VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

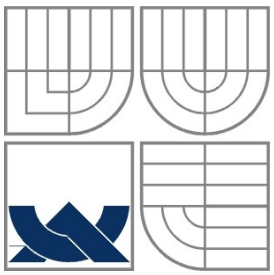
METODY EXTRAKCE INFORMACE Z TEXTOVÝCH DOKUMENTŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

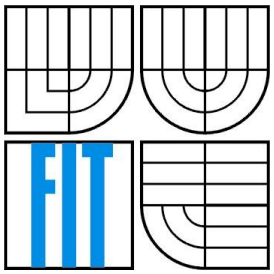
AUTOR PRÁCE
AUTHOR

Bc. Tomáš Sychra

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

METODY EXTRAKCE INFORMACE Z TEXTOVÝCH DOKUMENTŮ

METHODS FOR INFORMATION EXTRACTION IN TEXT DOCUMENTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Tomáš Sychra

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Vladimír Bartík, Ph.D.

BRNO 2008

Abstrakt

Získávání znalostí z textových dokumentů představuje podmnožinu obecného získávání dat – dataminingu. Textové dokumenty však mají vlastnosti odlišné od běžných databází. Tato práce obsahuje přehled metod použitelných pro dolování informací z textů. Nejpoužívanější dolovací úlohou je klasifikace. Popíši možné přístupy při klasifikování dokumentů. V závěru představím algoritmus Winnow, který by měl při klasifikaci dosahovat dobrých výsledků v porovnání s ostatními algoritmy. Součástí práce je i popis implementace algoritmu Winnow a přehled dosažených výsledků.

Klíčová slova

textové dokumenty, extrakce, extrakce informace, klasifikace, kategorizace, lineární klasifikace, Winnow, Balanced Winnow, Positive Winnow

Abstract

Knowledge discovery in text documents is part of data mining. However, text documents have different properties in comparison to regular databases. This project contains an overview of methods for knowledge discovery in text documents. The most frequently used task in this area is document classification. Various approaches for text classification will be described. Finally, I will present algorithm Winnow that should perform better than any other algorithm for classification. There is a description of Winnow implementation and an overview of experimental results.

Keywords

text documents, information extraction, knowledge discovery, classification, categorization, linear classification, Winnow, Balanced Winnow, Positive Winnow

Citace

Sychra Tomáš: Metody extrakce informace z textových dokumentů. Brno, 2008, diplomová práce, FIT VUT v Brně.

Metody extrakce informace z textových dokumentů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Sychra
4. 1. 2008

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Ing. Vladimíru Bartíkovi, Ph.D. za pomoc a radu při vytváření této práce.

Obsah

1 Úvod

Extrakce znalostí z textových dokumentů je činnost podobná získávání znalostí z databází (datamining). Jde o hledání nových, dříve neznámých a pro nás zajímavých znalostí z různých zdrojů, v našem případě z textových dokumentů. V současné době netrpíme nedostatkem zdrojů, ale právě naopak máme k dispozici spoustu dokumentů, které v sobě mohou skrývat zajímavé informace. Metody pro extrakci těchto informací se proto musí vypořádat s velkým množstvím dat a efektivně se zbavovat těch informací, které pro nás nejsou podstatné.

Uplatnění takto získaných informací je možné v řadě oblastí. Je to např. filtrování dokumentů s nevhodným obsahem (filtrování spamu), nalezení podobných dokumentů, vyhledání klíčových slov nebo častých frází.

V této práci stručně popíši různé přístupy k extrakci dat, zaměřím se na metody pro klasifikaci dokumentů a v závěru podrobně představím algoritmus Winnow, který podle různých zdrojů dosahuje při klasifikaci zajímavých výsledků. Algoritmus naimplementuji a experimentálně tak ověřím jeho výkon.

2 Dolování z textu

Zpracování textových dokumentů je odlišné od klasického získávání znalostí z databází. Vyplývá to z rozdílů mezi textovými dokumenty a běžnými databázemi. Textové dokumenty mají žádnou pevně danou strukturu a mohou mít velmi rozdílnou velikost.

Textové dokumenty jsou psány přirozeným jazykem, jehož strojové zpracování je již samo o sobě dosti náročné. V textu se mohou vyskytovat homonyma, tedy slova, která se stejně píší, ale mají v různých souvislostech různé významy (např. oko – na punčoše nebo lidské). Při zpracování je také vhodné počítat se synonymy, tj. různými slovy, která ale mají stejný nebo podobný význam (např. hezký a pěkný). Problém synonym se snaží vyřešit slovník synonym spravovaný univerzitou v Princetonu [1]. Česká slova se v textu často nevyskytují v základním tvaru (např. kůň), ale různě ohnutá (např. koněm) v závislosti na použitém pádu, čísle (u podstatných a přídavných jmen) či osobě a čase (u sloves). Při zpracování slova je proto potřeba nalézt jeho kmen, což ale není triviální záležitost. Angličtina neobsahuje tolik různých tvarů slov jako čeština a zpracování slova je zde jednodušší.

Zatímco v databázích zpravidla nezáleží na pořadí výskytu atributů, u textových dokumentů je tomu naopak. Prohození pořadí slov ve větě může zcela změnit její význam. Význam věty se také může měnit díky interpunkci.

Přirozené jazyky obsahují velké množství slov. Udává se, že čeština obsahuje přibližně 200 000 slov a angličtina dokonce 600 000 slov [2]. V textových dokumentech se na jednu stranu vyskytuje velké množství různých slov, v porovnání s celkovou slovní zásobou jazyka je ale tento počet velmi malý. Pokud bychom dokumenty zobrazovali v prostoru, kde jednotlivé atributy dokumentu (slova, fráze) představují jeho dimenze, zjistíme, že musíme pracovat velkým počtem dimenzí a výskyt dokumentů v tomto prostoru je velmi řídký.

Při extrahování znalostí z textu můžeme využít stejné metody jako při obecném dataminingu, které jsou upraveny pro text. Lepších výkonů ale dosahují algoritmy speciálně navrženy pro dolování z textu.

3 Rozdělení metod

V závislosti na typu hledaných informací lze metody pro extrakci informací rozdělit do několika skupin. Do každé skupiny potom patří algoritmy, které extrahují stejný typ informací, ale dosahují různých výsledků.

První skupinou jsou algoritmy pro nalezení frekventovaných vzorů. Frekventované vzory jsou části dokumentů (slova nebo seskupení slov), které se v dokumentu vyskytují často. Z těchto vzorů lze potom vytvořit tzv. asociační pravidla, která popisují závislosti výskytů jednotlivých vzorů na vzorech ostatních.

Klasifikační algoritmy slouží pro automatické zařazení dokumentů do předem zadaných kategorií. Dokument může náležet do více kategorií nebo také do žádné ze zadaných. Dostatečně robustní algoritmy by si s touto skutečností měly poradit.

Metody pro predikci jsou podobné klasifikačním. Výsledkem predikce však není hodnota z dané množiny, ale obecně spojitá hodnota. Pro získávání znalostí z textu není příliš vhodná.

Poslední zde uvedenou skupinou jsou shlukovací metody. Ty slouží k nalezení podobných dokumentů, které spolu nějak souvisí, a jejich zařazení do tříd. Výsledek je tedy stejný jako u klasifikace. Rozdíl je ale v tom, že u shlukování dopředu neznáme vlastnosti tříd a mnohdy ani jejich počet.

3.1 Asociační pravidla

Již jsme si řekli, že asociační pravidla popisují výskyty frekventovaných vzorů. Definujme nyní tato pravidla přesněji:

Nechť $I = \{i1, i2, i3, \ldots\}$ je množina atributů (v případě textových dokumentů to jsou slova), $D = \{d1, d2, d3, \ldots\}$ je množina všech dokumentů, kde dokument je tvořen slovy (atributy) $d_i \subseteq I$. Asociační pravidlo je implikace ve tvaru $A \Rightarrow B$, kde $A \subset d_i$, $B \subset d_i$ a $A \cap B = \emptyset$. Výskyt atributu (nebo atributů) A implikuje výskyt atributu (nebo atributů) B.

Každé pravidlo ohodnocujeme pomocí podpory (support) a spolehlivosti (confidence). Pravidlo $A \Rightarrow B$ má podporu s, pokud množina dokumentů D obsahuje s% dokumentů obsahujících $A \cup B$. Tedy $s(A \Rightarrow B) = P(A \cup B)$. Pravidlo $A \Rightarrow B$ má spolehlivost c, pokud c% dokumentů množiny D, které obsahují A, obsahují i B. Tedy $c(A \Rightarrow B) = P(B \mid A)$. Tyto metriky nám pomáhají vybrat pro nás vhodná asociační pravidla.

3.1.1 Algoritmus Apriori

Obecný algoritmus pro generování asociačních pravidel vypadá následovně:

- Nalezení frekventovaných vzorů, které mají požadovanou minimální podporu

- Generování asociačních pravidel, zachována jsou jen ta, která splňují minimální spolehlivost

Tento algoritmus je však příliš obecný a pro větší množství dokumentů je v podstatě nepoužitelný díky své exponenciální časové obtížnosti [14].

Algoritmus Apriori slouží k vytváření frekventovaných množin. Začíná vytvořením kandidátních množin o jednom prvku (tzv. 1-množin) a testováním jejich podpory. Množiny s nižší než minimální podporou jsou odstraněny. V každém dalším kroku se vytváří množiny o jeden prvek větší (k+1-množiny) a to pouze za použití množin z předchozího kroku. Algoritmus vychází z předpokladu, že podmnožina, která sama není frekventovaná, nemůže být součástí jiné frekventované množiny. U nově vygenerované množiny spočítáme její podporu a případně zahodíme. Algoritmus končí, jestliže všechny množiny vygenerované v daném kroku mají příliš nízkou hodnotu podpory.

Množinu obsahující všechny frekventované vzory pak získáme sjednocením množin získaných v jednotlivých krocích algoritmu.

Pro zvýšení efektivity algoritmu Apriori existuje několik rozšíření. Např. použitím hašování získáme nižší časovou složitost. Při získávání k-množin jsou totiž zavrženy i k+1-množiny, které již nemohou získat potřebnou minimální podporu.

Pseudokód algoritmus Apriori:

```
1)   L1 = nalezni_frekventované_1-množiny (D);
2)   for (k = 2; L_{k-1} ≠ ∅; k++) {
3)       C_k = generuj_kandidáty(L_{k-1}, min_supp);      // generování nových kandidátů
4)       for all dokument d ∈ D {                         // projdi D pro zjištění počtů výskytů
5)           C_d = subset (C_k, d);                       // kandidáti obsažení v dokumentu d
6)           for all kandidát c ∈ C_d
7)               c.počet_výskytů++;
8)       }
9)       L_k={c ∈ C_k | c.počet_výskytů ≥ min_supp};
10)  }
11)  return L=∪_k L_k;
```

Pseudokód pro funkci generuj_kandidáty(L_{k-1}, min_supp):

```
1)  for all množina l_1 ∈ L_{k-1}
2)      for all množina l_2 ∈ L_{k-1}
3)          if ((l_1[1] = l_2[1]) ∧ (l_1[2] = l_2[2]) ∧ ... ∧ (l_1[k-2] = l_2[k-2]) ∧ (l_1[k-1] = l_2[k-1])) {
4)              c = l_1 JOIN l_2;                        //spojovací krok – generování kandidátů
5)              if (má_nefrekventovanou_podmnožinu(c,L_{k-1})) // vylučovací krok
6)              odstraň c;                               // odstraňování nefrekventovaných kandidátů
```

7) **else** přidej c do C_k;

8) }

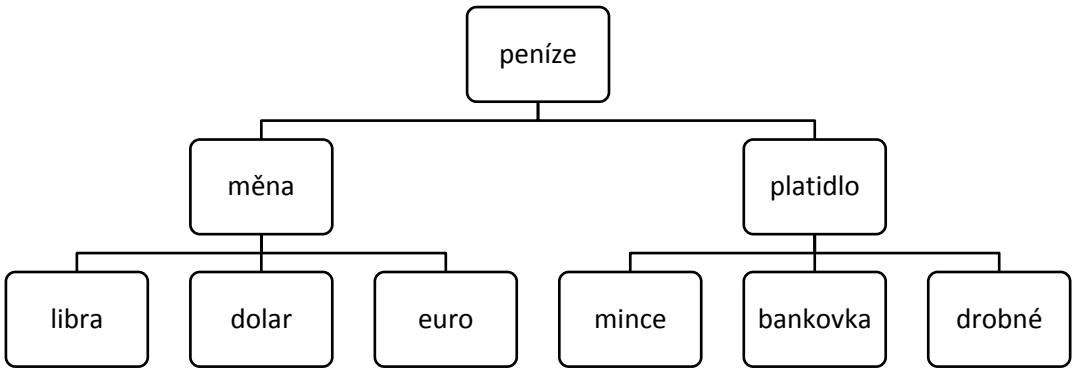
9) **return** C_k;

3.1.2 Současné přístupy

Při získávání informací z velkého množství dokumentů roste počet i nalezených asociačních pravidel. Některé z nich mohou být redundantní nebo nezajímavé, přestože odpovídají požadovaným hodnotám pro podporu a spolehlivost.

3.1.2.1 Víceúrovňová pravidla

Jednou z možností jak odstranit redundantní pravidla a nalézt pravidla další, je použití hierarchického rozdělení atributů do úrovní (obrázek 1). Příkladem může být sada dokumentů, ve které nalezneme různá pravidla obsahující jednotlivé měny států (libra, dolar, …). Ty však nemusí mít dostatečnou podporu a takto bychom o ně přišli. Při použití víceúrovňových pravidel nahradíme jednotlivé měny atributem na vyšší úrovni (měna) a výsledné pravidlo pak získá požadovanou podporu.



Obrázek 1. Příklad konceptuální hierarchie pro dolování asociačních pravidel

Je vhodné použít pro každou vrstvu atributů různé minimální podpory. Čím vyšší vrstva (vyšší abstrakce atributu), tím větší by měla být hodnota minimální podpory.

3.1.2.2 Nové metriky

Omiecinski uvedl v roce 2003 [3] dvě nové metriky. Namísto podpory (support) a spolehlivosti (confidence) používá záruku (bond) a celkovou spolehlivost (all-confidence).

Pokud má množina atributů *Z* celkovou spolehlivost *c*, pak platí, že všechna pravidla, která z ní lze generovat, mají hodnotu spolehlivosti větší nebo rovnu hodnotě *c*.

$$all - confidence(Z) \leq confidence(z \in Z)$$

$$all - confidence(Z) = min\big(confidence(z \in Z)\big)$$

$$all - confidence(Z) = min(confidence(z \in Z))$$

$$all - confidence(Z) = \frac{supp(Z)}{\max(support(z \in Z))} = \frac{P(Z)}{\max(P(z \in Z))}$$

Záruka je počet dokumentů, které obsahují pouze atributy ze Z, dělený počtem dokumentů, které obsahují alespoň jeden atribut ze Z.

$$bond(Z) = \frac{count(d;\ d\ \in D\ \wedge\ d \subseteq Z)}{count(d;\ d \in D\ \wedge\ z \in d\ \wedge\ z \in Z)}$$

Tyto metriky popisují zajímavost nalezených pravidel. Konkrétně, jak závislé na sobě jsou jednotlivé atributy v asociacích.

3.1.2.3 Negativní asociační pravidla

Tento přístup vytváří pravidla ve tvaru A⇒¬B. Při generování je náročné zjistit atribut, jehož nepřítomnost je pro nás zajímavá.

3.2 Klasifikace

Jak jsme si již řekli, klasifikace slouží pro zařazení dokumentu do předem známých kategorií. Příkladem může být např. filtrování nevyžádané pošty, tj. nalezení zpráv (dokumentů), které patří do kategorie spam. Kategorii je možné popsat pomocí požadavku (query) nebo profilu.

Požadavek je vytvořen manuálně uživatelem tak, aby odpovídal dané kategorii. Ten může vypadat např. takto: „do kategorie bankovnictví patří všechny dokumenty, kde se alespoň 10× vyskytne slovo banka“. Aby bylo dosaženo patřičné přesnosti, finální požadavek bývá ve tvaru několika konjunkcí a disjunkcí jednodušších požadavků. Vytvořit takový požadavek je však náročné a velmi náchylné na chybu.

Profil dokumentu je množina (nebo multimnožina) slov, která z dokumentu vznikla jejich extrakcí. Profil kategorie je potom vytvořen pomocí profilů dokumentů, které do této kategorie náleží, tak aby těmto dokumentům odpovídal. Profily kategorií lze tvořit manuálně, ale stejně jako u požadavků (query) je to časově náročné, a proto i drahé.

Při klasifikaci několika tisíců dokumentů je proto žádoucí vytvořit klasifikátor na základě profilů s automatickým vytvářením profilů tříd.

Podle způsobu výběru příslušných kategorií rozlišujeme dva typy klasifikací:

- mono-klasifikace: každý dokument patří právě do jedné z n tříd
- multi-klasifikace: každý dokument patří do žádné nebo několika tříd z n.

Rozdíl mezi těmi to dvěma případy je ve způsobu přiřazování dokumentů do tříd. Při mono-klasifikaci je dokumentu přiřazena ta třída, která dosáhne nejvyššího ohodnocení. Speciálním případem mono-klasifikace je binární klasifikace, kdy existují jen dvě třídy – patří × nepatří.

Při multi-klasifikaci jsou dokumentu přiřazeny všechny třídy, jejichž ohodnocení přesáhne určitou mez. Multi-klasifikaci lze brát jako několik nezávislých binárních mono-klasifikací.

Klasifikační metody lze rozdělit do několika skupin podle jejich přístupu k trénování a hledání odpovídajících kategorií.

Metody s klasifikačními pravidly při trénování hledají pravidla, která popisují danou kategorii. Např. (výskyt_slova(peníze) ∧ počet_slov(úředník) > 5) ⇒ kategorie(banka). Pravidla mohou být použita pro vytvoření rozhodovacího stromu.

Lineární klasifikátory popisují dokumenty i kategorie lineárním vektorem. Při trénování se vektory kategorií nastaví tak, aby odpovídaly trénovacím dokumentům. Při řazení neznámého dokumentu se podle daného algoritmu porovná jeho vektor s vektorem všech kategorií a vybere se kategorie s nejvyšším skóre (případně několik kategorií, jejichž skóre překročí jistou hranici).

Metody založené na příkladech při hledání kategorie nejprve naleznou k neznámému dokumentu dokumenty jemu podobné a podle jejich zařazení zjistí kategorii nového dokumentu.

Do poslední skupiny patří upravené algoritmy pro strojové učení, např. genetické algoritmy, použití fuzzy množin a další.

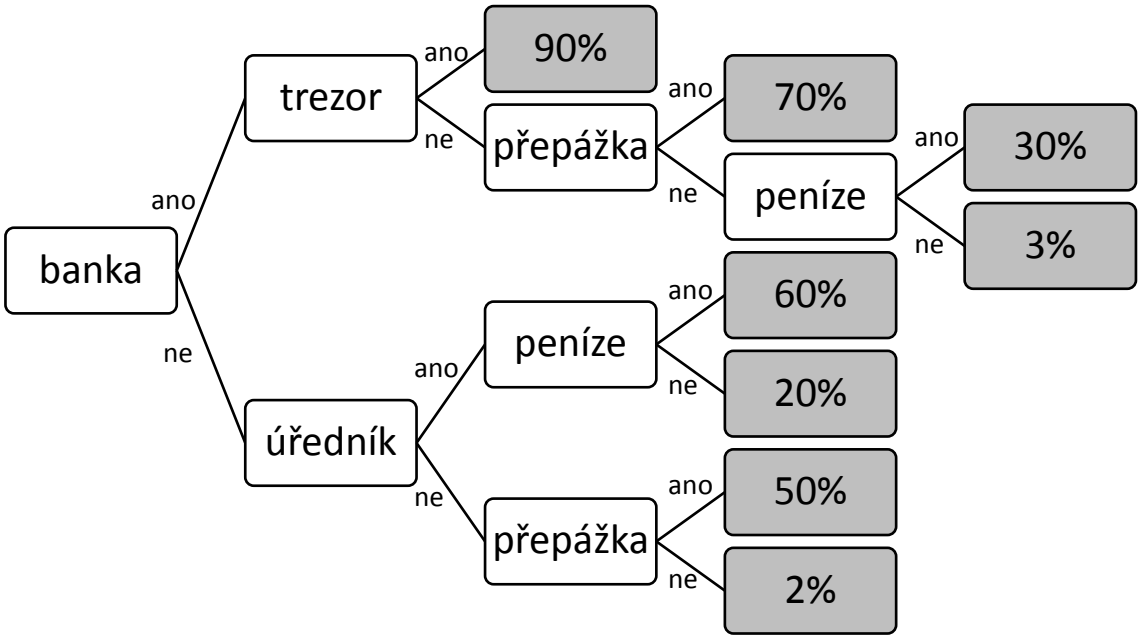
3.2.1 Založená na pravidlech

Metody této kategorie se učí hledáním pravidel z trénovacích dokumentů. Lze využít algoritmu pro hledání asociačních pravidel, nalezená pravidla pak nazýváme klasifikační. Tato pravidla jsou potom použita pro definování jednotlivých kategorií. Pravidla můžeme uspořádat do rozhodovacího stromu.

Rozhodovací stromy jsou grafy se stromovou strukturou. Vnitřní uzly reprezentují klasifikační pravidla a koncové uzly příslušnost (nebo pravděpodobnost příslušnosti) ke třídě. Při klasifikaci do více kategorií je třeba pro každou kategorii vytvořit samostatný strom.

Při tvorbě stromu se nejdříve použijí pravidla s nejvyšší rozhodovací schopností. Jsou to ta, která mají velký vliv na příslušnost ke kategorii. Listový uzel je vytvořen v případě, že pravděpodobnost příslušnosti již přesáhla jistou hranici (prepruning) nebo jsme dosáhli maximální hloubky stromu. Prepruning zabrání generování těch větví, který by nám nepřinesly další užitečnou informaci o příslušnosti je třídě.

Pruning je možné provést i po vytvoření stromu, nazýváme ho postpruning a odstraňuje větvení, která vznikla přispěním okrajových hodnot nebo vlivem přetrénovaní.



Obrázek 2. Příklad rozhodovacího stromu

Ukázka rozhodovacího stromu na obrázku 2 popisuje kategorii dokumentů zabývajících se bankovními loupežemi. Ve skutečných případech však hloubka stromů bývá větší.

3.2.1.1 Možná vylepšení

Klasifikace rozhodovacími stromy nedosahuje příliš vysoké přesnosti a je náchylná na přetrénování. Zvýšení přesnosti lze dosáhnout použitím více stromů pro každou kategorii. Základem je rozdělení trénovacích dat do více skupin [4].

Jednodušší přístupe je tzv. bagging – dokumenty jsou náhodně vybírány do skupin, ale zůstávají i v původní sadě. Může se tak stát, že některé dokumenty budou zvoleny pro trénování vícekrát a některé se do nových skupin vůbec nedostanou.

Druhý přístup se nazývá boosting a dosahuje lepší přesnosti než bagging. Výběr dokumentů a tvorba stromů probíhá iterativně. Pro každý nově tvořený strom jsou vybrány ty dokumenty, které byly v předchozím kroku chybně zařazeny.

3.2.2 Lineární klasifikátory

Dokumenty zpracovávané těmito metodami jsou převedeny na lineární vektory. Jednotlivé hodnoty vektorů odpovídají lexikálním jednotkám dokumentů (např. slovům). Kategorie mají také tvar lineárního vektoru. Metody se pak liší způsobem, jak vytvořit vektor popisující kategorii a jak přiřazovat dokumenty do kategorií.

3.2.2.1 Jednoduchá Bayesovská klasifikace (SBC)

Jednoduchá Bayesovská klasifikace je pravděpodobnostní metoda pro klasifikaci. Vychází z Bayesova teorému o nezávislosti předpokladů. Pro každý dokument se snaží nalézt

pravděpodobnost, s jakou náleží do jednotlivých kategorií. Vybrána je potom kategorie s nejvyšší pravděpodobností nebo ty kategorie, u nichž pravděpodobnost přesáhla jistou prahovou hodnotu.

$$P(c|X) \tag{1}$$

kde *c* je kategorie, *X* je vektor popisující dokument. *P* je pravděpodobnost výběru kategorie *c* za předpokladu, že máme vektor *X*. Podle Bayesova vzorce platí

$$P(c|X) = \frac{P(X|c) \cdot P(c)}{P(X)} \tag{2}$$

kde P(c) je pravděpodobnost výskytu kategorie c, P(X) je pravděpodobnost výskytu dokumentu popsaného vektorem X, P(X|c) je pravděpodobnost, že dokument vybraný z kategorie c bude popsán vektorem X

Vektor X popisující dokument se skládá z hodnot pro jednotlivé atributy dokumentu,

$$X = (a_1, a_2, a_3, \dots, a_n)$$

Proto můžeme zapsat

$$P(X) = P(a_1, a_2, a_3, \dots, a_n)$$

$$P(X|c) = P(a_1, a_2, a_3, \dots, a_n|c)$$

Protože předpokládáme, že jednotlivé atributy jsou na sobě navzájem nezávislé, platí

$$P(a_1, a_2, a_3, \dots, a_n) = P(a_1) \cdot P(a_2) \cdot P(a_3) \cdot \dots \cdot P(a_n) = \prod_{j=1}^{n} P(a_j)$$

$$P(a_1, a_2, a_3, \dots, a_n|c) = P(a_1|c) \cdot P(a_2|c) \cdot P(a_3|c) \cdot \dots \cdot P(a_n|c) = \prod_{j=1}^{n} P(a_j|c)$$

Hodnota atributu a_j je diskrétní. Může nabývat hodnot 0 a 1, pokud určuje jen výskyt atributu nebo jeho hodnota odpovídá počtu výskytu atributu v dokumentu. Pravděpodobnost výskytu atributu a_j v kategorii c je

$$P(a_j|c) = \frac{p_{cj}}{p_c}$$

Kde p_{cj} je počet výskytů atributu a_j v kategorii c a p_c je počet výskytů všech atributů v kategorii c

Výslednou hodnotu podmíněné pravděpodobnosti můžeme zapsat jako

$$P(c|X) = P(c) \cdot \prod_{j=1}^{n} \frac{P(a_j|c)}{P(a_j)} \tag{3}$$

Je důležité, aby pravděpodobnost výskytu každého atributu byla nenulová. V opačném případě by výsledný součin pravděpodobností byl vždy nulový. V testovaných dokumentech se ale mohou objevit atributy nové, s dosud neznámou pravděpodobností. Zavádíme proto implicitní pravděpodobnost (např. 0,001), kterou těmto atributům přiřadíme.

Bayesovská klasifikace může vypadat slibně vzhledem k tomu, že bere v potaz všechny atributy. Na druhou stranu je ale zpracování velkého množství atributů výpočetně náročné. Mohou se tak také projevit okrajové a zašuměné hodnoty atributů. Algoritmus také předpokládá nezávislost jednotlivých atributů (tj. slov). Ve skutečnosti ale mohou jednotlivá slova na sobě záviset.

3.2.2.2 Rocchio

Algoritmus Rocchio je založen na pravděpodobnostních metodách, vyžívá tzv. TF/IDF váhování a zpětnou vazbu pro kontrolu relevance klasifikovaných dokumentů.

TF/IDF váhování udává důležitost atributu s přihlédnutím na výskyt termu v daném dokumentu i v ostatních dokumentech. Skládá se ze dvou částí: TF (term frequency) určuje počet výskytů termu *t* (v tomto případě slova) v dokumentu *d* a IDF (inverse dokument frequency) určuje počet výskytů termu *t* ve všech dokumentech.

$$TF_{t,d} = n_{t,d}$$

$$IDF_t = \frac{1}{n_{t,D}}$$

Výsledná hodnota váhy TD/IDF

$$TF/IDF = TF \cdot IDF = n_{t,d} \cdot \frac{1}{n_{t,D}} \tag{4}$$

Tu lze ještě upravit a normalizovat tak, aby měla lepší vypovídací hodnotu i při klasifikování dokumentů rozdílných velikostí.

V algoritmu Rocchio je pro výpočet hodnot jednotlivých atributů použit tento vzorec

$$s(f,d) = \frac{v(f,d)\log\left(\frac{N}{n_f}\right)}{\sum_{k\in F} v(k,d)\log\left(\frac{N}{n_k}\right)} \tag{5}$$

kde N je počet všech dokumentů, F je množina všech atributů, n_k je počet dokumentů s atributem k, funkce $v(f, d)$ je relativní relevance atributu f v případě více výskytů v dokumentu d

$$v(f,d) = max\left(0, log\left(n_{f,d}\right)\right) \tag{6}$$

kde $n_{f,d}$ je počet výskytu atributu f v dokumentu d

Pro každou kategorii je potom pomocí vektorů trénovacích dokumentů vytvořen vektor, kde hodnota pro každý atribut se vypočte

$$w(f,c) = max\left(0, \frac{\beta}{|D_c|}\sum_{d\in D_c} s(f,d) - \frac{\gamma}{|\overline{D_c}|}\sum_{d\in \overline{D_c}} s(f,d)\right) \tag{7}$$

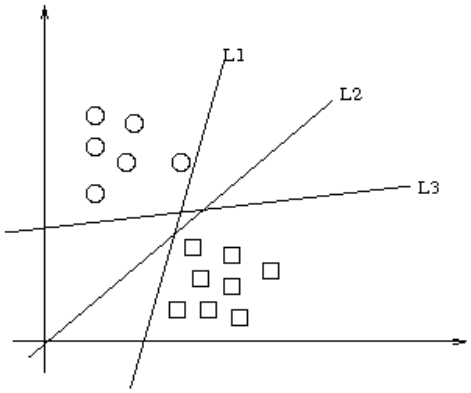
Kde D_c je množina dokumentů zařazených do kategorie c, D_c je množina dokumentů, které nejsou zařazeny do kategorie c, parametry β a γ určují váhu positivních a negativních případů, jejich standardní hodnoty jsou $\beta = 16$ a $\gamma = 4$

Rocchio dosahuje v porovnání s ostatními algoritmy dobrých výsledků při trénování na malé trénovací množině. Na větších množinách jej ostatní algoritmy předčí. Drobným nedostatkem může být použití pouze nezáporných vah, nemohou se tak dostatečně projevit atributy, které jednoznačně značí nenáležitost ke kategorii.

3.2.2.3 Support Vector Machine

SVM je univerzální klasifikátor, který lze použit i jako klasifikaci při rozpoznávání obrazu. V základní verzi patří SVM mezi lineární klasifikátory, lze ho ale modifikovat i jako polynomiální klasifikátor. Jeho výhodou je, že přesnost není závislá na velikosti prostoru atributů.

Při trénování tento algoritmus hledá nadrovinu, která rozděluje data do dvou skupin. Jako měřítko optimálnosti slouží velikost okraje, s jakým dokáže separovat dokumenty patřící do dané kategorie od ostatních. Dalším měřítkem je dimenzionalita nadroviny, nižší počet dimenzí značí lepší výběr atributů pro klasifikátor.



Obrázek 3. SVM klasifikace (převzato z [3])

V příkladu na obrázku 4. jsou umístěny data popsaná 2D vektorem, rozdělující nadrovina je v tomto případě přímka. Nejoptimálnější je rozdělení pomocí klasifikátoru L2, který obě skupiny dělí tak, že mezi skupinami vznikne největší okraj.

3.2.2.4 Sleeping Expert

Algoritmus Sleeping Experts (spící odborníci) pro klasifikaci kombinuje výsledky několika jednodušších klasifikátorů (odborníků). Odhadu každého „odborníka“ je přiřazena váha, která udává vliv na celkové zařazení. V průběhu trénování jsou váhy jednotlivých klasifikátorů upravovány podle jejich úspěšnosti na právě trénovaném příkladu. Některé klasifikátory nemusí být aktuálně aktivní, potom se nazývají spící (sleeping).

V případě klasifikace dokumentů odpovídají jedné lexikální jednotce (v nejjednodušším případě slovu) *w* dva experti $\overline{w}_1$ a $\overline{w}_0$. První je nastaven tak, aby předpovídal 1 (tj. výskyt slova znamená příslušnost ke třídě), zatímco druhý předpovídá 0 (výskyt slova znamená, že dokument do kategorie nepatří). Expert je aktivní, pokud se jeho slovo vyskytuje v dokumentu, v opačném případě je neaktivní – spící. Množina aktivních expertů pro daný dokument *d* je potom

$$E_d = \{\overline{w}_k | w \in W_d, k \in \{0,1\}\}$$

kde W^d je množina slov obsažených v dokumentu *d*

Algoritmus jako celek obsahuje tzv. pool (bazén), což je množina p uchovávající pro každé slovo jeho váhu.

Celkové rozhodnutí algoritmu je dáno váženým součet rozhodnutí jednotlivých expertů.

$$y_t = \sum_{\bar{w}_k \in E_t} p_{\bar{w}_k} \cdot k \tag{8}$$

Pokud výsledná hodnota překročí jistou prahovou hodnotu (typicky 0.5) je dokument zařazen do kategorie, v opačném případě do kategorie nepatří.

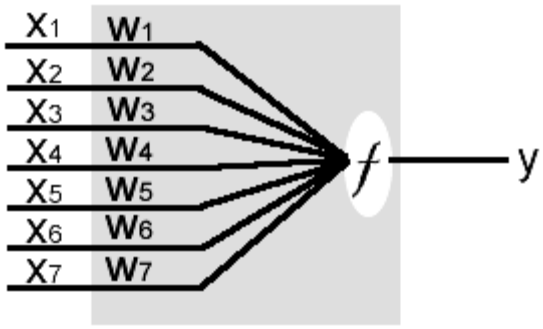
3.2.2.5 Perceptron

Perceptron je základní stavební prvek pří tvorbě neuronových sítí. Pokud je použit samostatně, stává se z něj lineární klasifikátor. Kategorie i dokumenty jsou vyjádřeny lineárním vektorem

$$document = \{s_1, s_2, s_3, \dots, s_n\}$$

$$category = \{w_1, w_2, w_3, \dots, w_m\}$$

Kde s_i jsou váhy jednotlivých atributů v dokumentu a w_i jsou váhy atributů, rozhodující o jejich vlivu na zařazení do dané kategorie.



Obrázek 4. Perceptron

Výsledné rozhodnutí se určí podle součtu vážených hodnot aktivních atributů. Perceptron je binární klasifikátor, dokument je zařazen do kategorie, pokud hodnota součtu y přesáhne jistou prahovou hodnotu.

$$y = \sum_{i=0}^{n} w_i \cdot s_i \tag{9}$$

Při trénování perceptron upravuje váhy aktivních atributů pouze, pokud došlo k chybě. Jedná se tedy o algoritmus typu mistake-driven. Váhy se mění aditivním způsobem, tj. dojde k přičtení, resp. odečtení malé konstanty.

3.2.2.6 Winnow

U algoritmu Winnow zde popíši jeho vlastnosti jen stručně, podrobně se mu budu věnovat v pozdější kapitole.

Winnow je klasifikátor podobný perceptronu, uvádí se [5], že oba patří do stejné rodiny. I zde jsou tedy kategorie i dokumenty popsány lineárními vektory. Při trénování jsou váhy kategorií upraveny, jen pokud dojde k chybě.

$$document = \{s_1, s_2, s_3, \dots, s_n\}$$

$$category = \{w_1, w_2, w_3, \dots, w_m\}$$

Rozdíl algoritmu Winnow v základní verzi oproti algoritmu Perceptron je pouze ve způsobu změny vah. U Winnow jsou váhy upravovány multiplikativně.

První verze Winnow se jmenuje Positive Winnow a obsahuje jen jeden vektor vah pro každou kategorii, pozdější vylepšení – Balanced Winnow – pracuje se dvěma vektory a výsledná váha se vypočte z jejich rozdílu. Toto vylepšení znamená, že váha u některých atributů může být záporná.

3.2.3 Example based

Metody patřící do této skupiny využívají jisté podobnosti mezi dokumenty. Při klasifikování neznámého dokumentu hledají mezi klasifikovanými dokumenty jemu podobné. Podle jejich příslušnosti ke třídám pak zjistí hledanou třídu pro nový dokument.

Mezi klasifikátory této skupiny patří např. algoritmus k-NN (k-nearest neighbor – k-nejbližší sousedství [6].

3.2.4 Ostatní

Genetické algoritmy jsou inspirovány přírodou a snaží se tak kopírovat přirozený vývoj. Gen každého tvora je odvozen od genů jeho rodičů. Na přežití mají větší šanci silnější tvorové, tj. ti s lepším genem. Při klasifikaci přiřadíme každé kategorii náhodný gen (zakódovaný pomocí jedniček a nul) a při trénovaní jednotlivé bity měníme tak, aby byl gen co nejlepší a postihl ty dokumenty, které do kategorie náleží.

Algoritmy založené na fuzzy množinách při rozhodování využívají rozhodovací pravidla. Výsledek pravidla není ostře 1 - pravda nebo 0 - nepravda, ale hodnota z intervalu <0,1>, která lépe vyjadřuje náležitost k určité kategorii.

3.3 Predikce

Předmětem predikce je odhadnutí neznámé hodnoty (ze spojité funkce). Je to rozdíl oproti klasifikaci, kde máme na výběr z množiny možných kategorií.

Uplatnění predikce je především při dolování dat z transakčních databází, kde se jednotlivé položky skládají jen z několika málo atributů. Nejznámější metodou tzv. lineární jednoduchá a lineární násobná regrese [7].

Při regresi prokládáme data přímkou tak, aby jim co nejvíce odpovídala. V případě jednoduché regrese je hledaná přímka ve tvaru

$$Y = aX + b$$

kde X jsou vstupní parametry dat a Y jsou výstupní parametry. Úkolem regrese je nalézt parametry *a* a *b*.

U lineární vícenásobné regrese lze použít více vstupních parametrů. Hledaná rovnice je tedy ve tvaru

$$Y = a_0 + a_1X_1 + a_2X_2 + \ldots + a_vX_v \tag{10}$$

kde vstupní parametry X, výstupní parametry Y a hledané parametry A lze zapsat jako matice

$$X = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1v} \\ 1 & x_{21} & \ldots & x_{2v} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{s1} & \ldots & x_{sv} \end{pmatrix}$$

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{pmatrix}$$

$$A = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_v \end{pmatrix}$$

3.4 Shlukování

Shlukování (clustering) je rozdělování množiny objektů do podmnožin tak, že v každé podmnožině jsou objekty s podobnými vlastnostmi a rozdíly mezi jednotlivými podmnožina jsou co největší. Před rozdělováním neznáme vlastnosti podmnožin a mnohdy ani jejich počet. Pro trénování není nutná trénovací množina (množina objektů, u kterých známe jejich zařazení), jedná se o učení bez učitele.

V případě shlukování dokumentů jsou objekty jednotlivé dokumenty. Dokument je popsán lineárním vektorem vytvořeným na základě jejich atributů (tj. slov). Pro výpočet podobnosti se používá tzv. vzdálenostní funkce.

3.4.1 K-means

Základním algoritmem využívaným pro shlukování je *k*-means. Dělí množinu objektů do k podmnožin (shluků) tak, aby vzdálenosti uvnitř shluků byly co nejmenší. Snažíme se minimalizovat funkci V

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} \left(x_j \cdot \mu_i\right)^2 \tag{11}$$

kde S_i je *i*-tý shluk, x_j je objekt (dokument) ve shluku, μ_i je střed shluku *i*

Algoritmus probíhá v iteracích. Na začátku je množina objektů rozdělena na k shluků, a to buď zcela náhodně, nebo za použití nějaké heuristiky.

- pro každý shluk se zjistí jeho střed (nebo těžiště)
- objektu jsou přeskupeny do shluků podle jejich vzdáleností od nových středů

Tyto dva body se opakují tak dlouho, dokud objekty mění příslušnost ke shlukům nebo dochází ke změně poloh středů (těžišť).

K-means negarantuje nalezení optimálního rozdělení, výsledek je závislý na určení počátečních shluků. Pro vylepšení výsledku je možné spustit algoritmus několikrát s různým počátečním nastavením a vybrat výsledek s nejlepšími parametry. Další nevýhodou je nutnost zadání počtu shluků.

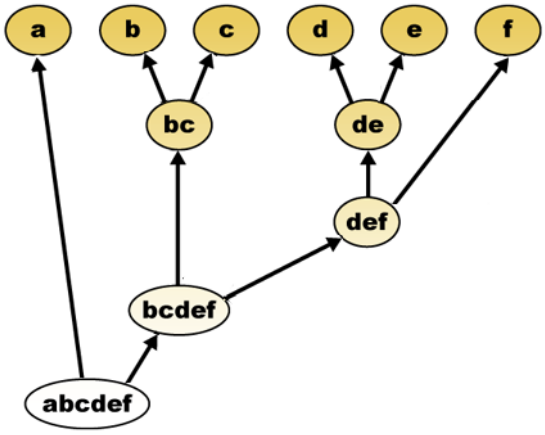
3.4.2 BiSec-k-means

Metoda BiSec-k-means je variantou k-means, ale liší se v inicializaci. Zkratka BiSec vychází ze slova bisekční a znamená, že zde bude docházet k dělení shluků.

Algoritmus začíná s jedním shlukem, který obsahuje všechny dokumenty. Algoritmus dále pokračuje:

1. Vyber shluk pro rozdělení
2. Pomocí k-means rozděl shluk na dvě části (opakuj vícekrát a vyber nejlepší rozdělení, viz problém s nalezením optimální řešení výše)
3. Opakuj body 1 a 2, dokud nemáme dostatečný počet shluků nebo suma vzdáleností neklesla pod určitou hranici

Výběr shluku pro rozdělení může být na základě jeho velikosti (počet obsažených dokumentů), míry podobnosti uvnitř shluku (suma vzdáleností od středu) nebo obojího.



Obrázek 5. Hierarchické shlukování

BiSec-k-means ve své podstatě vytváří hierarchické shlukován. Pokud mají být výsledkem shluky na stejné úrovni, následuje po bodu 2 vylepšení pomocí standardního *k*-means algoritmu.

3.4.3 SCAD

Simultaneous Clustering and Attribute Discrimination (SCAD) byl navržen tak, aby zároveň hledal optimální středy shluků a váhy pro jednotlivé atributy. Každý shluk má vlastní vektor vah a míru příslušnosti. Funkce, kterou se SCAD snaží minimalizovat je

$$V = \sum_{i=1}^{C} \sum_{x_j \in X_i} \sum_{k=1}^{n} v_{ik} \cdot \left(x_{jk} - c_{ik}\right)^2 + \sum_{i=1}^{C} \delta_i \sum_{k=1}^{n} {v_{ik}}^2 \tag{12}$$

kde C je množina všech středů shluků, $c_i = (c_{i1}, c_{i1}, \dots, c_{ik}, \dots, c_{in})$ je střed shluku i, X_i je množina všech dokumentů přiřazených do shluku i, $x_i = (x_{i1}, x_{i1}, \dots, x_{ik}, \dots, x_{in})$ je dokument ve shluku i a v_{ik} je váha atributu k ve shluku i, která nabývá hodnot 0 a 1.

δ_i určuje rozložení vah pro atributy. Je-li hodnota δ_i příliš malá, bude jeden atribut v rámci shluku i dominantní. Naopak, bude-li hodnota δ_i velká, budou všechny atributy shluku i relevantní a budou mít stejnou váhu.

V případě, kdy jsou shlukovány objekty se všemi atributy stejně relevantními, dosahuje SCAD podobných výsledků jako k-means. U shlukování textových dokumentů, kde má každý atribut jinou váhu, dosahuje SCAD výrazného zlepšení [8].

4 Metodologie klasifikace

Nyní, když jsme prošli všechny druhy metod pro získávání informací z textových dokumentů, můžeme se více soustředit na klasifikaci, která má největší využití při zpracování textů a dolování informací z nich. Popíšeme si, jak klasifikace probíhá, jak hodnotíme její výsledky a s jakými problémy se při klasifikování můžeme setkat.

4.1 Trénování a testování

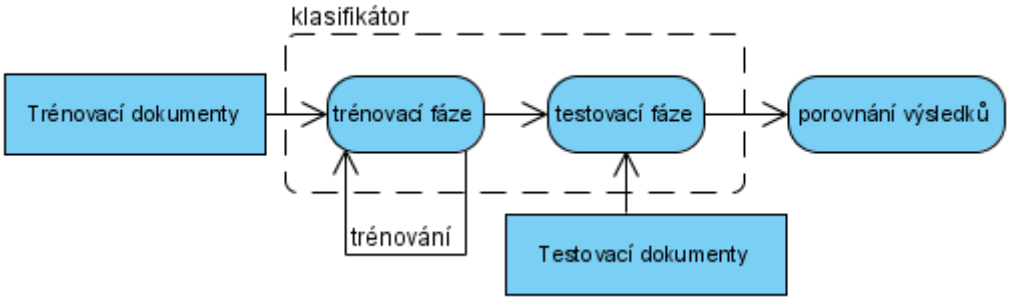
Každý klasifikační algoritmus se skládá ze dvou fází, z trénovací a testovací fáze.

4.1.1 Trénovací fáze

Při trénování dochází k natrénování klasifikátoru na trénovacích datech. Jsou to dokumenty, u kterých známe jejich zařazení do kategorií. Jedná se o trénování s učitelem, algoritmy mají v dispozici správné zařazení a podle něj mohou rozhodnout o ukončení procesu trénování.

4.1.2 Testovací fáze

V testovací fázi ověřujeme výkon algoritmu na testovacích datech. Opět to jsou dokumenty, u kterých známe jejich zařazení do kategorií. Platí, že průnik množiny trénovacích a testovacích dat je prázdný. Algoritmus tedy testujeme na neznámých datech. Dokumentům jsou pak podle natrénovaných znalostí přiřazeny odpovídající kategorie a výsledek porovná se správným zařazením.



Obrázek 6. Činnost klasifikátoru

4.2 Hodnocení

Pro zjištění výkonu algoritmů slouží několik metrik, pomocí kterých dokážeme přesně popsat a porovnat míru jejich úspěšnosti. Při hodnocení předpokládáme binární klasifikaci, tj. určení, zda

dokument patří či nepatří do kategorie. Pro výpočet všech metrik je třeba získat tyto počty dokumentů:

- p^+ = počet relevantních dokumentů, správně zařazených do kategorie (true positive)
- p^- = počet relevantních dokumentů, chybně nezařazených do kategorie (false negative)
- n^- = počet nerelevantních dokumentů, správně nezařazených do kategorie (true negative)
- n^+ = počet nerelevantních dokumentů, chybně zařazených do kategorie (false positive)

Základními metrikami jsou přesnost a úplnost.

4.2.1 Přesnost – precision

Přesnost při klasifikaci textových dokumentů je definována jako „počet správně zařazených dokumentů dělený počtem všech zařazených dokumentů“, tedy:

$$precision = \frac{p^+}{p^+ + n^+} \tag{13}$$

Další interpretace přesnosti je pravděpodobnost, s jakou mezi zařazenými dokumenty vybereme dokument správně zařazený.

4.2.2 Úplnost – recall

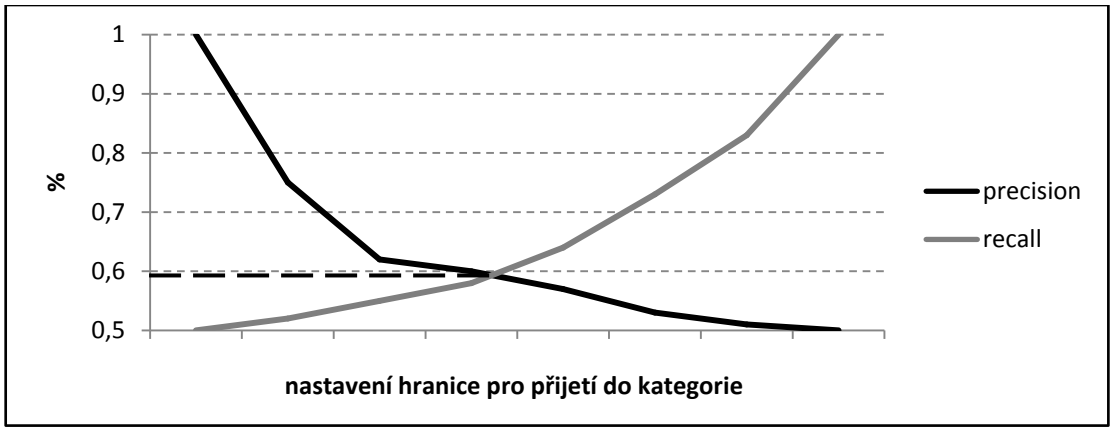
Úplnost při klasifikaci textových dokumentů je definována jako „počet správně zařazených dokumentů dělený počtem všech relevantních dokumentů“, tedy:

$$recall = \frac{p^+}{p^+ + p^-} \tag{14}$$

Další interpretace přesnosti je pravděpodobnost, s jakou je libovolný relevantní dokument správně zařazen do kategorie.

4.2.3 Break-even point

Break-even point je obecně bod, ve kterém mají dvě funkce stejnou hodnotu. Při hodnocení klasifikace jde o hodnoty přesnosti a úplnosti. Platí, že s rostoucí přesností klesá úplnost a naopak. Při maximální přesnosti totiž raději neklasifikujeme dokumenty, u kterých si nejsme příliš jistí. Při maximální úplnosti raději klasifikujeme co nejvíce dokumentů, abychom postihli všechny relevantní.



Obrázek 7. Příklad možného vývoje přesnosti / úplnosti

Graf znázorňuje vývoje přesnosti a úplnosti v závislosti na nastavení algoritmu, průsečík obou funkcí určuje hodnotu break-even pointu.

Break-even point bývá často používán pro hodnocení různých klasifikačních algoritmů, jeho hodnota v sobě totiž zahrnuje hodnotu přesnosti i úplnosti.

4.2.4 Fall-out

Fall-out (odpad) je při klasifikaci textových dokumentů definován jako „počet zařazených nerelevantních dokumentů dělený počtem všech nerelevantních dokumentů“, tedy:

$$fallout = \frac{n^+}{n^+ + n^-} \tag{15}$$

4.2.5 F-measure

F-measure je definován jako vážený harmonický průměr přesnosti a úplnosti, tradiční (vyvážená) rovnice má tvar:

$$F - measure = \frac{2 \cdot (precision \cdot recall)}{(precision + recall)} \tag{16}$$

kde přesnost má stejnou váhu jako úplnost a nazývá se též F_1*-measure*. Obecná rovnice F_α*-measure* umožňuje přikládat přesnosti i úplnosti různou váhu. Používané jsou F_2 resp. $F_{0,5}$, které zdvojnásobují váhu přesnosti resp. úplnosti. Tvar rovnice F_α*-measure*

$$F_\alpha - measure = \frac{(1 + \alpha) \cdot (precision \cdot recall)}{(\alpha \cdot precision + recall)} \tag{17}$$

4.2.6 Error-rate

Error-rate (chybovost) je při klasifikaci textových dokumentů definována jako „počet provedených chyb při klasifikaci dokumentů dělený počtem všech dokumentů“, tedy:

$$error\ rate = \frac{n^+ + p^-}{N} \tag{18}$$

kde $N = (p^+ + p^- + n^+ + n^-)$ je počet všech dokumentů

4.2.7 Průměrné hodnoty

V naprosté většině případů klasifikujeme dokumenty do více kategorií. Řazení do každé kategorie může mít jinou přesnost i úplnosti. Existují potom dva způsoby jak průměrovat dosažené výsledky.

4.2.7.1 Mikro průměrování

Jednotlivé počty dokumentů p^+, p^-, n^+, n^- se sčítají přes všechny kategorie. Hodnota takovéhoto průměru je nejvíce ovlivněna velkými kategoriemi (obsahující velké počet dokumentů).

4.2.7.2 Makro průměrování

Pro každou kategorii se vypočte její ohodnocení (přesnost, úplnost, …), to je sečteno a vyděleno počtem kategorií. Tato hodnota je potom ovlivněna malými kategoriemi (obsahující malý počet dokumentů), které mají stejnou váhu jako ty velké.

4.3 Problémy klasifikace

V procesu klasifikace může vyvstat řada problémů, které mohou velmi ovlivnit výkon klasifikace. Je to např. i doba trénování, která může být neúnosná pro jistý druh aplikace. Další problémy jsou přetrénování a přítomnost velkého počtu nevýznamových slov (např. já, se, a, …)

4.3.1 Přetrénování

K přetrénování dochází, pokud metoda dokáže přesně popsat trénovací data, ale selhává na datech testovacích. To může být způsobeno šumem v trénovacích datech, na který se klasifikátor přesně natrénuje. Některé metody jsou na přetrénování náchylné (např. neuronové sítě), u jiných k němu vůbec nedochází (např. Winnow).

4.3.2 Stoplist

Přítomnost velkého počtu nevýznamových slov snižuje výpočetní výkon klasifikátoru a také jeho přesnost. Pro odstranění těchto slov slouží stoplist, tj. seznam těchto slov. Při předzpracování dokumentů jsou jednotlivá slova porovnávána oproti tomuto seznamu a případně vypouštěna.

5 Winnow

Algoritmus Winnow byl poprvé uveden Nickem Littlestonem v roce 1988 v jeho knize Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. Ve skutečnosti je Winnow celou rodinou algoritmů, do které patří např. i perceptron (uvedený v roce 1958 Rosenblattem). Winnow v angličtině znamená prosívat obilí od plev a algoritmus při trénování efektivně separuje relevantní atributy (tj. slova) od nerelevantních. Winnow je popsán jako binární klasifikátor, rozšíření na multi-klasifikaci jsme si ukázali při dělení klasifikačních metod.

Winnow je lineární klasifikátor a patří mezi on-line a mistake-driven algoritmy. Každý dokument je tedy popsán vektorem $d = \{s_1, s_2, s_3, \dots, s_n\}$, kde s_i je síla i-tého atributu, trénování probíhá v jednotlivých iteracích a vektor vah atributů je upraven, jen pokud při trénování nastane chyba.

Klasifikování dokumentu se provádí jen na základě jeho aktivních atributů, ty neaktivní mají hodnotu nula. Algoritmus používá tři parametry: práh θ (obvykle 1), zvyšující parametr α ($\alpha > 1$) a snižující parametr β ($0 < \beta < 1$).

U Winnow je zaručeno, že nalezne ideální oddělovač, pokud existuje, dosahuje však i dobré úspěšnosti pokud oddělovač neexistuje. Na rozdíl od řasy jiných algoritmů (např. Bayesovská predikce) nepracuje s žádnými předpoklady o atributech (vzájemná nezávislost apod.).

5.1 Varianty

5.1.1 Positive Winnow

Algoritmus uchovává n-dimenzionální vektor s váhami atributů $w = \{w_1, w_2, w_3, \dots, w_n\}$, kde w_i je váha i-tého atributu. Tento vektor je upraven při chybě u trénování. Na počátku jsou váhy nastaveny na stejnou malou hodnotu, např. θ/d, kde d je průměrný počet aktivních atributů v dokumentu. Výsledné skóre by se mělo na počátku pohybovat v okolí prahové hodnoty θ.

Dokument je zařazen do kategorie, pokud skóre přesáhne práh, tedy

$$\sum_{i=0}^{n} w_i \cdot s_i > \theta \tag{19}$$

kde w_i je váha atributu a s_i je síla atributu ve zkoumaném dokumentu.

Váhy ve vektoru w jsou upraveny (pokud dojde k chybě) následovně: pokud algoritmus předpoví 0 (dokument nezařazen) a má být 1, potom váhy aktivních atributů jsou α-krát navýšeny. Pokud algoritmus předpoví 0 (dokument zařazen) a má být 0, jsou váhy všech aktivních atributů β-krát sníženy.

předpověď / správná hodn.	**0**	**1**
0	-	$w_i = w_i \cdot \beta$
1	$w_i = w_i \cdot \propto$	-

Obrázek 8. Změna vah u Positive Winnow

Váhy neaktivních atributů zůstávají v obou případech stejné.

5.1.2 Balanced Winnow

V případě Balanced Winnow jsou pro každý atribut uchovávány dvě váhy, w_i^+ a w_i^-. Výslednou váhu atributu zjistíme jejich rozdílem, může tedy nabývat i záporných hodnot. Počáteční nastavení vah je u vektoru w^+ $2\theta/d$ a u vektoru w^- θ/d, výsledné skóre se tedy na počátku opět pohybuje v okolí prahu θ.

Dokument je zařazen do kategorie, pokud skóre přesáhne práh, tedy

$$\sum_{i=0}^{n} (w_i^+ - w_i^-) \cdot s_i > \theta \tag{20}$$

kde w_i^+ je kladná váha atributu, w_i^- záporná váha atributu a s_i je síla atributu ve zkoumaném dokumentu.

Váhy vektorů w^+ a w^- jsou upraveny opět jen při chybě: pokud algoritmus předpoví 0 (dokument nezařazen) a má být 1, potom jsou kladné váhy w_i^+ aktivních atributů α-krát navýšeny. Pokud algoritmus předpoví 0 (dokument zařazen) a má být 0, jsou záporné váhy w_i^- všech aktivních atributů β-krát sníženy.

předpověď / správná hodn.	**0**	**1**
0	-	$w_i^- = w_i^- \cdot \beta$
1	$w_i^+ = w_i^+ \cdot \propto$	-

Obrázek 9. Změna vah u Balanced Winnow

Váhy neaktivních atributů zůstávají v obou případech stejné.

5.2 Rozšíření

Výkon základních variant algoritmu lze vylepšit použitím několika vylepšení, která se vypořádají např. s nestejnou velikostí dokumentů

5.2.1 Normalizace síly atributů

Velké dokumenty mají při klasifikaci nespornou výhodu oproti menším, vyskytuje se v nich větší počet atributů, jejich skóre tak snáze přesáhne prahovou hodnotu a dokumenty zařazeny do kategorie. Balanced Winnow toto řeší použitím záporných vah. Použití záporných vah slouží také pro odhalení atributů, které svou přítomností značí nenáležitost dokumentu do kategorie. K tomuto však při klasifikaci textových dokumentů příliš nedochází, záporné váhy tak slouží především pro tolerování různých délek dokumentů.

Při použití Positive Winnow (který používá pouze kladné váhy) však musíme síly atributů normalizovat vzhledem k délce dokumentu:

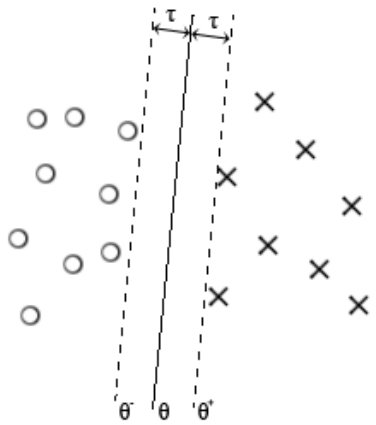
$$s^n(f,d) = \frac{s(f,d)}{\sum_{f \in d} s(f,d)} \tag{21}$$

kde d je dokument, f je atribut, $s(f, d)$ je původní síla atributu f v dokumentu d a $s^n(f, d)$ je normalizovaná síla atributu f v dokumentu d. V tomto případě musíme upravit počáteční nastavení váhového vektoru w, všechno hodnoty budou začínat s hodnotou θ (namísto θ/d).

Použití normalizace ale stále nedosáhne výkonu jako při použití záporných vah. Je to způsobeno rovnoměrnou normalizací, u velkých dokumentů neroste počet důležitých atributů lineárně s počtem všech atributů, ale jejich síla je lineárně normalizována.

5.2.2 Použití rozsahu prahů

Při trénování lineárního klasifikátoru hledáme váhový vektor. Je to vektor, který v prostoru odděluje relevantní a nerelevantní dokumenty. Pokud jsou dokumenty dokonale oddělitelné, je možné, že existuje více takovýchto vektorů, které je oddělují. Nejlepším vektorem bude takový, který obě skupiny oddělí tou nejširší hranicí. Tento přístup je možné použít, i pokud dokonalý oddělovač neexistuje.



Obrázek 10. Použití rozsahu prahů u Positive Winnow

K váhovému vektoru budeme hledat hyperplochu s největší šířkou τ, tak aby klasifikátor správně zařadil všechny relevantní dokumenty $F(d) > \left(\theta + \frac{\tau}{2}\right)$ a odmítl nerelevantní dokumenty $F(d) < \left(\theta - \frac{\tau}{2}\right)$.

Je možné hledat optimální šířku τ [9], nebo přímo určit horní a dolní hranici prahu (θ^+ a θ^-), tak že θ^+ a $\theta^- = \tau$. Při trénování potom algoritmus předpoví 0, pokud je skóre dokumentu menší než θ^-. Podobně algoritmus předpoví 1, pokud je skóre dokumentu větší než θ^+. Všechny dokumenty se skóre v intervalu $\langle \theta^-, \theta^+ \rangle$ jsou označeny jako chyby a musí tedy dojít k úpravě vah. Příklad nastavení hodnot: $\theta^- = 0{,}9$ a $\theta^+ = 1{,}1$.

5.2.3 Opakování atributů

V dokumentech se každý atribut může vyskytovat vícekrát než jen jednou. Je třeba zvážit, jak velkou sílu atributu přiřadit v závislosti na počtu jeho výskytů. Ve většině případů je více výskytů atributu znakem větší míry příslušnosti do kategorie. Na druhou stranu, při velkém počtu opakování atributu by jeho síla zastínila ostatní atributy.

Winnow je on-line algoritmus (zpracovává dokumenty každý zvlášť), nemůžeme tedy projít všechny dokumenty a zjistit inverzní frekvenci atributu ve všech dokumentech (idf, viz kapitola 3.2.2.2).

Pro určení síly atributy tedy existují tři možnosti:

- $s(f, d) = 1$, síla atributu je 1, pokud se atribut v dokumentu vyskytuje, jinak je 0
- $s(f, d) = n(f, d)$, síla atributu odpovídá počtu jeho výskytů v dokumentu
- $s(f, d) = \sqrt{n(f, d)}$, síla atributu odpovídá odmocnině počtu jeho výskytů v dokumentu

Podle Dagana [10] se nejlepších výsledků dosahuje při použití třetí možnosti.

5.2.4 Odstranění atributů

Winnow je schopen vypořádat se s velkým množstvím atributů, zařazení do kategorie je však často závislé jen na malém počtu atributů. Pro zvýšení efektivnosti je vhodné nerelevantní atributy odstranit. Mělo by tak dojít i ke zvýšení přesnosti, neboť nerelevantní atributy přinášejí jen šum do skóre klasifikátoru.

K odstraňování dochází v průběhu trénování. Po několika kolech trénování, až je dosažena jistá úspěšnost, projdeme hodnoty atributů ve váhovém vektoru. Ty hodnoty, které se příliš nezměnily od počátečního nastavení, můžeme odstranit. Případ kdy se hodnota nemění, znamená, že se příslušný atribut nepodílí na žádné chybě.

6 Implementace

Zvolený algoritmus Winnow jsem se rozhodl naimplementovat v jazyce Java. Pro trénování a testování budou použity Reuters články, které budou uložené ve formátu XML. Různá nastavení klasifikátoru se budou načítat ze souboru formátu INI.

6.1 Popis technologií

6.1.1 Java

Java je objektově orientovaný programovací jazyk vyvinutý společností Sun Microsystems uvedený v roce 1995. V roce 2007 Sun uvolnil zdrojové kódy Javy, ta bude nyní vyvíjena jako open-source. Syntaxe Javy je podobná jazyku C, neobsahuje ale některé konstrukce, jako např. příkaz goto nebo použití bezznaménkových čísel.

Java je interpretovaný jazyk, při překladu se nevytváří přímo strojový kód, ale tzv. bajtkód (mezikód). Tento kód je možné spustit na jakémkoli zařízení, které obsahuje interpret Javy, tzv. Java Virtual Machine (JVM). Správa paměti je jednodušší než v jazyce C/C++, v Javě je realizována pomocí Garbage collectoru, který automaticky uvolňuje již nepoužívanou paměti.

Nevýhodou Javy je pomalý start programu, protože JVM musí bajtkód nejdříve přeložit a až poté spustit. I jednoduchý program zabírá po spuštění hodně paměti, protože se musí spustit JVM.

6.1.2 XML

Značkovací jazyk XML (eXtensible Markup Language, rozšiřitelný značkovací jazyk) byl přijat konsorciem W3C v roce 1998. Byl vytvořen jako jednodušší podmnožina jazyka SGML.

Slouží pro ukládání informací tak, aby byly strojově snadno čitelné a zároveň srozumitelné pro člověka. Styl dokumentu lze popsat např. pomocí jazyka DTD, který umožňuje definovat možné značky a jejich obsah.

Jeho výhodou je především otevřenost, striktní syntaxe, textová podoba a velká spousta volně dostupných XML parserů. Nevýhodou pak zbytečná velikost a robustnost při ukládání informací.

6.1.3 INI

Formát INI je de facto standard pro konfigurační soubory, není však schválen žádnou standardizační organizací. Výhodou je jednoduchá syntaxe, nevýhodou omezené možnosti pro ukládání informací. Není samozřejmě tak robustní jako např. XML.

Jeho syntaxe se skládá z definice parametrů, sekcí a komentářů. Každý parametr je uveden ve tvaru

$$jméno = hodnota$$

V sekcích se shlukují parametry, které logicky patří k sobě. Neexistuje explicitní konec sekce, ta končí u deklarace následující sekce nebo na konci souboru. Deklarace má tvar

$$[jméno\ sekce]$$

Komentář je uveden znakem středníku. Veškerý text od středníku do konce řádku je brán jako komentář bez jakéhokoli významu pro aplikaci.

$$;\ komentář\ do\ konce\ řádku$$

6.2 Návrh aplikace

Program napsaný v jazyce Java bude implementovat klasifikační algoritmus Winnow. Klasifikaci bude provádět na dokumentech Reuters-21578, které budou uloženy ve formátu XML, různá nastavení programu se mohou měnit pomocí konfiguračního souboru INI.

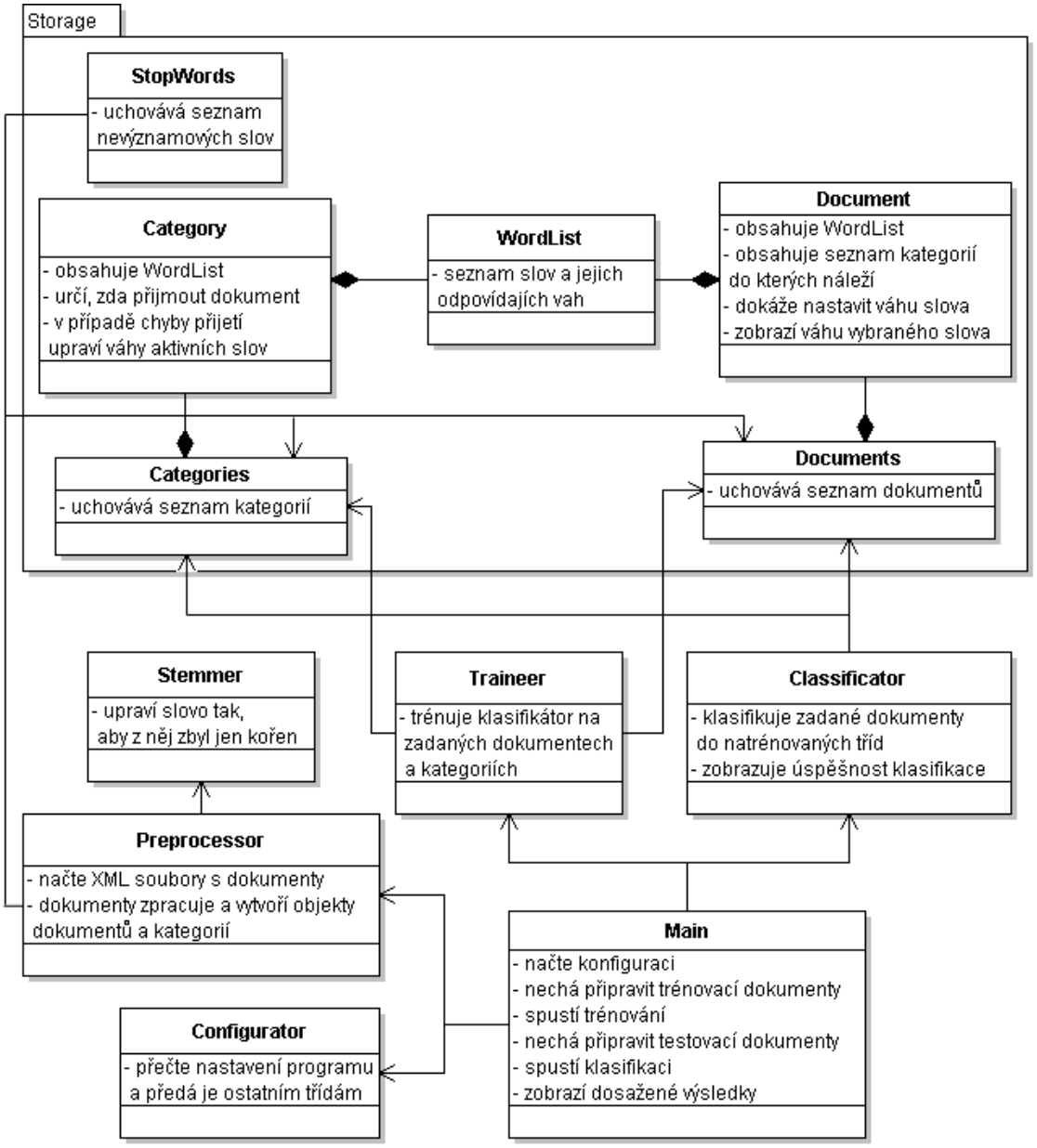
Činnost programu se skládá ze tří hlavních částí: předzpracování, trénování a validace. Před jejich startem je třeba načít konfigurační informace ze souboru. O to se postará třída `Configurator`, která předá načtené proměnné ostatním třídám. V konfiguračním souboru budou uloženy informace o úložišti trénovacích a testovacích dokumentů, použitém stoplistu, typu algoritmu (Positive Winnow × Balanced Winnow) a jeho nastavení (α, β, θ, požadovaná přesnost, …).

První fáze (předzpracování) je provedena v třídě `Preprocessor`, která načte XML soubory trénovacích dokumentů a dokumenty převede na lineární vektory (objekty třídy `Document`). Slova dokumentů jsou porovnána oproti stoplistu a případně vynechána. Pro odsekání koncovek slov je tu třída `Stemmer`, (třída vytvořená Martinem Porterem a upravené Martinem Uhlířem). Pomocí načtených dokumentů se vytvoří množina kategorií, do kterých se dokumenty budou klasifikovat. Kategorie jsou uloženy v objektech `Category`.

Při trénování (třída `Trainer`) budeme cyklicky procházet všechny dokumenty (budou v objektu `Documents`) a zařazovat je do kategorií (uložených v `Categories`). Pokud nastane chyba klasifikace, příslušná kategorie si upraví svoje váhové vektory. Trénování končí, pokud dosáhneme jistého počtu iterací (nastaveno v konfiguračním souboru) nebo přesnost natrénování překročí jistou hranici (opět nastaveno v konfiguračním souboru).

Před validací proběhne načtení testovacích dokumentů (objektem třídy `Preprocessor`). Poté jsou dokumenty zařazovány do kategorií stejně jako v předchozí fázi, pouze nedochází k úpravě vektorů kategorií. To zařizuje třída `Classificator`, která navíc uchovává počty p^+, p^-, n^- a n^+ pro hodnocení klasifikace.

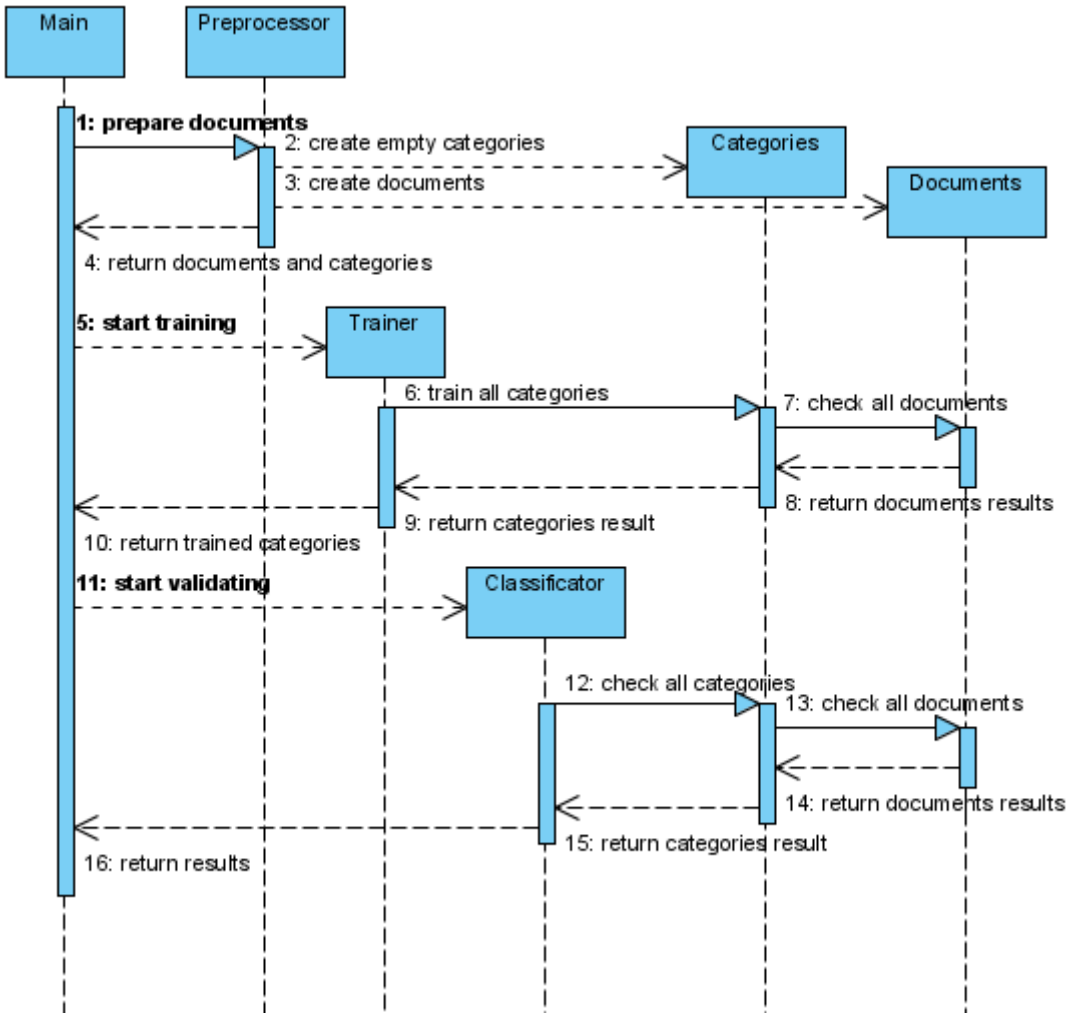
V balíku `Storage` jsou soustředěné třídy sloužící pro uchování dat, jsou to třídy: `Document`, `Documents`, `Category`, `Categories`, `CategoriesName`, `WordList` a `StopWords`.



Obrázek 11. Diagram tříd

`WordList` slouží pro uchování vektorů dokumentů a kategorií. Pokud je použita verze Balanced Winnow, obsahuje každá kategorie dva objekty třídy `WordList`.

`StopWords` uchovává seznam slov, která nemají pro klasifikaci význam, a umožňuje v něm jejich rychlé vyhledání.



Obrázek 12. Sekvenční diagram

6.2.1 Dokumenty REUTERS-21578

Dokumenty v sadě REUTERS-21578 jsou novinové články agentury Reuters z roku 1987. Ve stejném roce byly skupinou expertů zařazeny do kategorií. V roce 1990 byla celá sada uvolněna pro vědecké účely univerzitě v Massachusetts. Zde byly dokumenty převedeny Davidem D. Lewisem a Stephenem Hardingem do formátu SGML.

Dokumenty v této sadě jsou několika způsoby rozděleny na dvě skupiny – na trénovací a testovací dokumenty. Prvním rozdělením je Lewisovo (Lewis split), které dělí dokumenty na 13 626 trénovacích a 6 188 testovacích dokumentů. Aptovo rozdělení (Apte split) vychází z Lewisova, ale obsahuje jen dokumenty zařazené do kategorií (topics), z toho plyne 9 603 trénovacích a 3 299 testovacích dokumentů. CGI rozdělení (CGI split) obsahuje i nezařazené dokumenty. Je ale nejstarší a pro porovnání klasifikačních metod se již nepoužívá. Důležitá jsou tedy rozdělení Lewis split a Apte split.

Příklad formátování dokumentu ze sady REUTERS-21578:

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET"
         OLDID="16327" NEWID="1007">
  <DATE> 3-MAR-1987 09:30:07.60</DATE>
  <TOPICS><D>earn</D></TOPICS>
  <PLACES><D>canada</D></PLACES>
  <PEOPLE/>
  <ORGS/>
  <EXCHANGES/>
  <COMPANIES/>
  <UNKNOWN>
     E F
     f0314 reute
     r f BC-precambrian-shield   03-03 0054</UNKNOWN>
  <TEXT>
    <TITLE>PRECAMBRIAN SHIELD RESOURCES LTD YEAR LOSS</TITLE>
    <DATELINE>    CALGARY, Alberta, March 3 -  </DATELINE>
    <BODY>Shr loss 1.93 dlrs vs profit 16 cts
         Net loss 53,412,000 vs profit 4,479,000
         Revs 24.8 mln vs 32.7 mln
         Note: 1986 shr and net include 51,187,000 dlr writedown on
         U.S. operations, uneconomic coal operations and other mineral
         properties
          Reuter
    </BODY>
  </TEXT>
</REUTERS>
```

Pro svou klasifikaci používám informace o kategorii mezi značkami `<topics>` a informace o těle dokumentu, které jsou mezi značkami `<title>` a `<body>`.

6.2.2 Spouštění aplikace

Při spuštění aplikace převezme z příkazové řádky jméno konfiguračního souboru, ve kterém jsou nastaveny parametry pro běh aplikace.

6.2.2.1 Konfigurační soubor

Konfigurační soubor je rozdělen do čtyř sekcí. V první a druhé jsou uvedeny názvy souborů, ze kterých budou získány trénovací (první sekce) a testovací dokumenty (druhá sekce). Třetí sekce obsahuje název stoplist souboru, tj. souboru s nevýznamovými slovy. V poslední sekci mohou být nastaveny různé parametry pro běh aplikace. Je možné nastavit tyto parametry (uvedeny vždy s příkladem nastavení):

- `alfa=1.1`

  Parametr pro kladnou změnu vah (v případě odmítnutí dokumentu, který patří do kategorie)
- `beta=0.9`

  Parametr pro zápornou změnu vah (v případě přijetí dokumentu, který nepatří do kategorie)
- `theta=1`

  Parametr pro určení zda přijmout dokument do kategorie podle vzorce (19) nebo (20)
- `maxRounds=50`

  Parametr nastavuje maximální počet kol pro natrénování klasifikátoru
- `minPrecision=90`

  Parametr určuje minimální požadovanou přesnost klasifikátoru na trénovacích datech. Trénování je ukončeno, pokud je překročena hodnota `maxRounds` nebo `minPrecision`
- `minDocumentLength=20`

  Parametr určuje minimální délku, které musí dokument dosáhnout, aby se podílel na trénování, resp. testování.
- `useBalanced=true`

  Parametr pro nastavení verze algoritmus Winnow, implicitní je `true`, tedy Balanced, `false` znamená použití verze Positive Winnow.
- `minDocuments=20`

  Parametr určuje minimální počet dokumentů, které musí kategorie při trénování obsahovat, aby byla započtena do všech výsledků. Výsledné hodnoty jsou spočteny dvakrát, jednou pro všechny kategorie a podruhé jen pro kategorie, které odpovídají této podmínce.
- `weightType=binary`

  Parametr určuje, jak budou spočteny váhy pro jednotlivá slova, která se vyskytují v dokumentech. Varianta `binary` znamená použití jen vah 0 a 1, u varianty `count` váha odpovídá počtu výskytů slova v dokumentu a u varianty `sqrt` je váha spočtena jako odmocnina počtu výskytů. Více o nastavení vah v kapitole Opakování atributů.
- `splitUsed=lewis`

  Parametr umožňuje použít přednastavenou sadu dokumentů pro trénování a testování. Je možné použít Lewisovo rozdělení (`lewis`) nebo Aptovo rozdělení (`apte`). Dokumenty jsou stále vybírány ze souborů uvedených v sekcích jedna a dvě, pokud chceme použít celou sadu, je nutné uvést všechny soubory. Také je potřeba nastavit minimální délku dokumentů (`minDocumentLength`) na hodnotu nula.

- `verbose=true`

  Parametr rozlišuje dva způsoby výstupu aplikace – výstup všech informací a výstup pouze výsledných hodnot.

6.2.2.2 Příkazová řádka

Některé parametry mohou být nastaveny i pomocí příkazové řádky, toto nastavení má větší prioritu než nastavení v konfiguračním souboru.

Možné způsoby spuštění tedy budou:

```
java winnow.Main
```

- Pokud není nastaven konfigurační soubor, aplikace se pokusí otevřít defaultní konfigurační soubor. Pokud se toto nezdaří, činnost aplikace končí.

```
java winnow.Main config.ini
```

- Je otevřen konfigurační soubor config.ini, ze kterého jsou načtena všechna nastavení. Pokud některý parametr chybí, je použita defaultní hodnota.

```
java winnow.Main config.ini ([parametr=hodnota]*)
```

- Je otevřen konfigurační soubor config.ini, ze kterého jsou načtena nastavení o trénovacích a testovacích dokumentech. Ostatní nastavení mohou být získána z příkazové řádky. Parametr může nabývat stejných hodnot jako parametry v konfiguračním souboru v sekci [variables].

Aplikaci lze spustit i z archivu JAR

```
java –jar Winnow.jar config.ini
```

Při práci s velkým množstvím dat je třeba spustit program s požadavkem na přidělení více operační paměti, tj.

```
java -jar Winnow.jar –Xms512M –Xmx512M config.ini
java winnow.Main –Xms512M –Xmx512M config.ini
```

6.2.3 Výstup aplikace

Aplikaci bude možno spustit ve dvou módech. První mód bude vypisovat veškerou aktuální činnost, např. průběh zpracování XML souborů, stav natrénování apod. Toto bude sloužit především pro kontrolu správné činnosti.

Příklad výstupu aplikace:

```
---------- TRAINING ----------
Parsing file: 'reut2-000.xml'... done
Parsing file: 'reut2-001.xml'... done
Parsing file: 'reut2-002.xml'... done
Parsing file: 'reut2-003.xml'... done
```

```
Parsing file: 'reut2-004.xml'... done
Parsing file: 'reut2-005.xml'... done
Parsing file: 'reut2-006.xml'... done
Parsing file: 'reut2-007.xml'... done
Parsing file: 'reut2-008.xml'... done
Parsing file: 'reut2-009.xml'... done
Parsing file: 'reut2-010.xml'... done
Parsing file: 'reut2-011.xml'... done
Parsing file: 'reut2-012.xml'... done
Parsing file: 'reut2-013.xml'... done
Read 6802 documents
Read 114 categories
Average length of document is 92.57806527491914
--------------------------------------------------
Training: round 1/50... done (55.31437888515235%)
Training: round 2/50...
```

Druhá možnost spuštění bude uplatněna při sběru výsledků. Výstupem bude jen řádek hodnot vzájemně oddělených tabelátorem. První polovina hodnot budou počáteční nastavení klasifikátoru (např. počet trénovacích a testovacích dokumentů, použitá verze algoritmu) a druhá budou dosažené výsledky (dosažená přesnost a úplnost).

Příklad výstupu aplikace:

`[počet kol] [natrénovaná úspěšnost] [počet trénovacích dokumentů] [počet testovacích dokumentů] [počet kategorií] [použitý split] [způsob přiřazení vah] [alfa] [beta] [theta] [minimální úspěšnost] [použitý stoplist] [balanced] [minimální délka dokumentu] [minimální počet dokumentů v kategorii] [mikro přesnost] [mikro úplnost] [makro přesnost] [makro úplnost] [mikro přesnost velkých kategorií] [mikro úplnost velkých kategorií] [makro přesnost velkých kategorií] [makro úplnost velkých kategorií]`

6.2.4 Popis důležitých tříd

V této kapitole podrobněji představím implementaci tříd, které mají podstatný vliv na správnou činnost aplikace.

6.2.4.1 Třída Main

Hlavní třída, která zpracuje vstupy z příkazové řádky. Spouští zpracování dokumentů, trénování a testování. Na závěr zobrazuje výsledky.

6.2.4.2 Třída Preprocessor

Zpracovává XML dokument obsahující Reuters články. Vytváří seznam dokumentů a aktivních kategorií.

Pro zpracování je volána metoda `processXML()`, která prochází postupně všechny předložené XML soubory a získává z nich jednotlivé dokumenty. Implicitně jsou zpracovány jen dokumenty mající kategorii a více než 20 slov. Lze také zvolit takové nastavení, aby zpracované dokumenty odpovídaly Lewis splitu, příp. Apte splitu.

Tělo dokumentu je rozděleno na jednotlivá slova. Každé slovo projde úpravou, kde je převedeno na malá písmena a jsou z něj odstraněny všechny znaky, které nejsou písmeny (např. čárky tečky, pomlčky, …). Následuje odstranění případné koncovky třídou `Stemmer`. Na závěr je slovo porovnáno proti seznamu nevýznamových slov a v případě shody vynecháno.

Pokud dokument obsahuje kategorii, je tato kategorie vložena do seznamu kategorií.

Třída poskytuje metody `getDocuments()`, resp. `getCategories()` pro získání načtených dokumentů, resp. kategorií.

Privátní metody `processWord(String s)`, `removeNonLiterals(string s)` a `replaceHTMLtags(String s)`, které slouží pro zpracování slov, byly převzaty od Ing. Martina Uhlíře.

6.2.4.3 Třída Category

Obsahuje natrénovaný vektor vah, rozhodne, zda přijmout dokument a upravit svoje váhy.

Metoda `accept(Document doc)` určuje, jestli bude dokument `doc` přijat do kategorie na základě jeho vektoru slov. Metoda v cyklu prochází svůj vektor vah a od dokumentu získá jeho odpovídající váhu. Výsledek je zjištěn na základě použité verze Winnow (Positive nebo Balanced) podle vzorce 19, resp. 20.

Metoda `checkDocument(Document doc)` slouží pro natrénování kategorie. Od dokumentu `doc` zjistí, zda má patřit do této kategorie a metodou `accept`, zda je dokument přijat. Pokud nastane chyba, jsou upraveny váhy privátní metodou `updateDocument()`. Metoda vrací výsledek zařazení dokumentu (true positive, true negative, false positive, false negative).

Metoda `updateDocument(Document doc, Type result)` upravuje váhy slov, která jsou aktivní v dokumentu `doc`. Změna vah je provedena v závislosti na typu výsledku `result` a použité verzi Winnow podle vzorců – viz Obrázek 8 a Obrázek 9.

6.2.4.4 Třída Document

Obsahuje obsah dokumentu ve formě vektoru slov, klasifikované dokumenty obsahují navíc pole kategorií `CategoriesName`, do kterých náleží.

Metoda `updateWeights()` upravuje váhy opakujících se slov podle zvoleného nastavení, které je získáno z třídy `Configurator`. Možné jsou tři způsoby úpravy: `binary` – přiřazuje slovu váhu jedna, pokud se alespoň jednou vyskytne v dokumentu; `count` – váha slova odpovídá jeho počtu výskytů v dokumentu a `sqrt` – váha slova je vypočtena jako odmocnina počtu výskytů slova v dokumentu.

6.2.4.5 Třída Trainer

Trénuje kategorie, aby odpovídaly trénovacím dokumentům.

Metoda `train()` prochází v kolech všechny kategorie a všechny dokumenty. Pro každou kombinaci kategorie a dokumentu je zjištěn výsledek (metodou `checkDocument()` třídy `Category`), podle kterého se zjistí přesnost aktuálního kola (podle vzorce (13)). Trénování je ukončeno, pokud je dosaženo minimální požadované přesnosti nebo maximálního počtu kol.

6.2.4.6 Třída Classificator

Klasifikuje testovací dokumenty do kategorií a počítá výkon klasifikátoru.

Metoda `classify(Document doc)` se snaží najít kategorie, ke kterým zadaný dokument náleží. V cyklu se prochází všechny natrénované kategorie a u každé z nich se metodou `accept()` testuje přijeti dokumentu. Metoda vrací seznam kategorií formou objektu `CategoriesName`.

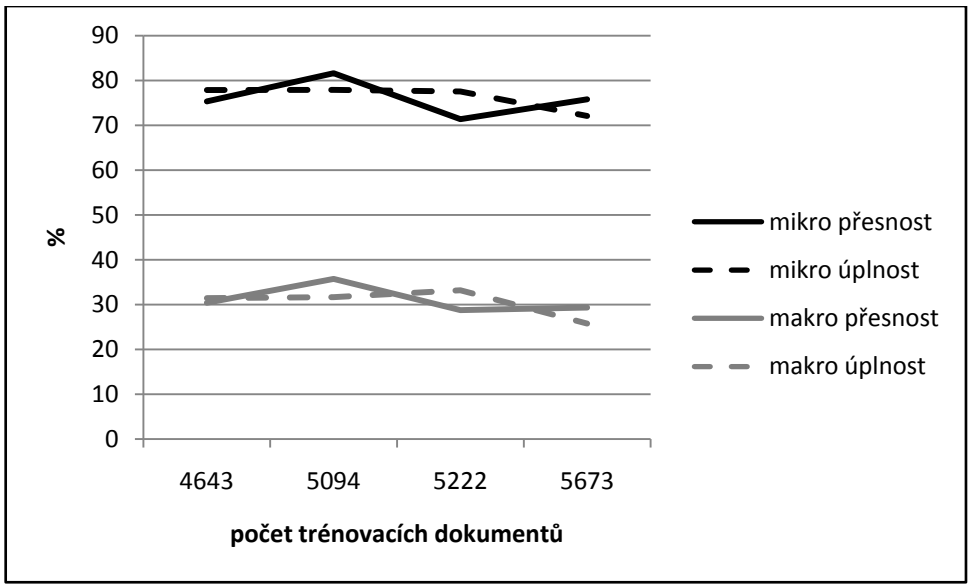
Metoda `getPerformance(Documents docs)` vrací výsledky testování dosažené na testovací sadě v objektu `docs`. Výsledkem testování je zjištění hodnot přesnosti a úplnosti, získaných mikro i makro průměrováním (více v kapitole 4.2.7). Je možné z výsledků vypustit ty kategorie, které při trénování neměly dostatečný počet odpovídajících dokumentů. Proto jsou spočteny výsledky i bez těchto kategorií. Výsledky jsou uloženy v poli (`HashMap`), kde klíč odpovídá názvu metriky (např. `macroPrecision`) a hodnota její hodnotě.

7 Výsledky

Při návrhu aplikace jsem postupoval tak, aby bylo možno jednoduše měnit nastavení klasifikátoru a snadno tak získat jednoduše porovnatelné výsledky. Pro porovnání výsledků s ostatními algoritmy provedu testy na rozdělené sadě Lewis a Apte. Dále budu experimentovat s nastavením, abych dosáhl co možná nejlepších výsledků. Kromě jiného se pokusím naleznout nejlepší seznam nevýznamových slov, nejvhodnější typ přiřazení vah slovům a vyzkouším, zda algoritmus nepodlehne přetrénování.

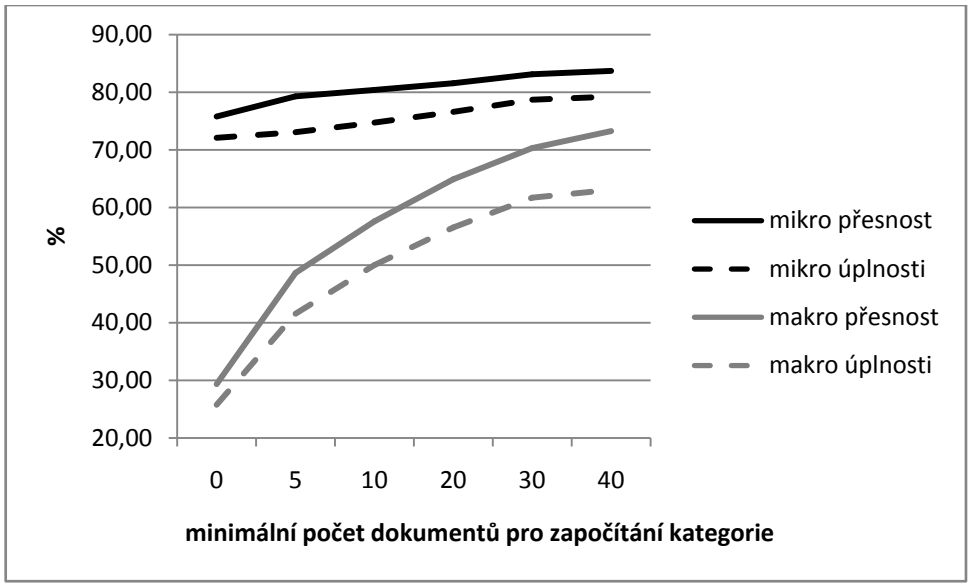
7.1 Mikro × makro přesnost

Protože klasifikace probíhá do více kategorií, je možné získat průměrné hodnoty přesnosti a úplnosti makro nebo mikro průměrováním. Porovnání obou možností je zobrazeno v následujícím grafu na obrázku Obrázek 13. Klasifikátor byl nastaven na verzi Balanced s minimální trénovací přesností 80% a minimální délkou dokumentu 20 slov.



Obrázek 13. Graf – mikro a makro hodnoty přesnosti a úplnosti

Zřetelný rozdíl v hodnotách je způsoben přítomností kategorií, které při trénování obsahují malý počet dokumentů a není tak možné je dobře natrénovat. Vypouštěním těchto malých kategorií tak makro přesnost i úplnost stoupá.



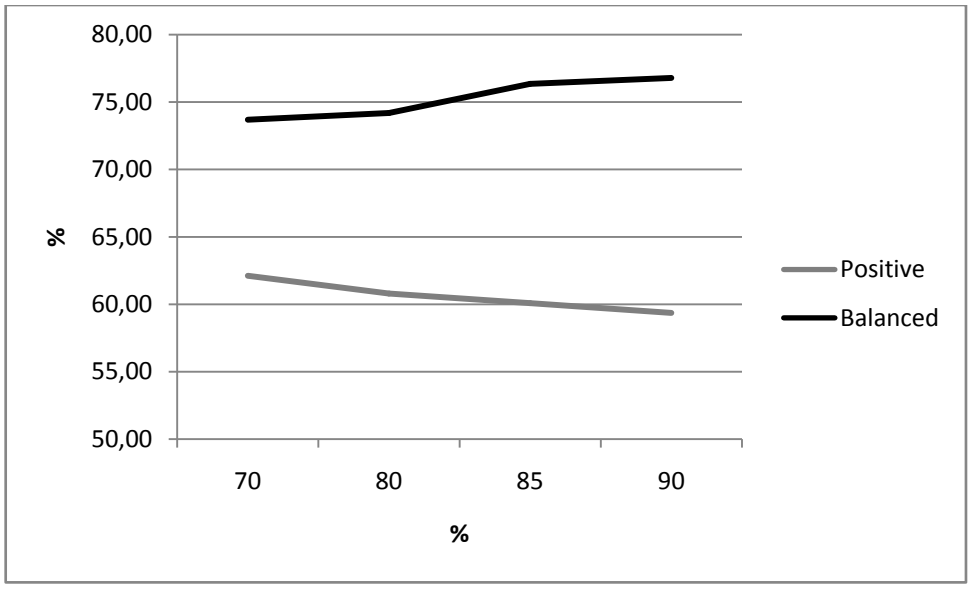
Obrázek 14. Graf – vývoj přesnosti při vynechání malých kategorií

Z grafu je vidět výrazné zlepšení hodnot získaných makro průměrováním. V sadě dokumentů Reuters-21578 je uvedeno celkem 135 různých kategorií, pouze 57 z nich se vyskytuje u alespoň 20-ti dokumentů. Mikro hodnoty přesnosti a úplnosti se při vynechání malých kategorií příliš nemění.

V dalších výsledcích budu uvádět pouze hodnoty získané mikro průměrováním. Pokud nebude uvedeno jinak, hodnoty zobrazené v následujících grafech budou odpovídat hodnotě break-even pointu pro přesnost a úplnost.

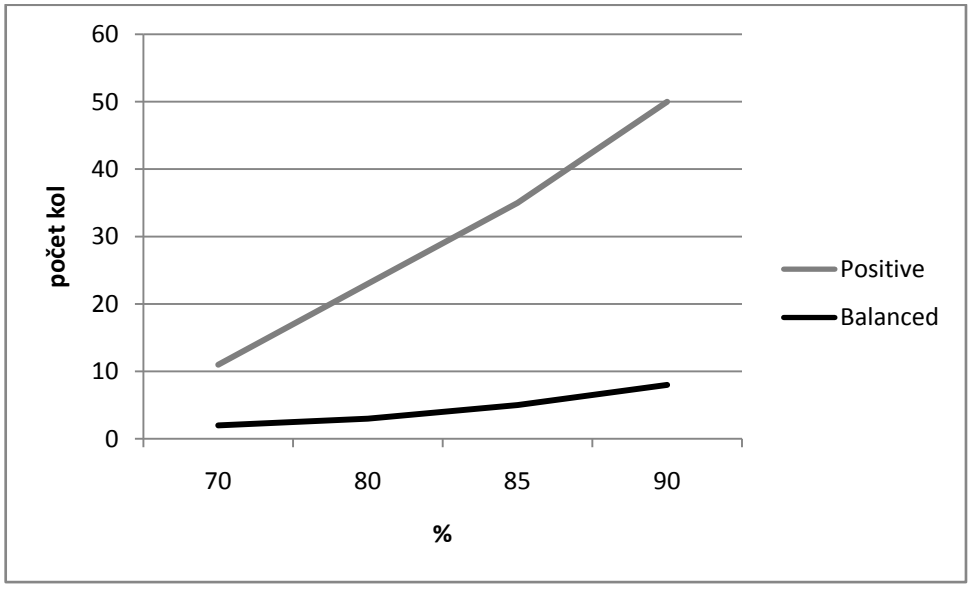
7.2 Positive × Balanced Winnow

Obě verze algoritmu Winnow jsem testoval na stejných trénovacích i testovacích dokumentech bez použití stoplistu Poměr trénovacích dat k testovaným byl 15:7. Podle předpokladů by měla verze Balanced výrazně předčít verzi Positive, která je velmi citlivá na různé velikosti klasifikovaných dokumentů.



Obrázek 15. Graf – break-even point u verzí Winnow podle přesnosti trénování

Rozdíl mezi těmito verzemi není znatelný jen ve výsledné hodnotě přesnosti a úplnosti, ale i u času potřebnému pro natrénování. Zatímco verzi Balanced stačí pro natrénování na 80% přesnosti 3 kola, u verze Positive je potřeba až 35 kol. Další výsledky jsou v grafu na následujícím obrázku.



Obrázek 16. Graf – počet kol potřebných pro natrénování Positive a Balanced Winnow

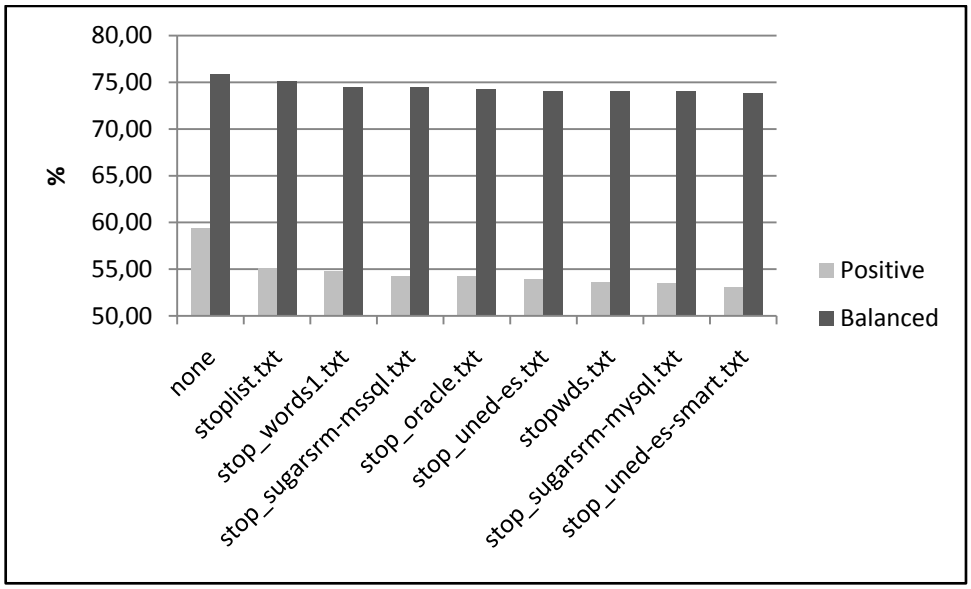
Verze Balanced Winnow se osvědčila a pokud nebude uvedeno jinak, bude implicitně používána při získávání dalších výsledků.

7.3 Nejlepší stoplist

V příloze Příloha 1 je uveden seznam použitých stylistů pro testování. Jde o seznamy slov použité v různých databázích nebo v projektech zabývajících se zpracováním textu. Nejkratší z nich má 39 slov, nejdelší jich obsahuje 568.

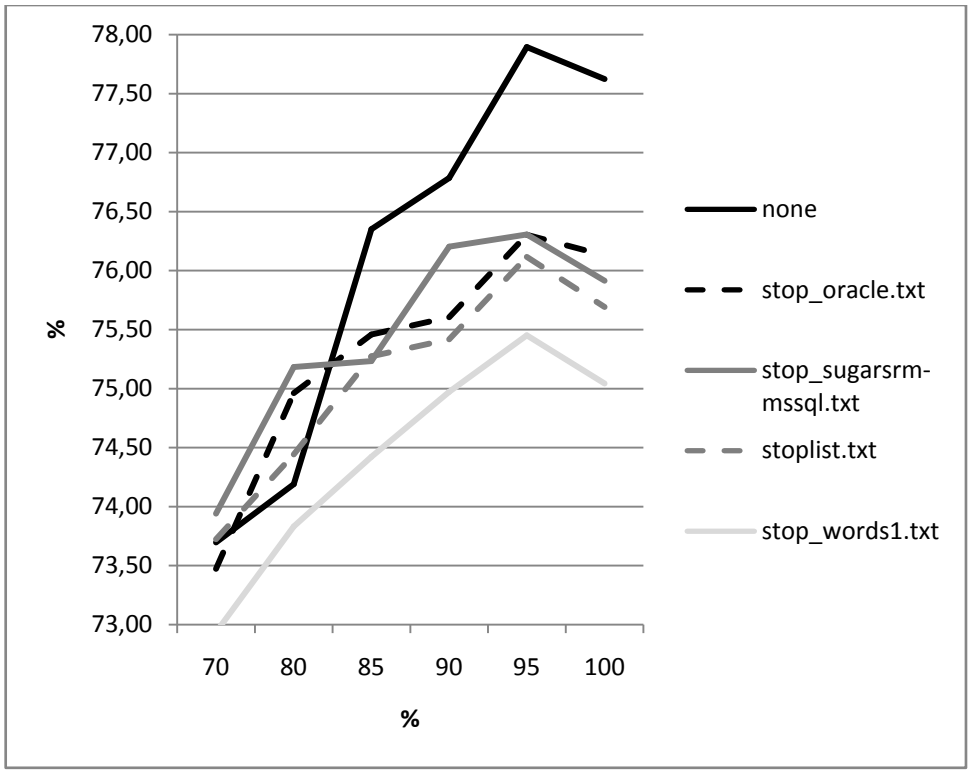
Seznamy by měly sloužit pro zlepšení přesnosti klasifikátoru odstraněním zbytečných slov. Provedené testy však prokázaly, že odstraněním jakýchkoli slov, ztrácíme přesnost. Doporučuji tedy při lineární klasifikaci stoplist nepoužívat.

Stoplisty jsem porovnával na Positive i Balanced Winnow s minimální trénovací přesností 90% a minimální délkou dokumentu 20 slov. Celkový počet trénovacích i testovacích dokumentů byl různý, protože každý stoplist odstranil různé množství slov. Maximální počet dokumentů byl 5 596 bez použití stoplistu a minimální 5 272 při použití stop_list1.txt. Počet aktivních kategorií byl u všech použitých stoplistů stejný, a to 109.



Obrázek 17. Graf – nejlepší stoplist

Pro čtyři nejlepší stylisty jsem s testováním pokračoval i pro různé minimální hodnoty trénovací přesnosti. Výsledek je v následujícím grafu.



Obrázek 18. Graf – stoplisty podle minimální přesnosti při trénování

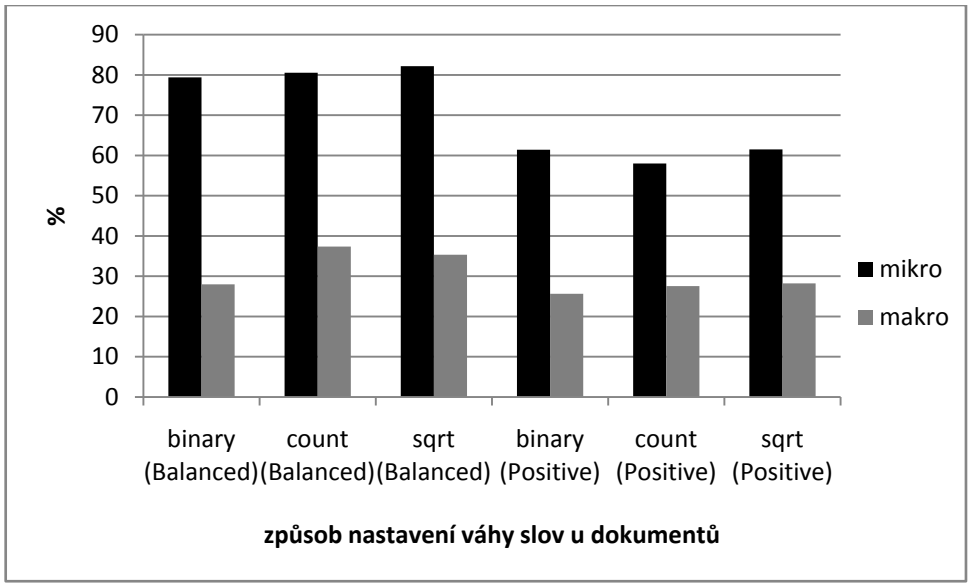
Graf ukazuje jasnou převahu při vynechání stoplistu, ideální bylo natrénování na 95%. Vyšší trénovací přesnost vedla k mírnému přetrénování a snížení hodnoty přesnosti i úplnosti. Při nastavení požadované přesnosti na 100% však bylo dosaženo pouze přesnosti okolo 98%, protože byl překročen maximální počet trénovacích kol.

7.4 Výběr váhy u slov

V kapitole 5.2.3 byly uvedeny možnosti přiřazení vah slovům, která se v dokumentu vyskytují více než jedenkrát.

Výsledky testování neurčily jednoznačného vítěze pro způsob výpočtu vah. Hodnoty break-even pointu byly pro všechny tři možnosti velmi podobné. Nejlepších hodnot dosáhlo použití odmocniny počtu výskytů v dokumentu.

Pokud nebude uvedeno jinak, budu tedy při výpočtu vah používat tuto možnost.

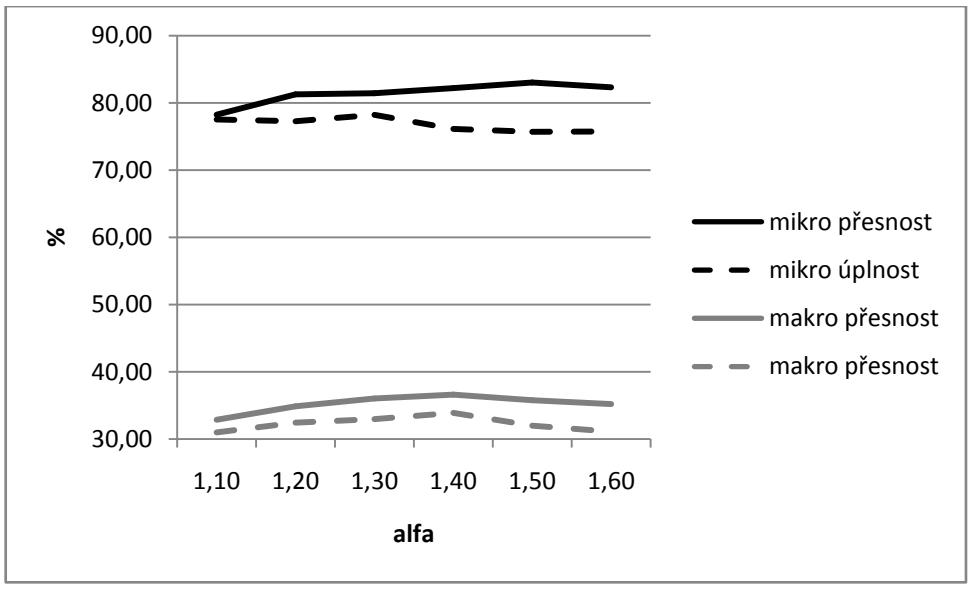


Obrázek 19. Graf – způsoby nastavení vah slov u dokumentů

7.5 Nastavení parametrů α, β, θ

V literatuře [10] se jako doporučené nastavení parametrů alfa, resp. beta uvádí hodnoty 1.1, resp. 0.9, práh θ by měl mít hodnotu 1.

Změna hodnoty prahu se neprojevila ani na výsledné přesnosti (a úplnosti) ani na počtu kol potřebných pro natrénování. Při změně parametrů alfa a beta však docházelo ke změnám ve výkonu i počtu trénovacích kol. Výsledky je možné porovnat v grafu na obrázku Obrázek 20. Mimo zvýšení přesnosti došlo i k rychlejšímu natrénování. Při hodnotě α=1.1 bylo potřeba 16 trénovacích kol, při α=1.4 to bylo jen 9 kol.



Obrázek 20. Graf – nastavení parametrů pro úpravu vah kategorií

Nutno poznamenat, že při zvýšení hodnoty α, byla odpovídajícím způsobem snížena hodnota parametru β, např. α=1.3 znamená použití β=0.7

Testování probíhalo na 7 000 trénovacích a 3300 testovacích dokumentech (poměr 15:7) při odstranění kategorií, které obsahují méně než 20 dokumentů. Nejlepších výsledků bylo dosaženo pro parametry α=1.3 a β=0.7.

7.6 Trénovací přesnost

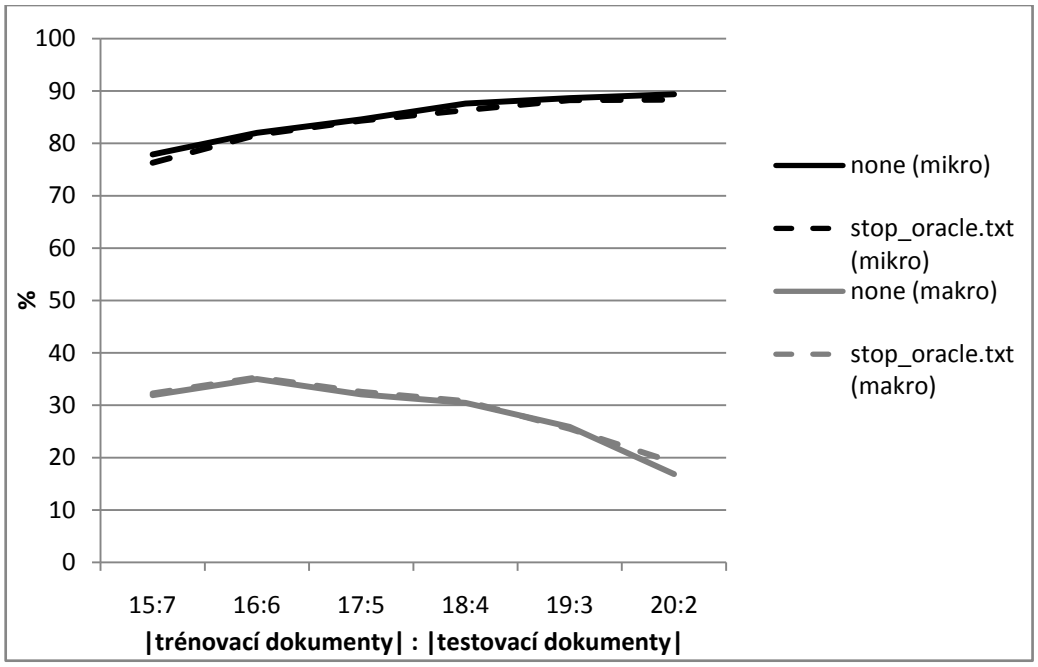
Volba minimální požadované přesnosti při trénování má vliv nejen na výsledek klasifikace, ale i na počet kol potřebných k dosažení této přesnosti a tím na časovou náročnost běhu celého algoritmu. Verze Balanced se učí rychleji a je proto možné požadovat vyšší přesnost při trénování (až 95%), naopak u verze Positive je vhodné použít nižší hodnotu požadované přesnosti (70-80%). Více na grafu Obrázek 16.

7.7 Velikost trénovacích a testovacích dat

Dokumenty se sady Reuters-21578 jsou rozděleny do 22 souborů, lze je proto dělit na trénovací a testovací podle prostého rozdělení souborů na trénovací a testovací. Rozdělení u Lewis splitu je přibližně 15:7 (15 trénovacích dokumentů na 7 testovacích), u Apte splitu je to zhruba 16:6.

Provedl jsem testování za účelem zjistit, zda by jiné rozdělení mohlo znamenat výraznou změnu výsledné přesnosti.

Na grafu Obrázek 21, uvedeném níže, je vidět se vzrůstající převahou trénovacích dokumentů stoupání mikro hodnot pro přesnost i úplnost. To je ale kompenzováno stále nižšími hodnotami pro makro přesnost a úplnost. Je to způsobeno tím, že většina malých kategorií neobsahuje žádné klasifikovatelné dokumenty a má proto přesnost i úplnost rovnu nule.



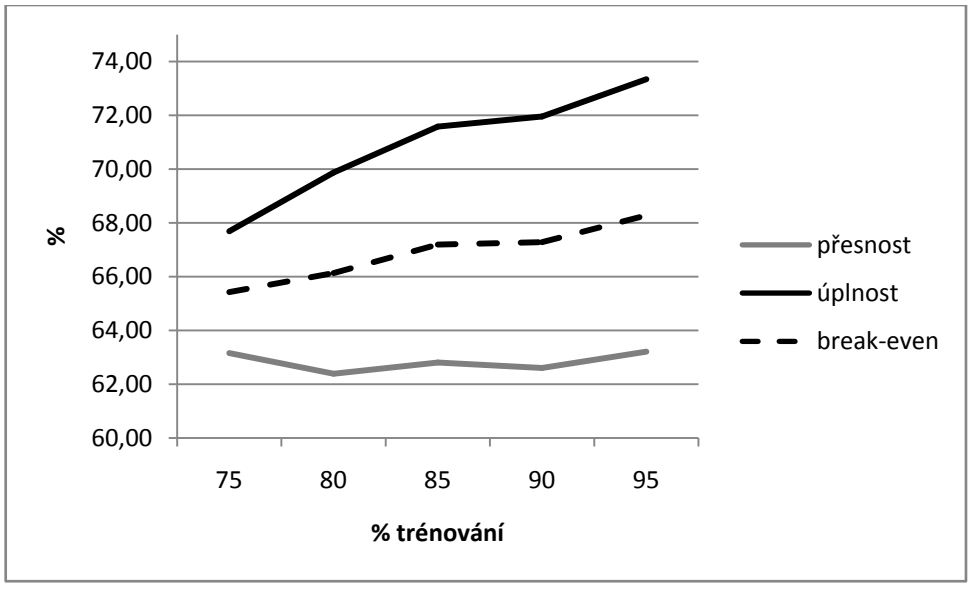
Obrázek 21. Graf – vliv velikosti trénovacích a testovacích dat

Nejvhodnější rozdělení pro klasifikaci je podle dosažených výsledků 16:6, které odpovídá Apte splitu a je u něj dosaženo lepších mikro i makro hodnot než u rozdělení Lewis split.

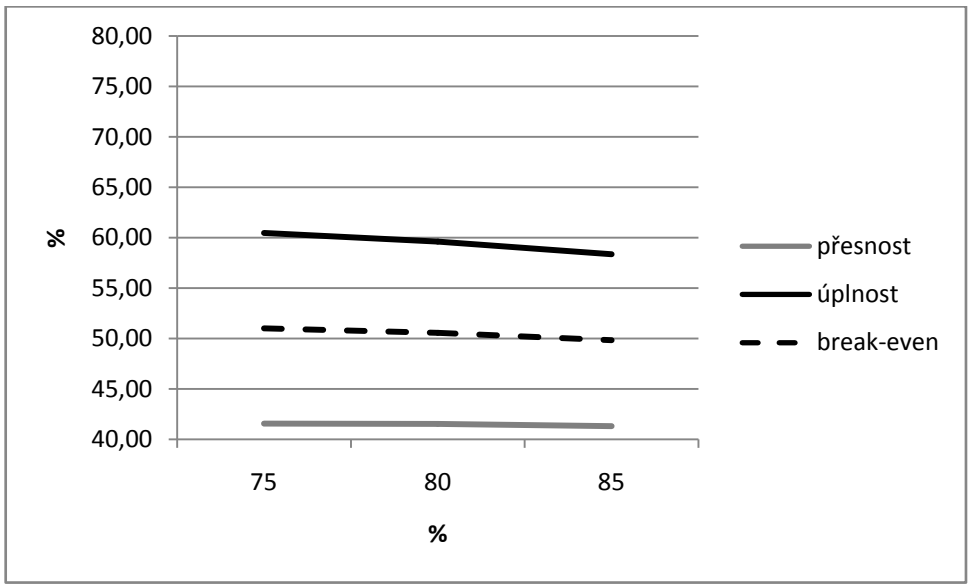
7.8 Lewis Split

Pro porovnání výsledků různých klasifikátorů slouží rozdělení dokumentů Lewis split a Apte split.

Lewis split obsahuje i dokumenty, které nejsou zařazeny do žádné kategorie. Výsledky na této sadě proto jsou horší než u Apte splitu. Nejprve bylo testováno nastavení minimální trénovací přesnosti. U Balanced Winnow bylo nejlepších výsledků dosaženo pro trénovací úspěšnost 95%, kde výsledná hodnota break-even pointu byla 68.28%. Positive Winnow zde dopadl hůře, nejlepším výsledkem je hodnota 51.02%. Grafická znázornění výsledků jsou na grafech Obrázek 22 a Obrázek 23.

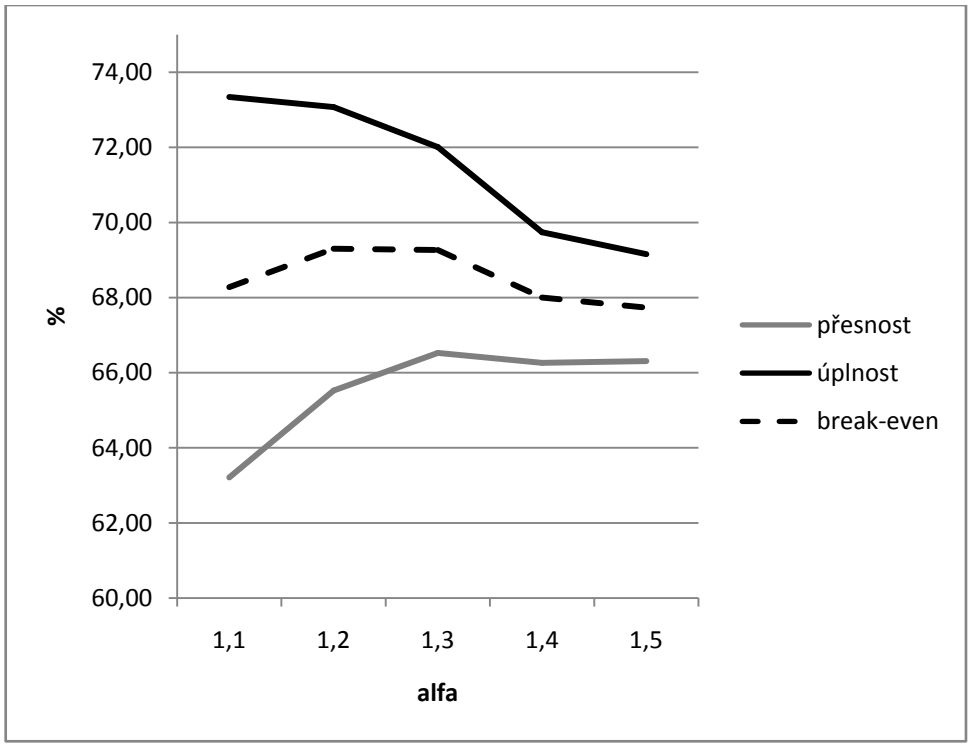


Obrázek 22. Graf – Lewis split: nastavení minimální přesnosti trénování u Balanced Winnow



Obrázek 23. Graf – Lewis split: nastavení minimální přesnosti trénování u Positive Winnow

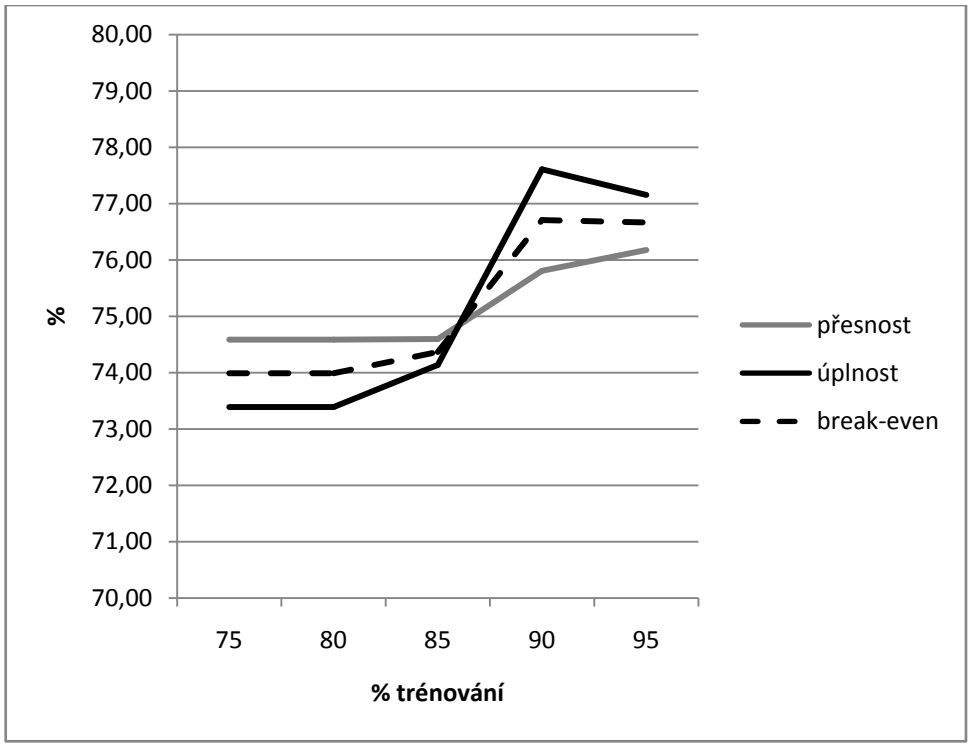
Při nalezené nejlepší hodnotě trénování jsem mohl pokračovat nalezením ideálního parametru alfa, z testu v kapitole 7.5 se nabízela hodnota α=1.3. Výsledky na Lewis splitu určily jako nejlepší hodnotu α=1.2 s dosaženým break-pointem 69.30% (což je o více než 1% lepší výsledek v porovnání s α=1.1). Při použití α=1.3 bylo dosaženo výsledku 69.27%.



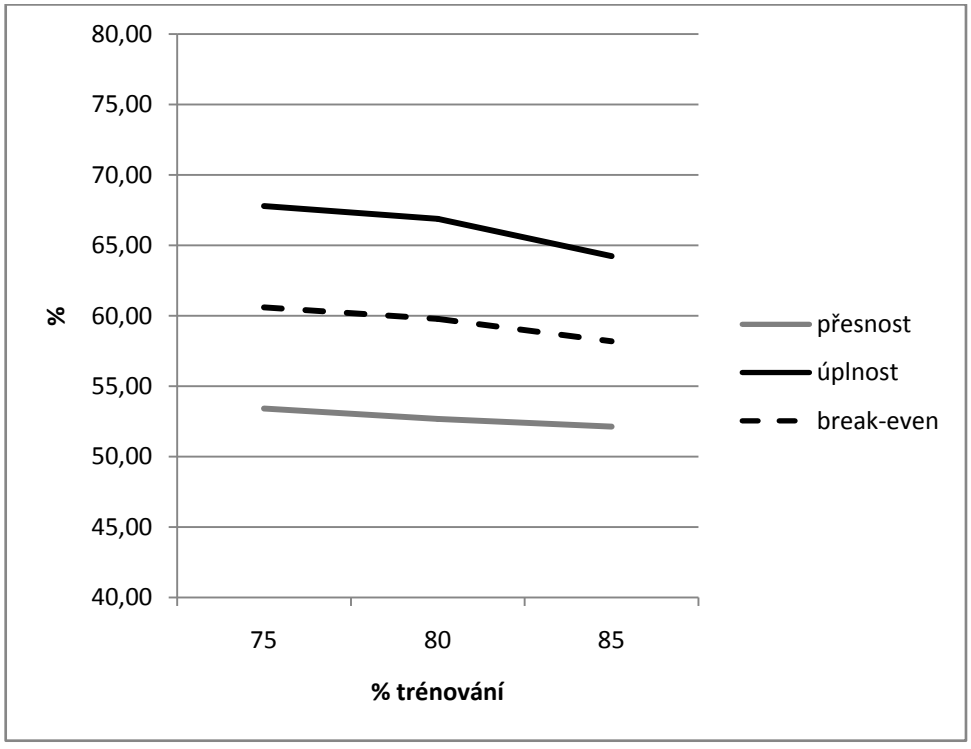
Obrázek 24. Graf – Lewis split: nastavení parametrů α, β u Balanced Winnow

7.9 Apte Split

Apte split obsahuje menší množství dokumentů než Lewis split. Apte split by měl obsahovat jen dokumenty, které jsou zařazeny do nějaké kategorie. Ve skutečnosti to však není pravda a obsahuje i dokumenty bez přiřazených kategorií. Výsledky zde budou lepší než u Lewis splitu, ale horší než při použití mého rozdělení, které počítá opravdu jen s dokumenty zařazenými do některé z kategorií.

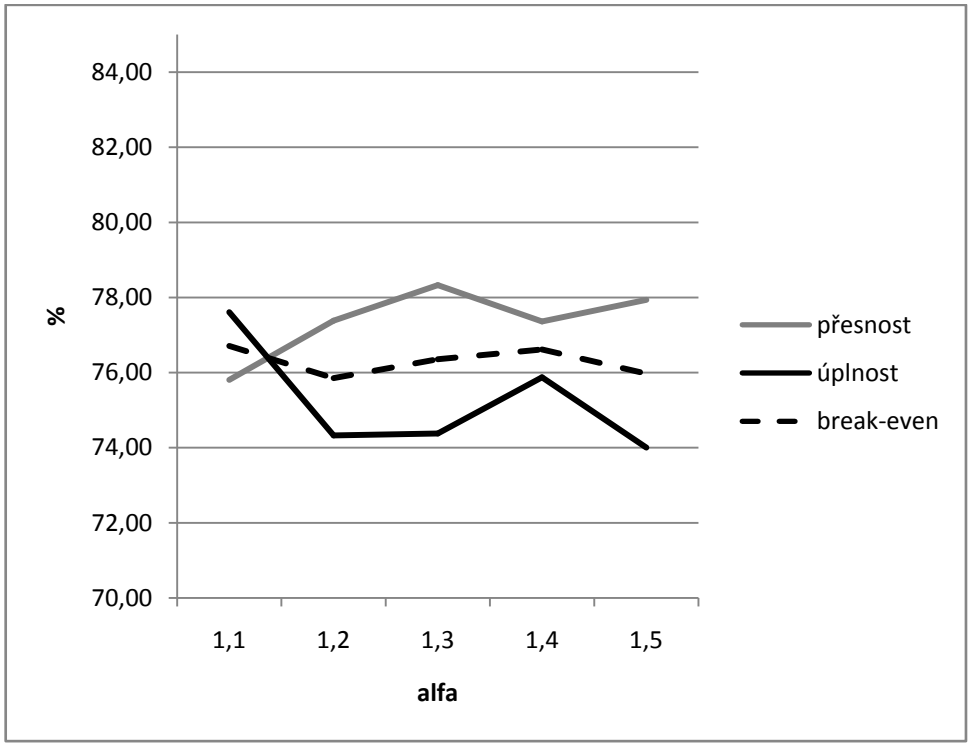


Obrázek 25. Graf – Apte split: nastavení minimální přesnosti trénování u Balanced Winnow



Obrázek 26. Graf – Apte Split: nastavení minimální přesnosti trénování u Positive Winnow

Nejlepšího výsledku bylo dosaženo při natrénování s přesností 90%. S tímto nastavením jsem pokračoval stejně jako u rozdělení Lewis split, tedy hledal jsem nejvhodnější parametry alfa a beta. Výsledky na Apte splitu určily jako nejlepší hodnotu α=1.1 s dosaženým break-pointem 76.71%.



Obrázek 27. Graf – Apte split: nastavení parametrů α, β u Balanced Winnow

Nejlepšího výsledku této implementace Balanced Winnow bylo dosaženo při trénování na 7 237 dokumentech a testování na 3 087 dokumentech (poměr přibližně 15:7) bez použití stoplistu (Sychra split). Nastavení bylo α=1.3, β=0.7, θ=1, minimální přesnost natrénování 95%, použity byly pouze dokumenty obsahující více než 20 slov, váha slov odpovídala odmocnině počtu jejich výskytů. Výsledná hodnota break-even pointu byla 79.23%.

	Lewis split	Apte split	Sychra split
Positive Winnow	51.0%	60.6%	62.2%
Balanced Winnow	69.3%	76.7%	79.2%

8 Závěr

V úvodní teoretické části jsem uvedl typy metod pro získávání znalostí z textových dokumentů. Nejvíce využívané jsou metody pro klasifikaci, které jsem důkladněji rozdělil a popsal. Zajímavá je skupina lineárních klasifikátorů, z nichž algoritmus Winnow se jeví jako algoritmus dosahující nejlepších výsledků.

Při ohodnocení výsledků jsem uvedl makro i mikro hodnoty přesnosti i úplnosti. Pro porovnání jsem nejvíce používal hodnotu break-even pointu pro mikro přesnost a úplnost, která velmi dobře prezentuje úspěšnost klasifikace.

Výsledky potvrzují lepší výkon verze Balanced Winnow oproti Positive Winnow jak v úspěšnosti klasifikace, tak i v rychlosti učení. Balanced Winnow k natrénování stačí méně než 10 kol.

Winnow pro klasifikaci nepotřebuje stoplist, při jeho použití dosahuje horších výsledků než bez něj. Dokáže totiž zpracovat velké množství atributů a i nevýznamová slova pro něj obsahují jistou informaci.

Všechny výsledky algoritmu jsou shrnuty v poslední kapitole 7 (Výsledky), na rozdělení Lewis split bylo dosaženo hodnoty break-even pointu 69.3%, na rozdělení Apte split 76.7%. Výsledky jsou porovnatelné s hodnotami dosaženými v [10]. V závěru kapitoly jsem uvedl nastavení, při kterém bylo dosaženo nejlepšího výsledku.

Tato základní verze programu by mohla být vylepšena implementací rozšíření uvedených v kapitole 5.2. Bylo by také vhodné otestovat použití Winnow na jiných datech, zařazení článků v sadě Reuters-21578 totiž není úplné a Winnow dosahuje lepších výsledků, pokud jsou použity jen dokumenty zařazené do kategorií.

Další testování doporučuji např. v oblasti filtrování elektronické pošty od nevyžádaných zpráv. Bylo by zajímavé experimentovat s nastavením tak, aby byla zajištěna vysoká hodnota úplnosti (aby nebyla normální zpráva označena jako spam) a zároveň co největší hodnota přesnosti.

Seznam obrázků

Literatura

[1] WordNet. [Online] [Citace: 1. 1 2008.] http://wordnet.princeton.edu/.

[2] Wikipedia. [Online] [Citace: 1. 1 2008.] http://en.wikipedia.com.

[3] **Rushing, John A.** [Online] 2004. [Citace: 1. 1 2008.] http://www.cs.uah.edu/~jrushing/cs696-summer2004/notes/Advanced.ppt.

[4] **Apte, Chidanand, Damerau, Fred a Weiss, Sholom M.** *Text Mining with Decision Trees and Decision Rules.* 1998.

[5] **Koster, Cornelis H.A.** Text classification. [Online] 2003. [Citace: 1. 1 2008.] http://www.cs.ru.nl/~kees/ir2/papers/h03.pdf.

[6] **Han, Eui-Hong (Sam), Karypis, George a Kumar, Vipin.** Text Categorization Using Weight Adjusted k -Nearest Neighbor Classification. [Online] 2001. http://citeseer.ist.psu.edu/article/han99text.html.

[7] **Zendulka, doc. Ing. Jaroslav.** *Získávání znalostí z databází - Studijní opora.* Brno : FIT VUTBR, 2006.

[8] **Frigui, Hichem a Nasraoui, Olfa.** *Simultaneous Categorization of Text Documents and Identification of Cluster-dependent Keywords.* Memphis : Department of Electrical and Computer Engineering University of Memphis, 2002.

[9] **Cortes, Corinna a Vapnik, Vladimir.** *Support-Vector Networks.* 1995.

[10] **Dagan, Ido, Karov, Yael a Roth, Dan.** *Mistake-Driven Learning in Text Categorization.* 1997.

[11] **Luz, Saturnino.** *Implementing a Text Categorisation System: a step-by-step tutorial.* Dublin : Dept of Computer Science, Trinity Computer Dublin, 2007.

[12] **Brückner, Michael, Haider, Peter a Scheffer, Tobias.** *Highly Scalable Discriminative Spam Filtering.* 2006.

[13] **Kotsiantis, Sotiris a Kanellopoulos, Dimitris.** *Association Rules Mining: A Recent Overview.* místo neznámé : Department of Mathematics, University of Patras, Greece, 2006.

[14] **Burda, Ing. Michal.** *Získávání znalostí z databází - asociační pravidla.* Ostrava, 2004.

[15] **Sehgal, Aditya Kumar.** *Text Mining: The search for novelty in text.* 2004.

[16] **Bear, John, a další.** *Using Information Extraction to Improve Document Retrieval.* Menlo Park, 1998.

[17] **Lewis, David D. a Ringuette, Marc.** *A Comparison of Two Learning Algorithms for Text Categorization.* Las Vegas, 1994.

[18] **Mun, P.P.T.M. van.** *Text Classification in Information Retrieval using Winnow.* Nijmegen : Department of Computing Science, Catholic University of Nijmegen.

[19] **Uden, Mark van.** *Rocchio: Relevance Feedback in Learning Classification Algorithms.* Nijmegen : Department of Computing Science, University of Nijmegen.

[20] **Apte, Chidanand, Damerau, Fred a Weiss, Sholom M.** *Towards Language Independent Automated Learning of Text Categorization Models.* 1994.

[21] **Joachims, Thorsten.** *Text Categorization with Support Vector Machines: Learning with Many Relevant Features.* Dortmund : Universität Dortmund, 1997.

[22] **Sebastiani, Fabrizio.** *Text Categorization.* Padova : Dipartimento di Matematica Pura e Applicata, Universita di Padova, 2005.

[23] **Hotho, Andreas, Staab, Steffen a Stumme, Gerd.** *Ontologies Improve Text Document Clustering.* Karlsruhe : Institute AIFB, University of Karlsruhe, 2003.

[24] **Hotho, Andreas, Staab, Steffen a Stumme, Gerd.** *Explaining Text Clustering Results using Semantic Structures.* Karlsruhe : Institute of Applied Informatics and Formal Description Methods AIFB, University of Karlsruhe, 2003.

[25] **McCallum, Andrew a Nigam, Kamal.** *A Comparison of Event Models for Naive Bayes Text Classification.* Pittsburgh, 1998.

[26] **Yang, Yiming a Pedersen, Jan O.** *A Comparative Study on Feature Selection in Text Categorization.* Nashville, 1997.

Seznam příloh

Příloha 1. Přehled stoplistů

1. **stop_words1.txt**

   39 slov

   zdroj: Uhlíř Ing. Martin, Metody pro získávání asociačních pravidel z dat, Brno 2007

2. **stoplist.txt**

   66 slov

   zdroj: http://www.csc.kth.se/~xmartin/java/

3. **stop_oracle.txt**

   76 slov

   zdroj: http://download.oracle.com/docs/cd/A58617_01/cartridg.804/a58165/appa.htm#2067

4. **stop-mssql.txt zdroj:**

   154 slov

   http://www.sugarcrm.com/wiki/index.php?title=Overview_of_Full_Text_Stop_Words

5. **stop_uned-es.txt**

   250 slov

   zdroj: http://terral.lsi.uned.es/~ircourse/examples/stoplist.html

6. **stopwds.txt**

   319 slov

   zdroj: http://ronaldo.cs.tcd.ie/esslli07/

7. **stop -mysql.txt**

   544 slov

   zdroj: http://dev.mysql.com/doc/refman/5.0/en/fulltext-stopwords.html

8. **stop_uned-es-smart.txt**

   568 slov

   zdroj: http://terral.lsi.uned.es/~ircourse/examples/stoplist.html