénovací databázi. Dále (pozn. 2) výsledek řazení ve stejném formátu jako v souboru cate90_train_rel.read_d (nebo test.read_d), kde opět samostatně na řádku je vždy číslo dokumentu: témata, do kterých byl zařazen, případně poznámka nezarazeno. Další výpis (pozn. 3) nám sděluje, kolik dokumentů bylo celkově řazeno a kolik jich bylo nezařazených (váha pro jakékoliv téma nebyla dostatečně velká) a také kolika algoritmus přiřadil alespoň jedno téma. Položky -dokumentu shodne zarazenych a dokumentu jinak zarazenych (pozn. 4) nejsou vypsány v případě nastavení „pouzit stejne dokumenty k razeni jako ke klasifikaci“ na hodnotu 0 (false). Je to proto, že tyto 2 údaje nám říkají, o kolik se liší zařazení trénovacích dokumentů v případě, že je ve fázi klasifikace použijeme jako nezařazené dokumenty místo nových dokumentů, které se ve fázi učení nevyskytly. Poslední řádek výstupu představuje čas, který spotřeboval program při samotné klasifikaci dokumentů.

5.1.2.2 Soubor Database.txt

Soubor, který je vytvořen ve fázi předzpracování dat. Původně byl myšlen jako jediný vstupní soubor s trénovací databází, neboť se tato práce neměla fází předzpracování dat a vytvoření slovníku nepotřebných slov v prvopočátku zabývat. Formát souboru musí být striktně dodržen, neboť s jakoukoliv chybou program nepočítá a může dojít k chybným výsledkům. Formát souboru s trénovací databází je tedy následující:

položka položka … položka <tr> <číslo>

položka položka … položka <tr> <číslo>

…

Kde - položka představuje slovo z dokumentu a je odděleno mezerou, případně koncem řádku. Žádná z položek se nesmí v jednom dokumentu (jedné transakci) vyskytnout vícekrát (jedná se tedy o množinu jedinečných položek dokumentu). Značka <tr> označuje konec transakce (trénovacího dokumentu). Značka (tag) <číslo>, kde číslo je číslo typu int větší než 0, označuje číslo tématu, do kterého dokument patří.

5.1.2.3 Soubor Nezarazene.txt

Soubor, který je vytvořen ve fázi předzpracování dat. Obsahuje dokumenty určené k řazení. Formát souboru je obdobný formátu Database.txt:

položka položka … položka <tr> číslo

položka položka … položka <tr> číslo

…

Kde - položka představuje slovo z dokumentu a je odděleno mezerou případně koncem řádku. Na rozdíl od trénovacích dokumentů nevyžaduje algoritmus, aby v jednom dokumentu byla položka pouze jednou. Při předzpracování se ale vypisují pouze jedinečné položky, neboť to zrychluje algoritmus. Značka (tag) <tr> označuje konec transakce (dokumentu). Číslo poté představuje číslo dokumentu (například v rámci reuters 21578 [RC]), které je použito následně do výpisu výsledného zařazení.

5.2 Předzpracování dat

V této fázi program buď ze souboru test.sgm, nebo ze souborů kolekce Reuters 21578 [RC], vytvoří 2 soubory. Soubor s testovací databází (Database.txt) a soubor s dokumenty určenými k řazení (nezarazene.txt). Samozřejmě záleží na nastavení konfiguračního souboru, zda použije testovací databázi nebo Reuters (nastavení test).

Program postupně prochází soubory s databází a na základě souboru cate90_train_rel.read_d (případně test.read_d), jehož obsah načte do pomocného pole, určí podle příslušnosti k testovacím dokumentům a jejich přiděleným tématům (každý dokument má svoje identifikační číslo), zda dokument zapíše jako transakci do souboru Database.txt, nebo do souboru nezarazene.txt. Aby byl následný výpočet (natrénování a klasifikace) co nejrychlejší, a v případě trénovací fáze zajištěna funkčnost, je každá transakce složena pouze z jedinečných položek (žádná položka se v transakci nesmí opakovat). Také jsou vyjmuta taková slova, která jsou obsažena ve slovníku nepotřebných slov (5.3), a taková jejichž délka je menší než 3 znaky.

Trénovací dokumenty, které jsou zařazeny do více témat najednou, jsou z procesu trénování vyjmuty a jsou uloženy jako nezařazené.

5.3 Vytvoření slovníku nepotřebných slov

Slovník nepotřebných slov zrychluje algoritmus a také snižuje nároky na paměť. Je také vhodný pro odstranění slov, která se vyskytují příliš často a proto nejsou vhodná jako frekventovaná, neboť jejich váha je podle vzorce 4.2 podobná pro všechna témata a nemají tak na výslednou klasifikaci žádný vliv.

Vytvoření slovníku nepotřebných slov probíhalo ve dvou fázích, a to:

1) Všechna slova (čísla), která byla ve více jak 500 dokumentech (a v různých třídách) byla označena za příliš častá.

2) Všechna slova (čísla), která byla pouze v jediném dokumentu v rámci kolekce Reuters 21578 [RC], a nemohla být tedy považována za frekventované položky, byla označena jako nepotřebná.

Původní slovník nepotřených slov měl více jak 12000 položek a při předzpracování byla bohužel celková časová náročnost kolem 7 minut na testovací konfiguraci, neboť každé slovo z dokumentu bylo vyhledáváno ve slovníku nepotřebných. Nyní má slovník pouze 1574 položek, a to takových, které se vyskytují pouze jednou v rámci celé kolekce Reuters 21578 [RC], a jsou 3 písmenné (číselné) plus takové které se vyskytují ve více jak 500 dokumentech a mají 4 znaky (čtyřpísmenná slova). Čas potřebný pro předzpracování Reuters 21578 [RC] se snížil na maximálně 70 vteřin a čas ve fázi učení je také již uspokojivý. Ve slovníku se vyskytují také některé dvoupísmenné položky, které ale vzhledem k následnému omezení v programu, na slova skládající se ze 3 a více písmen, nemají vliv na běh algoritmu.

Vliv na rychlost a kvalitu klasifikace v závislosti na velikosti slovníku nepodstatných slov v kapitole 6.

5.4 Manuál k programu

Tato kapitola slouží jako návod k použití programové části této práce. Samotný program lze spustit přeloženým souborem ***itemsets.exe*** (překlad pro platformu win32), nebo lze přeložit zdrojový soubor itemsets.cpp jakýmkoliv vhodným c++ překladačem.

Po spuštění program vytvoří „dosovské“ okno, ve kterém je možno průběžně sledovat jednotlivé fáze programu včetně času, za který byly tyto fáze dokončeny (viz kapitola 2.1 o předzpracování, kapitola 4.2.1 o fázi učení a kapitola 4.2.2 o fázi klasifikace). Po dokončení klasifikace program čeká na stisk klávesy. Veškeré informace vypisované do okna, kromě případných chybových hlášení, jsou také uloženy do souboru vysledek.txt (kap. 5.1.2.1).

Je nezbytně nutné, aby soubory, které program používá pro vstup a odpovídají aktuální potřebě závislé na nastavení v souboru config.txt (kap. 5.1.1.1), se nacházely ve stejném adresáři jako spustitelný program. Tabulka na obr. 5.1 uvádí konkrétní příklady nastavení některých parametrů v config.txt v souvislosti s potřebou souborů v adresáři se spustitelným programem pro jeho bezchybnou, a nejspíše i očekávanou, funkčnost:

Konfigurační parametry			Je třeba načíst soubor (ANO/NE)						
K_S_R	Test	P_DB	a	b	c	d	e	f	g
1až22	1	0	ANO	NE	NE	NE	NE	ANO	ANO
1až22	1	1	ANO	ANO	ANO	NE	NE	NE	NE
1až22	0	0	NE	NE	NE	ANO	NE	ANO	ANO
1až22	0	1	NE	NE	ANO	ANO	ANO	NE	NE

Kde:

K_S_R je hodnota nastavení *kolik_souboru_reuters_zpracovat*

Test je *test*

P_DB je *predzpracovavat_databazi*

a je soubor test.read_d

b je soubor test.sgm

c je soubor slovnik_nepotrebnych.txt

d je soubor cate90_train_rel.read_d

e jsou soubory reut2-000.sgm až reut2-021.sgm

f je soubor Database.txt

g je soubor nezarazene.txt

Obr. 5.1 : tabulka potřeby přítomnosti souborů na aktuálním nastavení v config.txt

Tabulku doplním o fakt, že soubor config.txt je pro funkčnost programu samozřejmě nezbytný. Také je v tabulce patrný údaj o zadaném počtu souborů z kolekce Reuters 21578 [RC], kterýžto není z prostorových důvodů rozepsán do více řádků, a je tedy logické, že pokud zadáme například hodnotu 4 (*kolik_souboru_reuters_zpracovat: 4 v config.txt*), pak není nutné, aby v době běhu algoritmu bylo přítomno všech 22 souborů, ale pouze první 4.

Pokud je nastaveno, že nemá dojít k fázi předzpracování, pak program předpokládá, že existují soubory Database.txt a nezarazene.txt. Příslušnost dokumentů k tématům je vybrána podle nastavení, zda jde o test nebo je klasifikována kolekce Reuters. Z tohoto důvodu je nutné, aby byla možnost nepředzpracování databáze vybrána pouze, pokud jsme si jistí, že to opravdu nechceme a soubory Database.txt a nezarazene.txt obsahují takové dokumenty, které odpovídají souboru s příslušností dokumentů k tématům. (Toto nastavení vzniklo pro potřebu zrychlení práce při experimentech, kdy je potřeba spustit program několikrát s různými nastaveními, která nemají vliv na fázi předzpracování.)

Dalším důležitým nastavením programu, ovlivňujícím celkové chování programu, v konfiguračním souboru (5.1.1.1), jsou nastavení minsup (minimální podpora – kapitola 4.1.1) a metoda_zmeny_minsup (4.1.4).

Při nastavení zda použít metodu proměnné minimální podpory na ano (1), je hodnota minsup použita jako výchozí nejnižší hodnota minimální podpory a algoritmus pouze může aktuální hodnotu použitou při výběru frekventovaných položek zvyšovat dle maximální velikosti pole, do kterého jsou výsledné frekventované množiny zapsány z množiny kandidátních množin prvků.

Naproti tomu při „vypnutí“ metody proměnné minimální podpory může dojít při nevhodně zvolené hodnotě minimální podpory snadno k přetečení pole a program o této události pouze informuje a do pole další prvky nezapisuje. Výsledkem je získání, vždy v každém kroku Apriori algoritmu [8], pouze prvních x frekventovaných množin, splňujících podmínku minimální podpory, a vynechání dalších byť frekventovanějších množin. To vede samozřejmě ke špatnému natrénování klasifikátoru a špatné klasifikaci.

Jak již bylo uvedeno, program má v první řadě za cíl vygenerovat soubor vysledek.txt (kap. 5.1.2.1), ze kterého můžeme přečíst všechny potřebné údaje získané aplikací programu na databázi.

5.5 Užité programové prostředky

Pro tuto práci byl nakonec vybrán programovací jazyk C, neboť v něm implementovaný Apriori algoritmus [8] byl snadno modifikovatelný jednak pro lepší využití paměti počítače a zároveň zrychlení pomocí převodu položek na čísla typu int s využitím jediné tabulky jedinečných položek.

Jako programovací prostředí byl vybrán program MinGW Developer Studio verze 2.01 v prostředí Microsoft Windows XP professional SP4.

5.6 Dokumentace k programové části

Program se skládá z pěti částí (souborů). Hlavní soubor, se zdrojovým kódem, je soubor itemsets.cpp, který mimo funkce pro řazení obsahuje také hlavní funkci main. Ostatní čtyři soubory zdrojového kódu jsou pouze vloženy (include) do hlavního souboru itemsets.cpp a obsahují pouze příslušné funkce, definice konstant a globálních proměnných. Program využívá těchto standartních knihoven jazyka c: stdio.h, ctype.h, string.h, stdlib.h a time.h.

Soubor konstanty.h obsahuje definice potřebných konstant. Jsou to jednak konstanty používané implementovaným lexikálním analyzátorem, a také jsou to omezující konstanty pro potřeby definic polí a příslušných používaných omezení. Soubor promene.h obsahuje deklarace potřebných polí a globálních proměnných. Komentáře ve zdrojových souborech a stejně tak názvy jednotlivých konstant a proměnných jsou natolik výmluvné, že není potřeba zde znovu jednotlivé proměnné a konstanty zmiňovat a popisovat.

Soubor zpracuj_reuters.c obsahuje deklarace funkcí použitých během fáze předzpracování. Soubor apriori.c obsahuje funkce použité pro získání množin častých položek (podrobně v [8]).

Rozsah programové části (samotného kódu) je možné vyčíst v kapitole 5.6.1. Jak program pracuje je popsáno v kapitole 5.6.2. Ty nejdůležitější funkce, jejichž činnost nemusí být na první pohled do zdrojového kódu jasná, jsou pak podrobně popsány v 5.6.3. Jelikož byl výpočet v původní podobě (přesně podle kapitoly 4) programu příliš pomalý, bylo přidáno několik optimalizací, popsaných v kapitole 5.6.4.

5.6.1 Rozsah programové části

Soubor konstanty.h má 31 řádků a definuje celkem 32 konstant. Soubor promene.h má 135 řádků a deklaruje celkem: 10 ukazatelů na soubor, 5 polí typu char, 40 proměnných typu int, 5 proměnných typu short, 14 polí typu short, 1 proměnnou typu float a jedno dvourozměrné pole typu float.

V souboru zpracuj_reuters.c je deklarace celkem devíti funkcí a soubor má 719 řádků.

Soubor apriori.c obsahuje deklaraci 16-ti funkcí a má 571 řádků.

Soubor itemsets.cpp obsahuje 11 deklarací funkcí, plus hlavní funkci main, a má celkem 822 řádků.

Celkově má kód programu 2278 řádků a 37 funkcí včetně funkce main. Funkce a jejich popis v kapitole 5.6.3. Funkce main je popsána v kapitole 5.6.2.

5.6.2 Jak program pracuje

V této kapitole je popsán způsob práce programu po jeho spuštění. Jde hlavně o to shrnout, jaké funkce jsou volány a v jakém pořadí a co je jejich účelem během samotného výpočtu. Bližší popis jak jednotlivé funkce pracují viz kapitola 5.6.3.

Jak již bylo řečeno v úvodu této kapitoly, funkce main, volaná jako první po spuštění programu, se nachází v souboru itemsets.cpp.

První funkcí, která je volaná ve funkci main, je funkce napln_pole_nepodstatnych_slovicek. Tato funkce přečte ze souboru slovnik_nepotrebnych.txt všechna slova a naplní jimi pole tabulka_nepodstatnych_slov. Pokud není soubor nalezen, nebo nelze otevřít, pak program vypíše upozornění, že nemohl tento soubor otevřít, ale není ukončen. Výpočet tedy může pokračovat i bez slovníku nepotřebných slov (viz 2.1.2 a 5.1.1.2).

Dalším krokem je pokus otevřít, nebo vytvořit soubor vysledek.txt (viz 5.1.2.1). V případě že se nepodaří soubor otevřít nebo vytvořit, je program ukončen.

V následujícím kroku je volána funkce nacti_konfigurak, která se pokusí přečíst data z konfiguračního souboru config.txt (viz 5.1.1.1), a v případě že se jí to nepodaří, přiřadí příslušným proměnným výchozí hodnoty a vypíše chybovou zprávu.

Po načtení konfiguračního souboru program podle hodnoty proměnné predzpracovavat_databazi provede:

a) V případě, že je hodnota této proměnné rovna 0. Načtení názvu témat (naplněním pole tabulka_nazvu_temat) a naplnění pole příslušnosti trénovacích dokumentů k tématům (pole tabulka_zarazenych_z_reuters_cislo_doc_a_cislo_tematu) ze souboru test.read_d (viz 5.1.1.5), nebo ze souboru cate90_train_rel.read_d (viz 5.1.1.3), zavoláním funkce, deklarované v zpracuj_reuters.c, napln_pole_s_tematy_a_cisly_dokumentu.

Nebo b) V případě, že je hodnota této proměnné rovna 1. Začne měřit čas. Zavolá funkci reuters_na_vstupni_data, která předzpracuje databázi. To znamená, že provede načtení ze souboru uvedené v možnosti a (predzpracovavat_databazi = 0). A dále načítá jednotlivé soubory databáze. To záleží na tom, zda se jedná o test, nebo zda zpracováváme kolekci Reuters 21578. Také je důležité, kolik souborů z reuters si přejeme zpracovat. (Viz config.txt kap. 5.1.1.1). Vždy je otevřen jeden soubor a na ten je poté aplikována funkce zpracuj_jeden_soubor_z_reuters. Po zpracování souboru je tento neprodleně uzavřen. Výsledkem funkce reuters_na_vstupni_data je vytvoření souborů Database.txt (kapitola 5.1.2.2) a nezarazene.txt (kapitola 5.1.2.3). Následuje ukončení měření času za který byla fáze předzpracování hotová, a výpis informace o ukončení fáze předzpracování a naměřený čas do okna programu a také do souboru vysledek.txt (viz 5.1.2.1).

Po dokončení fáze předzpracování program začne opět měřit čas a pokusí se otevřít vstupní soubor Database.txt (kapitola 5.1.2.2), obsahující trénovací transakční databázi. V případě, že se mu to nepodaří, program skončí. V opačném případě následuje volání funkcí Apriori algoritmu [8].

Nejprve je naplněno pole jedinečných položek a načtení trénovací databáze do paměti (naplněním pole trenovaci_database_num), voláním funkce najdi_jedinecne_polozky. Tato funkce přečte ze souboru Database.txt všechna slova oddělená mezerou, tabelátorem nebo koncem řádku, a pomocí implementovaného lexikálního analyzátoru rozpoznává položky, konce transakcí a čísla témat, do kterých jsou zařazeny jednotlivé transakce a ukládá výsledky analýzy do polí

trenovaci_database_num a tabulka_jedinecnych_polozek. Také naplňuje pole tabulka_Ck_itemset_se_vyskytuje_v_transakcich, které slouží dále pro výpočet vah, pole tabulka_poctu_vyskytu_jedinecnych_polozek, které slouží k vyhodnocení, zda je položka frekventovaná.

Následuje volání funkce vytvor_v_Fk_jednoprvkove_mnoziny, která na základě hodnoty minimální podpory naplní pole mnozina_Fk čísly položek, jejichž výskyt je častější, nebo roven, než je hodnota minimální podpory. Funkce pridej_jednoprvkove_do_itemsets pouze překopíruje čísla frekventovaných položek do výsledného pole všech položek mnozina_vsech_Fk_tzv_itemsetu.

Nyní jsou generovány v cyklu, za předpokladu že je zvoleno řazení za použití víceprvkových itemsetů v config.txt (kap. 5.1.1.1) nastavení max_polozek_v_transakci_omezeni_apriory, střídavě kandidátní množiny prvků, voláním funkce vytvor_Ck_z_Fk, a z této množiny jsou poté, za pomocí funkcí spocti_vyskyty_v_DB_mnozin_z_Ck a vytvor_Fk_z_Ck_podle_podpory, vygenerovány frenkventované množiny prvků. Tyto frekventované, ještě před testem dalšího proběhnutí cyklu while, jsou překopírovány funkcí pridej_Fk_do_itemsets do pole mnozina_vsech_Fk_tzv_itemsetu. Samozřejmostí jsou informativní výpisy do souboru vysledek.txt (5.1.2.1) a do okna programu, a také kontrola hlídajících proměnných na přetečení, polí pokud se jedná o klasický Apriori algoritmus bez proměnné hodnoty minsup.

Po ukončení, nebo přeskočení cyklu while, je volána funkce spocitej_vahy. Tato funkce vypočítá příslušné váhy pro každý itemset a každou třídu (téma) podle vzorce z kapitoly 4.2.1 a naplní jimi pole tabulka_vah_itemsetu. Po dokončení je ukončeno měření času, neboť fáze učení je hotová. Příslušný čas, v sekundách, je opět vypsán do okna a do souboru vysledek.txt (viz 5.1.2.1).

Program uzavře soubor Database.txt (kapitola 5.1.2.2) a vypíše hlášení o celkovém počtu itemsetů a začne opět měřit čas. Následuje totiž fáze samotné klasifikace.

Nyní se program pokusí otevřít soubor nezarazene.txt (kap 5.1.2.3) s nezařazenými dokumenty ve formě transakční databáze (kap 2.2.2 definice 2.1). Pakliže se mu to nepodaří, je předčasně ukončen. V opačné případě je zavolána funkce klasifikuj. Tato funkce vždy načte ze souboru jednu transakci do pomocného pole tak, že vybere jedinečné položky, a to pouze takové, které jsou označeny ve fázi učení jako frekventované (kapitola 5.6.4 o optimalizaci). Spočítá váhy pro každé téma za pomoci nalezených itemsetů, nalezených v tomto nezařazeném dokumentu, a na základě parametrů, načtených z konfuguračního souboru, dokument označí v souboru vysledek.txt (5.1.2.1) buď jako nezařazený, nebo vypíše témata, do kterých tento soubor patří (splňují podmínky dané nastavením – výsledná váha je větší nebo rovna x procentní hodnotě největší váhy ze souboru config.txt (5.1.1.1)).

Po ukončení funkce klasifikuj je ukončeno měření času a opět je vypsán čas, za který byla klasifikace dokumentů do tříd dokončena. Program nyní uzavře všechny zbývající otevřené soubory a čeká na stisk klávesy.

5.6.3 Popis funkcí programu

Zde je popsáno k čemu slouží jednotlivé funkce programu a u některých složitějších je navíc naznačeno, jak pracují. Jednotlivé funkce jsou seřazeny podle toho, ve kterém zdrojovém souboru se nacházejí.

Funkce používané při předzpracování (jejichž definice se nachází v souboru zpracuj_reuters.c) jsou:

void napln_pole_nepodstatnych_slovicek();

- tato funkce otevře soubor slovnik_nepotrebnych.txt a všechna slova z něj překopíruje do pole tabulka_nepodstatnych_slov (viz 5.1.1.2).

int reuters_hledej_polozku();

- funkce vyhledává příslušnou, právě přečtenou položku v polích nepodstatných slov a jedinečných slov. V případě že slovo má méně znaků než je přípustné a vrací číslo 1. Pokud slovo nalezne v některém z polí vrátí index na kterém se daná položka nachází. V případě že položka se nenachází ani v jednom z polí a má více znaků než je minimum, vrací funkce hodnotu –1.

int reuters_vloz_polozku();

- funkce zavolá funkci reuters_hledej_polozku, a pokud funkce vrátí hodnotu –1, uloží tuto aktuální položku do pole jedinečných položek právě zpracovávaného dokumentu.

int reuters_minilex();

- pomocná funkce lexikálního analyzátoru (pro předzpracování souborů s databází ve formátu kolekce Reuters 21578). Vrací buď TAG_BODY_END, TAG_BODY, TAG_REUTERS_END, TAG_NEWID, POLOZKA, LOMITKO nebo KONEC, což jsou pomocné konstanty lexikální analýzy. V případě chyby vrátí funkce konstantu NIC.

int reuters_lex();

- pomocná funkce pro zachycení případných chyb v databázi. Pouze vrací výsledek funkce reuters_minilex, pokud je různý od hodnoty pomocné konstanty NIC.

void zpracuj_jeden_soubor_z_reuters();

- funkce, která postupně prochází jeden aktuálně pro čtení otevřený soubor databáze a třídí a zároveň zpracovává jednotlivé dokumenty do souborů Database.txt (kap. 5.1.2.2) a nezarazene.txt (kapitola 5.1.2.3).

- v cyklu while zavolá vždy funkci reuters_lex, dokud nenarazí na konec souboru, a podle toho jaký tato funkce vrátí token, se provádí příslušné operace.

int hledej_tema_v_nazvech();

- funkce vrací buď –1, v případě že název aktuálně načteného tématu není v tabulce tabulka_nazvu_temat nalezen, a nebo identifikační číslo tématu, v případě že téma již v tabulce je.

void napln_pole_s_tematy_a_cisly_dokumentu();

- tato funkce je poněkud komplikovanější, neboť nevyužívá žádné další pomocné funkce lexikální analýzy. Funkce má za úkol načíst čísla dokumentů a názvy témat, do kterých patří, ze souboru cate90_train_rel.read_d (viz 5.1.1.3) nebo test.read_d (viz 5.1.1.5) podle nastavení config.txt (viz 5.1.1.1).

Výsledkem funkce je naplnění pole tabulka_zarazenych_z_reuters_cislo_doc_a_cislo_tematu.

- funkce nejprve otevře soubor cate90_train_rel.read_d (nebo test.read_d), a pokud se jí to nepodaří, tak ukončí program. Následuje komplikovanější zápis několika cyklů while, kde se provádí vždy na novém řádku (v souboru) načtení čísla dokumentu, a poté načtení názvu tématu. V případě že je dokument zařazen do více témat, je automaticky ignorován (pro účel natrénování). Při přečtení názvu tématu je volána funkce hledej_tema_v_nazvech, a pokud není toto právě načtené téma nalezeno, je uloženo do pole názvů témat jako nové. Poté co je naplněno pole tabulka_zarazenych_z_reuters_cislo_doc_a_cislo_tematu daty získanými z tohoto souboru, je provedena ještě korekce na základě konfiguračního souboru. Jde o to, všechny soubory zařazené do témat, která nemají dostatečný počet trénovacích dokumentů, označit jako nezařazená. To je snadno vyřešeno pomocí cyklu for a pomocného pole tabulka_poctu_doc_v_tematu.

int reuters_na_vstupni_data();

- tato funkce volá funkci napln_pole_s_tematy_a_cisly_dokumentu. Dále má za úkol otevřít soubory Database.txt a nezarazene.txt pro zápis, a také podle nastavení v konfiguračním souboru zajišťuje otevření a uzavření souborů databáze (nepředzpracovaných) a volání funkce zpracuj_jeden_soubor_z_reuters.

- Podle hodnoty proměnné kolik_souboru_reuters_zpracovat vždy otevře (počítáno od prvního) jeden soubor z kolekce Reuters21578 pro čtení, zavolá funkci zpracuj_jeden_soubor_z_reuters, a poté tento soubor uzavře.

Funkce používané během fáze učení (jejichž definice se nachází v souboru apriori.c) jsou:

int hledej_polozku();

- funkce vyhledává příslušnou, právě přečtenou položku polí jedinečných slov. V případě, že ho nalezne, vrátí id na kterém se daná položka nachází. V opačném případě vrací funkce hodnotu –1.

int vloz_polozku();

- funkce se snaží vložit aktuálně načtenou položku (slovo) do pole jedinečných položek. Pokud je již položka v poli nalezena, je pouze inkrementovana příslušná pozice v poli počtu výskytů jedinečných slov. V opačném případě je navíc toto slovo zapsáno do pole tabulka_jedinecnych_polozek a funkce vrací hodnotu 1 jinak –1.

int minilex();

- funkce je součástí lexikálního analyzátoru pro potřeby načtení trénovací databáze. Je obdobná funkci reuters_minilex, s tím rozdílem že je použita na již předzpracovanou databázi ve formátu transakční databáze, a tudíž je jednoduší.

int hlavni_lex();

- stejná funkce jako je reuters_lex, ale nevolá funkci reuters_minilex, nýbrž funkci minilex.

void vloz_polozku_do_DB();

- funkce vloží id číslo do pole trenovaci_database_num na příslušnou pozici.

int hledej_tema(int cislotematu);

- funkce prochází pole tabulka_temat, a pokud číslo parametru odpovídá některému číslu z toho pole, vrátí funkce id číslo tohoto tématu a v poli tabulka_kolik_je_transakci_v_tematu inkrementuje danou pozici (pro zjištění kolik má téma dokumentů použitém při výpočtu vah). Pokud číslo tématu není nalezeno, pak vrací funkce –1.

void pridej_tema(int cislotematu);

- Tato funkce používá funkci hledej_téma a slouží k přidání nových čísel témat z trénovací databáze.

void najdi_jedinecne_polozky();

- Funkce najde jedinečné položky trénovací databáze, spočte jejich podporu a „nahraje“ trénovací databázi DB do pole trenovaci_database_num.

int zmen_minsup_podle_potreby_pro_jednoprvkove(int old_minsup);

- funkce spočítá podle zadané hodnoty minimální podpory (v config.txt) počet frekventovaných prvků z množiny jedinečných položek. V případě že tento počet je větší, než dovolují možnosti definovaného pole frekventovaných množin, hodnotu minimální podpory inkrementuje. Počítání frekventovaných položek a následnou případnou inkrementaci, provádí dokud není počet frekventovaných položek menší, nebo rovný, dané mezi. Novou hodnotu minimální podpory poté funkce vrátí.

void vytvor_v_Fk_jednoprvkove_mnoziny();

- funkce nakopíruje čísla frekventovaných položek získaných podle hodnoty minimální podpory do pole mnozina_Fk.

void pridej_jednoprvkove_do_itemsets();

- funkce nakopíruje čísla frekventovaných položek z pole mnozina_Fk do pole mnozina_vsech_Fk_tzv_itemsetu. Také dopní příslušnými daty pole mnozina_vyskytu_itemsetu_v_transakcich, určené pro výpočet vah.

void spocti_vyskyty_v_DB_mnozin_z_Ck();

- funkce spočítá pro každou množinu z pole mnozina_Ck počet jejího výskytu v transakcích trénovací databáze. Výsledek zapíše do pole tabulka_poctu_vyskytu_Ck_mnozin_v_DB a také do pole, ve kterých transakcích se daná kandidátní množina vyskytuje.

void pridej_Fk_do_itemsets();

- Tato funkce stejně jako funkce pridej_jednoprvkove_do_itemsets přidává čísla frekventovaných množin z pole mnozina_Fk do pole mnozina_vsech_Fk_tzv_itemsetu. Na rozdíl od předchozí funkce to dělá pro víceprvkové itemsety a ne jenom jednoprvkové.

int zmen_minsup_podle_potreby_pro_k_prvkove(int old_minsup);

- funkce funguje a dělá přesně to samé, co funkce zmen_minsup_podle_potreby_pro_jednoprvkove, avšak používá při tom jiná pole, neboť je použitá v jiné části Apriori algoritmu, kde již nepočítáme s jednoprvkovými množinami.

void vytvor_Fk_z_Ck_podle_podpory();

- tato funkce vybere frekventované množiny prvků z kandidátních podle hodnoty minimální podpory. V případě že je v konfiguračním souboru možnost použití metody proměnné minimální podpory rovna jedné, je před vybráním frekventovaných množin zavolána funkce zmen_minsup_podle_potreby_pro_k_prvkove.

void vytvor_Ck_z_Fk();

- funkce se pokusí vytvořit z množiny frekventovaných k-prvkových množin množinu kandidátních (k+1)-prvkových množin (podrobně o Apriori algoritmu v [8]).

Funkce používané během fáze učení (jejichž definice se nachází v souboru apriori.c) jsou:

void spocitej_vahy();

- funkce spočítá váhy pro každý itemset a pro každé téma a výsledky zapíše do pole tabulka_vah_itemsetu.

int minilex2();

- pomocná funkce lexikálního analyzátoru pro načtení nezařazených dokumentů ze souboru nezarazene.txt (viz 5.1.2.3).

int lex2();

- stejná funkce jako je reuters_lex, ale nevolá funkci reuters_minilex, nýbrž funkci minilex2.

int je_cislo_polozky_ve_frekventovanych_jednoprvkovych(int cislo_polozky);

- funkce hledá v množině všech jednoprvkových itemsetů právě načtenou položku, a pokud ji nenajde, vrátí 0, jinak vrátí 1.

int nacti_nezarazenou_transakci();

- Fukce načte ze souboru nezařazených dokumentů jeden dokument. Čte položku za položkou a každou testuje, zda je frekventovaná a jestli je, pak ji přidá do pole pole_jedna_nezarazena_transakce, jinak ji ignoruje. V případě že se položka již jednou v dokumentu vyskytla, je také ignorována (Tento stav může nastat použitím jiného způsobu vytvoření souboru nezarazene.txt než použitím tohoto programu, který při fázi předzpracování opakování položek v transakci vylučuje).

int je_itemset_v_dokumentu(int ktery_itemset);

- Funkce hledá itemset v aktuálně řazeném dokumentu, a pokud je přítomen vrátí 1, v opačném případě –1.

void klasifikuj();

- Funkce spočítá výsledné váhy, podle vzorce z kapitoly 4.2.2 pro daný dokument pro všechna témata. Podle parametrů v kofiguračním souboru pak rozhodne a výsledek zapíše do souboru vysledek.txt (viz 5.1.2.1), do kterých témat, a zda vůbec, dokument patří.

int minilex3();

- Pomocná funkce lexikální analýzy, pro přečtení nastavení ze souboru config.txt (viz 5.1.1.1).

int lex3();

- stejná funkce jako je reuters_lex, ale nevolá funkci reuters_minilex, nýbrž funkci minilex3.

int nacti_jednu_konfiguraci();

- Funkce načte pomocí volání funkce lex3 jednu číselnou hodnotu nastavení z konfiguračního souboru. Toto načtené „číslo“ převede na číslo typu int a vrátí jako výsledek.

void nacti_konfigurak();

- Funkce se pokusí o otevření konfuguračního souboru a načtení příslušných hodnot nastavení. Jednotlivá nastavení ukládá do dále v programu používaných proměnných. V případě chyby nebo neúspěchu při otevírání souboru jsou použita výchozí nastavení a program pokračuje dále. Nakonec konfigurační soubor funkce uzavře.

int main();

- funkce main byla již podrobně popsána v kapitole 5.6.2.

5.6.4 Optimalizace programu

V této kapitole je zmíněno několik optimalizací použitých v programu, které zrychlují (a snižují paměťové nároky) bez ztráty funkčnosti celý výpočet, oproti původní verzi, jejíž implementace kopírovala doslovně popis Apriori algoritmu v kapitole 4.1.3, nebo také v [8]. Optimalizace se samozřejmě netýkají pouze Apriori algoritmu, ale je jich využito v rámci možností téměř všude.

První velkou změnou, oproti původní verzi, byl převod slov na čísla. To znamená, že slova jsou načtena do pomocného pole a dále se počítá pouze s jejich příslušnými identifikačními čísly. Tato změna si vyžádala změnu pomocných polí z typu char na typ short. Což vedlo k několikanásobnému urychlení (místo porovnávání položek znak po znaku je pouze porovnáno id, …) a také snížení paměťových nároků programu (třírozměrná pole se změnila na dvourozměrná a dvourozměrná na jednorozměrná).

Další optimalizace spočívá v zápisu nezařazených dokumentů do formátu stejného jako je formát trénovací transakční databáze. To znamená, že jsou z dokumentů odstraněna nepotřebná slova a také je vytvořena transakce složená z jedinečných položek. Transakce, ve které se položky neopakují, je mnohem rychleji zpracována než ta, ve které se musí vyhodnocovat položky několikrát.

Poslední optimalizace se stejně jako předešlá týká fáze samotného řazení, která není příliš patrná v případě malého počtu řazených dokumentů, ale v případě například kolekce Reuters 21578 jde již o minuty rozdílu v rychlosti. Spočívá v tom, že při načtení dokumentu určeného k řazení, se ignorují slova, která nejsou frekventovaná. Díky tomu je následné hledání itemsetů v dokumentu mnohem rychlejší, neboť se tím rapidně sníží počet slov dokumentu.

6 Testování metody itemsets

Kapitola se věnuje testování původní metody itemsets (popsané v [9]) a srovnáním jejích výsledků s použitím modifikovaného apriori algoritmu. Modifikace Apriori algoritmu [8] proměnnou minimální podporou viz kapitola 4.1.4. Pro testování byla použita celosvětově známá, a pro testování klasifikačních algoritmů nejvíce využívaná, kolekce Reuters 21578 [RC]. Testy probíhaly na počítači, jehož konfigurace je uvedena v kapitole 6.1.

6.1 Testovací konfigurace

Počítač, na kterém byl program testován (a zároveň vytvořen), měl tyto pro vyhodnocení rychlosti klasifikace podstatné parametry:

Procesor: Intel Celeron D 315 (2.26GHz)
Paměť: 512MB DDR 400
OS: Microsoft Windows XP professional SP4

6.2 Časové a paměťové nároky metody itemsets

Testování na potřebu paměti a času, za který program provedl klasifikaci, byl proveden na již (v kapitole 6.1) zmiňované konfiguraci počítače. Byla použita celá kolekce Reuters 21578 [RC] s omezením na 10 hlavních tříd. To znamená jen takové třídy, které mají více jak 90 dokumentů. Celkem bylo použito pro natrénování klasifikátoru 3001 dokumentů a klasifikováno bylo 19043 dokumentů.

Minimální podpora měla hodnotu 2 pro metodu proměnné minimální podpory. Pro klasický Apriori algoritmus [8] byla hodnota minimální podpory zvolena na základě výsledků testů s použitím metody proměnné minimální podpory tak, aby počet výsledných itemsetů byl téměř shodný. Vzhledem k faktu, že použití metody změny minimální podpory využívá maximálně možností pomocných polí, je paměťová a časová náročnost větší než u klasického Apriori algoritmu. Minimální možná váha byla nastavena na 0.000020 a minimální procento z maximální možné váhy pro zařazení bylo nastaveno na 75%.

Naměřené hodnoty experimentů při použití malého, většího a velkého slovníku nepotřebných slov (viz 5.3) pro metodu proměnné minimální podpory udává tabulka 6.1. Také je zde uveden vliv na čas a paměť při omezení se pouze na jednoprvkové itemsety. V tabulce 6.2 jsou uvedeny výsledky při použití klasického Apriori algoritmu [8]. Závislost počtu slov ve slovníku nepotřebných na čase

potřebného na fázi předzpracování je pak v grafu na obrázku 6.1. Závislost počtu slov ve slovníku nepotřebných na čase potřebného na fázi klasifikace je pak v grafu na obrázku 6.2.

Nepotřebných slov	Omezení Apriori algoritmu na k prvkové itemsety (max k)	Potřebná paměť maximálně (kB)	Čas fáze:		
			Předzpracování (s)	Učení (s)	Klasifikace (s)
10711	-	183408	410	41	165
1574	-	187988	65	46	230
0	-	181232	13	62	352
10711	1	56152	410	8	134
1574	1	66204	65	11	198
0	1	75780	13	17	276

Tabulka 6.1: Tabulka časových a paměťových nároků programu pro modifikovaný Apriori

Paměť, potřebná pro fázi předzpracování, byla téměř stejná pro všechny testované slovníky nepotřebných slov (viz 5.3), a to kolem 1000 kB (maximálně 1056 kB).

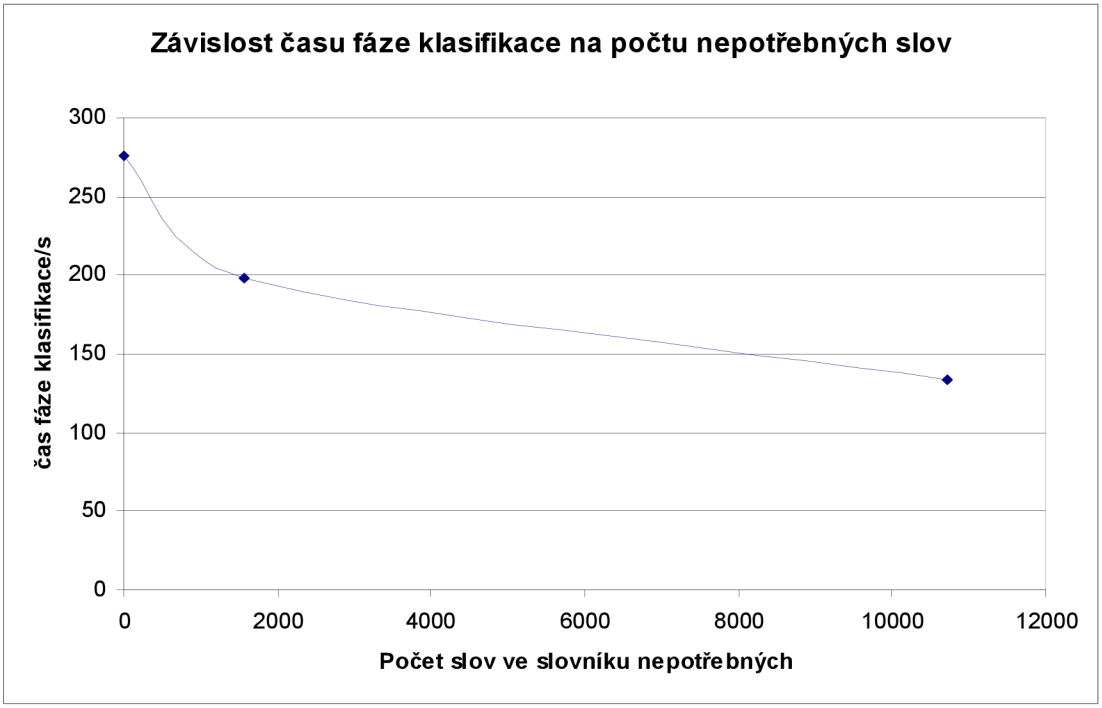
Z tabulky 6.1 je vidět, že maximální spotřeba paměti byla menší než 190 MB a nejméně paměti spotřebuje program při řazení za pomoci pouze jednoprvkových itemsetů a za pomoci velmi obsáhlého slovníku nepotřebných slov.

Z tabulky 6.1 také jednoznačně vyplývá fakt, že nejrychlejších výsledků program dosahuje při použití slovníku nepotřebných slov vytvořeného způsobem popsaným v kapitole 5.3. Samozřejmě, že nejrychlejšího času během fáze učení a samotné klasifikace dosáhneme použitím úplného (tedy největšího) slovníku nepotřebných slov, ale to na úkor fáze předzpracování, která oproti slovníku s menším počtem slov trvá několikanásobně delší dobu.

minsup	Nepotřebných slov	k prvkové itemsety (max k)	Potřebná paměť maximálně (kB)	Čas fáze:		
				Předzpracování (s)	Učení (s)	Klasifikace (s)
143	10711	-	76232	410	16	146
173	1574	-	85468	65	19	208
442	0	-	82424	13	22	293
90	10711	1	56152	410	8	134
98	1574	1	66204	65	11	198
126	0	1	75780	13	17	276

Tabulka 6.2: Tabulka časových a paměťových nároků programu pro klasický Apriori

Obrázek 6.1: Graf závislosti času potřebného k předzpracování na počtu nepotřebných slov



Obrázek 6.2: Graf závislosti času potřebného ke klasifikaci na počtu nepotřebných slov

Nyní je tedy z výsledků zřejmé, že vytvoření správného slovníku nepotřebných slov je pro rychlost předzpracování a fázi klasifikace klíčovým faktorem. Rychlost fáze učení a spotřeba paměti závisí na počtu itemsetů. Ten můžeme ovlivnit zadanou hodnotou minimální podpory.

6.3 Vliv hodnoty minimálních procent z maximální váhy na výsledek řazení

V tomto testu se snažíme zjistit vliv při zachování ostatních nastavení, hodnoty procent minimální váhy z maximální nalezené na výsledky řazení. Maximální počet položek itemsetu necháme neomezený. Hodnota minimální podpory pro klasický Apriori algoritmus [8] byla nastavena podle výsledků testů s použitím metody proměnné minimální podpory (kap 4.1.4).

Minsup	Metoda změny minsup	Minimálně % z maximální váhy	Itemset má max k prvku	P (%)	R (%)	U (%)
2	Ano	10	-	66.20	81.31	73.75
2	Ano	30	-	77.33	76.84	77.09
2	Ano	50	-	80.06	75.04	77.55
2	Ano	75	-	82.26	74.18	78.22
2	Ano	80	-	82.68	73.98	78.33
2	Ano	90	-	83.76	73.38	78.57
2	Ano	100	-	84.48	72.18	78.33
173	Ne	10	-	73.28	77.24	75.26
173	Ne	30	-	86.34	74.14	80.24
173	Ne	50	-	86.72	73.78	80.25
173	Ne	75	-	88.39	73.08	80.73
173	Ne	80	-	88.40	72.88	80.64
173	Ne	90	-	88.70	72.74	80.72
173	Ne	100	-	88.78	72.74	80.76

Tabulka 6.3: Tabulka závislosti nastavení % z maximální váhy na výsledcích řazení

Z tohoto testu je patrno, podle údajů v tabulce 6.3, že přesnost roste na úkor úplnosti a naopak. Nejlepších výsledků metoda itemsets dosahuje při řazení do tříd, jejichž váha přesahuje 30% největší získané váhy. Pro nižší hodnoty minimální váhy pro řazení je již míra přesnosti příliš nízká. Tento test již ukazuje na nedostatky metody proměnné minimální podpory (kap. 4.1.4), používané během fáze učení -Apriori algoritmu [8].

6.4 Řazení 1-prvkovými|k-prvkovými itemsety

Tento test odhaluje skutečnost již známou z práce [9], a to, že použitím pouze jednoprvkových itemsetů (frekventovaných položek) se klasifikace zrychlí (viz 6.2) a kvalita výsledku není významně horší.

Minsup	Metoda změny minsup	Minimálně % z maximální váhy	Itemset má max k prvků	P (%)	R (%)	U (%)
2(98)	Ano	80	1	81.46	73.78	77.62
2	Ano	80	2	82.68	73.96	78.33
2	Ano	80	3	82.68	73.96	78.33
2	Ano	80	4	82.68	73.96	78.33
2	Ano	80	5	82.68	73.96	78.33
2	Ano	80	-	82.68	73.96	78.33
98	Ne	80	1	81.46	73.78	77.62
173	Ne	80	2	88.40	72.88	80.64
173	Ne	80	3	88.40	72.88	80.64
173	Ne	80	4	88.40	72.88	80.64
173	Ne	80	5	88.40	72.88	80.64
173	Ne	80	-	88.40	72.88	80.64

Tabulka 6.4: Tabulka závislosti maximálního počtu prvků v itemsetu

Test nám také odhalil nevhodnost způsobu počítání minimální podpory v metodě proměnné minimální podpory, který ignoruje apriori vlastnost Apriori algoritmu, v případě že používáme víceprvkové itemsety. Při natrénování klasifikátoru víceprvkovými itemsety dochází ke zhoršení kvality klasifikace právě kvůli nesplnění apriori vlastnosti, která je při následném výpočtu vah brána za samozřejmou.

Přestože bylo v úvodu naznačeno, že význam víceprvkových itemsetů na kvalitu klasifikace je malý, tabulka 6.4 říká něco jiného. Všimněme si, že se rozdíl neprojevuje již při srovnání klasifikace s použitím maximálně jednoprvkových itemsetů s použitím víceprvkových, ale až když srovnáme řazení jedno a dvouprvkových itemsetů s řazením při použití víceprvkových. Je to způsobeno odlišnou hodnotou nastavení minimální podpory, a tím také odlišného počtu jednoprvkových itemsetů, které jsou jak jsme si řekli zjevně nejvýznamnější. Proveďme tedy ještě jeden test při použití maximálně jednoprvkových množin prvků s nastavením minimální podpory na hodnotu 173 (tabulka 6.5).

Minsup	Metoda změny minsup	Minimálně % z maximální váhy	Itemset má max k prvků	P (%)	R (%)	U (%)
173	Ne	80	1	88.40	72.88	80.64

Tabulka 6.5: Tabulka závislosti maximálního počtu prvků v itemsetu – doplňující test

Pokud srovnáme výsledek doplňujícího testu (tabulka 6.5) s tabulkou 6.4 vidíme, že význam víceprvkových itemsetů na klasifikaci byl v našem případě nulový.

6.5 Modifikovaná metoda itemsets versus normální metoda itemsets

Při srovnání původní metody itemsets (popsané v [9]) s její modifikací popsanou v kapitole 4.1.4, vyjdeme z výsledků předchozích testů z kapitol 6.2 až 6.4.

Závěry již popsané lze shrnout do několika následujících bodů, představujících plusy a mínusy modifikované metody oproti původní metodě.

Plus modifikace (kap 4.1.4):

+ Pomocí modifikovaného Apriori algoritmu lze získat pro konkrétní databázi a nastavená omezení programu správnou hodnotu minimální podpory.

+ Nemusíme zjišťovat vhodnou hodnotu minimální podpory. Program funguje pro jakoukoliv zadanou hodnotu minimální podpory bezchybně s relativně dobrými výsledky.

Mínus modifikace (kap 4.1.4):

- Časová náročnost je o něco větší, nežli při použití klasického Apriori algoritmu

- Paměťová náročnost je stejně jako paměťová větší. Je to dáno ale faktem, že modifikovaný aprioty algoritmus najde mnohem více itemsetů nežli klasický Apriori.

- Výsledky klasifikace, myšleno úspěšnost, přesnost a úplnost, jsou horší než v případě původní metody, za předpokladu že použijeme nevhodnou hodnotu minimální podpory.

7 Závěr

V této práci byla navržena, implementována a otestována modifikace klasifikační metody itemsets.

Modifikace metody změny minimální podpory do metody itemsets [9] nepřináší oproti prvotnímu očekávání zlepšení původní metody, ale může být využita pro získání správné hodnoty minimální podpory na konkrétní trénovací databázi. Vzhledem k časové náročnosti fáze učení, jejíž délka se pohybuje v desítkách vteřin, to nepředstavuje významné zpomalení. Aplikovali bychom modifikovaný Apriori algoritmus na databázi s cílem zjištění vhodné hodnoty minimální podpory. Podruhé by byl použit klasický Apriori algoritmus se získanou hodnotou minimální podpory. Při stejné hodnotě minimální podpory se z pochopitelných důvodů bude modifikovaná metoda chovat stejně, a se stejnými výsledky jako metoda původní.

Testy nám znovu prokázaly, a jest jedno zda posuzujeme metodu itemsets modifikovanou nebo původní, naprosto zanedbatelný pozitivní vliv dvouprvkových, tříprvkových, čtyřprvkových, pětiprvkových…. Obecně pak n-prvkových itemsetů, kde n představuje číslo větší než jedna, na relevantnost, úplnost a pochopitelně i úspěšnost klasifikace. Naproti tomu prokázaly výrazně negativní vliv na paměťové a časové nároky programu. Jako nejlepší volba, pro klasifikaci metodou itemsets, je použití jednoprvkových itemsetů.

Vzhledem k lepším výsledkům v posouzení z hlediska relevantnosti je modifikace metody itemsets zajímavá (lepší). Je možné, že při nalezení vhodného nastavení parametrů řazení a jiné metody výpočtu hodnoty minimální podpory, než snaha o co nejnižší hodnotu, bude srovnávací výsledek těchto metod opačný. Také nezapomeňme zmínit nízký vliv víceroprvkových itemsetů, což vede k myšlence odlišného výpočtu vah během fáze klasifikace. Například váhové faktory, které jsou větší pro víceprvkové itemsety, by mohly mít největší hodnotu v případě jednoprvkového itemsetu a dále by se tato hodnota snižovala pro víceprvkové itemsety až k nule. Tento postup by mohl být tím správným řešením faktu, že modifikací Apriori algoritmu, byla odstraněna v podstatě jeho apriori vlastnost, což vedlo k nalezení většího množství víceprvkových itemsetů, které, jak již víme, nemají mít na kvalitu klasifikace větší vliv.

Literatura

[1] Holt, J.D., Chunk, S.M.: *Efficient Mining of Association Rules in Text Databases*, Department of Computer Science and Engineering, Ohio State university, (1999), 234-242

[2] Sedláček, P.: *Text mining a jeho možnosti (aplikace).*, Zdroj: www.fi.muni.cz/usr/jkucera/pv109/2003p/xsedlac5.htm

[3] Bartík, V.: *Získávání asociačních pravidel z relačních dat.*: teze disertační práce, FIT VUT Brno(2003).

[4] Agrawal, R., Srikant, R.: *Fast Algorithms for Mining Association Rules.* IBM Almaden Research Center, (1994)

[5] Jerroudi, Z.: *Apriori-Algorithmus zur Entdeckung von Assoziationsregeln.* PG 402 Wissensmanagment, Lehrstuhl fur Kunstliche Intelligenz, (2001)

[6] Konkoly, K.: *DATAMINING.*, ODBS, (2004)

[7] Hidalgo, J.M.G.: *Text Mining and Internet Content Filtering.* Universidad Europea CEES, ECML/PKDD Tutorial, (2002)

[8] Smékal, L.: *Získávání znalostí z textových dat.*: ročníkový projekt, FIT VUT Brno(2005).

[9] Hynek J., KučeraM.: *Automatická klasifikace dokumentů do tříd metodou Itemsets, její modifikace a vyhodnocení.*: DATAKON, Brno (2001).

[10] Smékal, L.: *Získávání znalostí z textových dat.*: semestrální projekt, FIT VUT Brno(2006).

[RC] The Reuters-21578 Text Categorization Test Collection url: http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

[D1] Kučera, M.: *Modifikace metody Naive-Bayes o prvky itemsets metody.*: ZČU Plzeň (2002).