TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

DIPLOMOVÁ PRÁCE

Liberec 2008 **Jiří Černý**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: M 2612 – Elektrotechnika a informatika

Studijní obor: 3902T005 – Automatické řízení a inženýrská informatika

Automatická detekce témat

Automatic detection of topics

Diplomová práce

Autor:	**Jiří Černý**
Vedoucí DP práce:	Ing. Jindra Drábková, Ph.D.
Konzultant:	Ing. Jan Kolorenč, Ph.D.

V Liberci 15. května 2008

Zadání

Automatická detekce témat

Zásady pro vypracování:

1. Seznamte se s prostředím programovacího jazyka Perl.
2. Pomocí balíku LWP pro jazyk Perl získejte vhodná trénovací a testovací data pro

   klasifikátor a roztřiďte je.
3. Seznamte se s články zabývajícími se automatickou klasifikací dokumentů a proveďte

   jejich rešerši.
4. Seznamte se se systémem WEKA.
5. Porovnejte metody klasifikace a různé parametrizace textu.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah průvodní zprávy: cca 40 až 50 stran

Seznam odborné literatury:

[1] http://www.cs.waikato.ac.nz/∼ml/weka/

[2] http://nlp.fi.muni.cz/nlp/files/index_DB.html

[3] Richard O. Duda, Peter E. Hart, David G. Stork: Pattern Classification

[4] Vladimír Mařík, Olga Štěpánková, Jiří Lažanský, kolektiv: Umělá inteligence(4)

Prohlášení

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

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do její skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Velké poděkování patří rodině a přátelům za morální podporu během mého studia. Největší poděkování však patří Ing. Jindře Drábkové, Ph.D. a Ing. Janu Kolorenčovi, Ph.D. za odborné rady a čas, které mi věnovali během přípravy této diplomové práce.

Abstrakt

Cílem diplomové práce je najít takový postup, kterým lze úspěšně roztřídit množství neoznačených textových dokumentů podle typických příznaků, které nese každá skupina textových dat popisujících určité téma. Znamená to nacvičit klasifikátor na dostatečně velké množině trénovacích dat a otestovat jeho úspěšnost na testovacích množinách dat. Jako trénovací a testovací data jsou použity články ze serveru `zpravy.atlas.cz`. V první části se práce zabývá především vyhledáním a zhodnocením informací o automatické detekci témat, získáním a přípravě dat, druhá část se zaměřuje na nalezení vhodného klasifikátoru pomocí programu WEKA.

Pro zpracování velkého množství textu se hodí programovací jazyk Perl, který obsahuje vhodné a rychlé nástroje. Pro výtah dat z HTML souborů je použit balík LWP, který obsahuje funkce pro stažení souborů z internetu, dokáže analyzovat strukturu HTML dokumentu a vybrat z ní potřebné informace – texty článků.

Protože prostý text není vhodným tvarem pro další zpracování, vytvoří se globální slovník obsahující všechna slova vyskytující se v sadě dat. Pomocí něho se převedou dokumenty na příznakové vektory, kde každá jednotlivá pozice ve vektoru představuje jedno slovo ze slovníku. Tento vektor může mít různé reprezentace, které zohledňují výskyt příznaků v rámci jednotlivých dokumentů, témat, případně celé sady dat. Číselná hodnota vypovídá o četnosti výskytu, případně váze slova.

Silným nástrojem pro trénování klasifikátorů, shlukovou analýzu a hledání příznaků je program WEKA. Součástí práce je krátké seznámení s jeho možnostmi a ovládáním. Poté je pomocí něho testován vliv různých reprezentací příznakového vektoru a algoritmů klasifikátorů na úspěšnost třídění.

Abstract

The aim of diploma thesis is to find a sufficient sequence which can sort out unsigned text documents. It means to prepare a lot of training data for classifier learning. The fruitfulness of classifier is tested by the help of testing data. Newspaper articles from server zpravy.atlas.cz are used as a testing data. The first part of diploma thesis is about automatic detection theory. The second part of diploma thesis is about finding the classifier by the help of program WEKA.

Data is processed by the help of programming language Perl and package LWP. Simple text isn't suitable for next processing. For this reason a global dictionary is created. Documents are converted into feature vectors. These vectors can be written by the help of different representation. In diploma thesis different sorts of representation are tested. Program WEKA is used for training classifiers, cluster analysis and select attributes. In this program different representation feature vectors and classifiers algorithms are tested.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

aj.	a jiné
atd.	a tak dále
atp.	a tak podobně
apod.	a podobně
ARFF	Attribute-Relation File Format, datový soubor pro program WEKA
HTML	HyperText Markup Language, značkovací jazyk
LWP	Library for WWW Access in Perl, knihovna pro programovací jazyk Perl
např.	například
NTF	Normalised Term Frequency, normovaná frekvenční reprezentace
TF	Term Frequency, frekvenční reprezentace
TF-IDF	Term Frequency - Inverse Document Frequency, frekvenční reprezentace s inverzní frekvencí
tj.	to jest
tzv.	takzvaně
vs.	versus

Úvod

Se stále vzrůstajícím počtem elektronických dokumentů vyvstává potřeba umět je účinně třídit do příslušných tříd podle obsahu každého jednotlivého dokumentu. Nejlepších výsledků se dosahuje, pokud třídění zastává člověk. Avšak ruční třídění je neúměrně zdlouhavé, a tedy neproduktivní, v praxi často dokonce nemožné. Nejen z ekonomických a časových důvodů je třeba tento úkon předat výpočetní technice. Příkladnou aplikací automatické detekce témat, se kterou se lze denně setkat, je třídění elektronické pošty. V tomto případě lze tříděním oprostit uživatele od neustálého promazávání nevyžádané pošty. Je jasné, že musí být použit takový postup, který bezpečně oddělí obě skupiny od sebe. Automatická detekce témat umožňuje snažší práci s elektronickými dokumenty.

Automatická detekce témat je nový a rychle se rozvíjející obor zabývající se dobýváním znalostí z textových dokumentů a jejich dalším zpracováním. Cílem práce je zachycení problematiky kolem třídění textů v teoretické i praktické rovině. Teoretická část bude zaměřena především na druhy reprezentací dat, popis různých typů algoritmů a seznámení se s programovacím jazykem Perl, balíkem LWP a systémem pro dobývání znalostí WEKA. V praktické části jsou zachyceny postupy od získání cvičných dat přes jejich úpravu až po vyhledávání příznaků a klasifikaci. V závěru budou vybrány a zhodnoceny metody nejvhodnější pro praktické použití.

1 Rešerše

1.1 Automatická klasifikace

Automatická klasifikace je algoritmus, pomocí něhož je možné třídit textové dokumenty podle obsahu do různých tříd. Třídy v tomto případě představují tématické celky [16]. Po aplikaci takového algoritmu vzniknou shluky (clustery), které budou obsahovat podobné dokumenty. Algoritmus automatické klasifikace – tzv. klasifikátor – se trénuje na jedné sadě známých dat a testuje se na jiné sadě známých dat. Dobře nacvičený klasifikátor dokáže sám roztřídit dokumenty podle podobnosti k jednotlivým vzorům do příslušných tříd [7].

Rozpoznávání tříd – objekt (článek) se zařazuje na základě podobnosti jeho obrazu (vektorový popis) do jedné ze tříd (témat) [1].

Třída – je skupina objektů s podobnými vlastnostmi.

Obraz – popis objektů pomocí příznaků (v případě klasifikace textů pomocí slov, vystihujících danou třídu).

Podobnost – vlastnost měřitelná na obrazu objektu, vyjadřující vztah ke každé ze tříd.

Aby bylo možné automaticky třídit dokumenty, je třeba nejprve natrénovat klasifikátor. Ten musí umět správně rozpoznávat jednotlivé třídy dokumentů mezi sebou. Při rozpoznávání se využívá znalostí získaných již dříve – ve fázi učení. Ve fázi učení je třeba roztřídit co největší množství dat do jednotlivých tříd, do kterých se budou data později třídit automaticky. Tato data se dělí na dvě množiny – trénovací a testovací. Na trénovací množině se stanovují příznaky – nacvičení klasifikátoru a ten se testuje na testovací množině. Po roztřídění testovací množiny nacvičeným klasifikátorem se výsledek srovnává s původním rozdělením dokumentů, a tak se dá stanovit úspěšnost klasifikátoru. Nereálně dobrých výsledků

se dosahuje při testování na trénovací množině.

Cvičení klasifikátoru z více tříd se nazývá učení s učitelem (supervised learning). Třídy mohou obsahovat jak pozitivní, tak negativní příklady. Pozitivním příkladem se rozumí příznaky, které by dokument měl obsahovat, negativní příklady by se naopak v dokumentu vyskytovat neměly.

K nacvičení klasifikátoru stačí i jediná třída – částečné učení s učitelem (semi-supervised learning). Tento klasifikátor rozděluje dokumenty do dvou tříd. V praxi se využívá například k oddělení spamu – třídění vyžádaných vs. nevyžádaných zpráv.

Klasifikátor lze nacvičit i pomocí učení bez učitele (unsupervised learning). Takto nacvičeným klasifikátorem se rozdělují dokumenty z neoznačených tříd na základě jejich podobnosti k některé ze tříd. Z dokumentů stejných tříd vzniknou shluky a podle nich se dále třídí do příslušných tříd.

1.2 Zpracování textů

Vstupní data se používají ve formě textu bez formátovacích znaků. Jednotlivá slova jsou od sebe oddělena mezerou. Znaky jako tečky, čárky, uvozovky atd. jsou sice ve vstupních datech obsaženy, ale při lexikální analýze se odstraňují.

Extrakce dokumentů

Z důvodů velkého množství dat, které je nutno zpracovat, se nepracuje s celými dokumenty. Přistupuje se k tzv. příznakovému popisu dokumentu – dokument představují jen vybraná slova. Lexikální analýzou se zjistí veškerá slova nebo sousloví obsažená v dokumentech zařazených do jednoho tématu a ty se zaznamenávají do slovníku pro dané téma. Lexikální analýza probíhá tak, že se prochází dokumentem a jsou vybírány znakové řetězce, oddělené od sebe mezerami. Z těchto řetězců se poté odstraní znaky jako již zmíněné čárky a tečky. Tímto způsobem se získávají slova, která se zaznamenávají do slovníku [4].

Stoplist

Stoplist je negativní slovník, který obsahuje nejfrekventovanější slova v daném jazyce, spojky, předložky, částice apod. Ztráta těchto slov má na přesnost klasifikace minimální vliv. Využívá se při redukci slovníku.

Redukce slovníku

Slovníky obsahují značné množství slov, přičemž ke klasifikaci je využitelná jen malá část z nich. Při redukci slovníku se odstraňují slova obsažená ve stoplistu, všechna nevýznamová slova – tj. slova nemající význam (předložky, spojky, aj.) – a slova mající vysoký výskyt ve většině dokumentů. Pro účel klasifikace nemají tato slova význam. Slovník se po redukci mnohonásobně zmenší. Protože se všechny dokumenty později nahradí příznakovým vektorem, je menší velikost slovníku výhodná při dalším zpracování. Menší velikost příznakových vektorů urychluje práci s dokumenty.

Problémem při zpracovávání slovníku může být nejednoznačnost slov. Jsou to jednak homonyma – jeden tvar slova má více významů (např. kohout), anebo synonyma – různá slova mající stejný význam (láhev × flaška). Synonyma zbytečně zvětšují počet slov ve slovníku, protože lze více slov nahradit jediným. Homonyma počet slov ve slovníku naopak snižují, ale mohou zhoršit výsledky klasifikace.

K redukci slovníků se využívá stemizace – převodu na kořen slova nebo lemmatizace – převodu na základní tvar slova. Při lemmatizaci se jednotlivá slova srovnávají se slovy z lemmatizačního slovníku a zpět se dosazuje slovo v základním tvaru (vodník – vodníci, vodníkům, vodníky, vodníkovo, atp). Pro češtinu se používá např. morfologický analyzátor Ajka[1], který umí převádět slova do základního tvaru [6].

Porovnáním výskytů slov v těchto slovnících se vyberou slova, jež se výrazně častěji vyskytují jen v jednom z nich. Z těchto slov se vytvoří nový slovník, který se využije k příznakovému popisu. Tento slovník tedy bude obsahovat všechna významná slova z jednotlivých tříd dokumentů, která určují příslušnost k třídě.

[1] http://nlp.fi.muni.cz/projekty/ajka/ajkacz.htm

1.3 Typy reprezentací příznakového vektoru

Každý neznámý dokument se nahrazuje příznakovým vektorem. Délka vektoru je stejná s počtem slov ve slovníku, podle kterého budou dokumenty popisovány. Každá jednotlivá pozice vektoru zastupuje jedno slovo slovníku. Slova ve vektoru zastupuje číselná hodnota. Používají se různé reprezentace příznakového vektoru [4]. Volbou správné reprezentace se dá znatelně zlepšit úspěšnost klasifikace. Nevýhodou příznakového popisu je ztráta informace o pozici slov v dokumentu.

1.3.1 Binární reprezentace

Základní reprezentace podávající pouze informaci o tom, zda slovo v dokumentu je (1) či nikoli (0). I přesto se s ní dosahuje v mnoha algoritmech dobrých výsledků.

1.3.2 TF (Term Frequency) – frekvenční reprezentace

Zaznamenává se frekvence slova normovaná počtem slov v dokumentu. Oproti binární reprezentaci poskytuje více informací o frekvenci výskytu daného příznaku.

$$TF(i) = \frac{s}{p} \tag{1}$$

s – počet výskytů slova v dokumentu

d – celkový počet slov v dokumentu

1.3.3 NTF (Normalised Term Frequency) – normovaná frekvenční reprezentace

Tato reprezentace vychází z frekvenční reprezentace. Hodnoty frekvence slova v dokumentu při TF jsou velmi nízké a pokud je jejich výskyt větší než nastavený práh, je tato hodnota uměle zvětšena. Dále se zjednoduší klasifikace, protože slova s výskytem nižším než prahovým jsou ignorována. Slova vyskytující se častěji jsou zvýrazněna přičtením hodnoty 0,5 a znásobením hodnotou slova nejvíce zastoupeného v dokumentu.

$$NTF(i) = 0 \quad \text{pro} \quad TF(i) <= práh \tag{2}$$

$$NTF(i) = 0{,}5 + \frac{TF(i)}{2 \cdot max(TF(i))} \quad \text{pro} \quad TF(i) > práh \tag{3}$$

$TF(i)$ – frekvence výskytu v dokumentu

$max(TF(i))$ – frekvence výskytu nejčastěji zastoupeného slova v dokumentu

práh – prahová hodnota pro ignorování slova

1.3.4 TF-IDF (Term Frequency-Inverse Document Frequency) – frekvenční reprezentace s inverzní frekvencí

Jde o zahrnutí frekvence slov vzhledem k jeho výskytu v ostatních dokumentech. Čím častěji se určité slovo vyskytuje v ostatních dokumentech, tím je považováno za méně důležité. Vzhledem k i-té pozici bude hodnota vyjádřena vzorcem:

$$TF\text{-}IDF(i) = TF(i) \cdot log\frac{DOC}{DF(i)} \tag{4}$$

$TF(i)$ – frekvence výskytu v dokumentu

DOC – celkový počet dokumentů

$DF(i)$ – počet dokumentů v nichž je dané slovo obsaženo

1.3.5 Hadamardova reprezentace

Hadamardova reprezentace je normovaná frekvence slova v dokumentu násobená frekvencí slova v tréninkové množině. Normovaná frekvence slova je číselné ohodnocení výskytu slova, je to poměr výskytu slova vůči počtu ostatních slov v dokumentu.

$$H(i) = NTF(i) \cdot TF_T(i) \tag{5}$$

$NTF(i)$ – normovaná frekvence výskytu v dokumentu

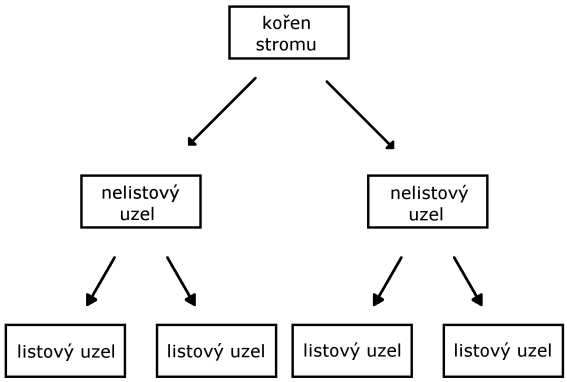
$TF_T(i)$ – frekvence výskytu slova v tréninkové množině

1.4 Metody automatické klasifikace

V následujících odstavcích je popsán přehled některých metod použitých pro automatickou klasifikaci textů [16].

1.4.1 Klasifikace prostřednictvím rozhodovacích stromů

Pomocí rozhodovacího stromu se lze pomocí otázek a odpovědí v nelistových uzlech stromu dostat od kořenu až do listových uzlů představujících dané třídy. Klasifikace textů je založena na postupném dělení prostoru na jednotlivé podmnožiny obsahující především příznaky dané třídy.



Obrázek 1: Rozhodovací strom

1.4.2 Rozhodovací pravidla

Rozhodovací pravidla jsou podobná rozhodovacím stromům. V tomto případě se hledají pravidla pomocí konstrukce IF-THEN, která spolehlivě oddělí různé třídy.

1.4.3 Naivní Bayesův klasifikátor

Naivní Bayesův klasifikátor patří mezi metody maximální pravděpodobnosti, tzn. předpovídá pravděpodobnost členství třídy. Naivní se nazývá proto, že předpokládá nezávislost mezi výskyty jednotlivých slov v dokumentu. Naivní Bayesův klasifikátor se v praxi vyznačuje vysokou rychlostí, přesností rozhodování a nízkou výpočetní náročností.

$$P(Tr/x) = \frac{P(x/Tr)P(Tr)}{P(x)} \tag{6}$$

$P(Tr/x)$ – pravděpodobnost, že x patří do třídy Tr.

$P(x/Tr)$ – podmíněná hustota pravděpodobnosti, udává rozložení pravděpodobnosti vektoru příznaků x pro třídu Tr.

$P(Tr)$ – pravděpodobnost výskytu prvků této třídy

$P(x)$ – hustota rozložení vektoru příznaků nezávisle na třídě

1.4.4 Pravidlo nejbližšího souseda

Pravidlo nejbližšího souseda patří mezi metody minimální vzdálenosti. Je často používanou metodou pro klasifikaci textu. Toto pravidlo přiřadí neznámý dokument do té třídy, k jejímuž představiteli má nejblíže. Třídy mohou být představovány jednotlivými vzorky trénovací množiny, což jsou nejčastěji se vyskytující slova nebo etalony. Etalon je vzorek třídy, který ji nejlépe reprezentuje ve smyslu minimální vzdálenosti. Pokud je třída představována jednotlivými vzorky trénovací množiny, pak ke kladům patří nenáročné trénování a dobré výsledky při rozpoznávání. Je nutná velká trénovací množina a s její velikostí se zvyšuje výpočetní náročnost. Vzdálenost $d(X,Y)$ je často definována jako Euklidovská vzdálenost mezi dvěma body $X=(x_1, x_2, \ldots, x_n)$ a $Y=(y_1, , y_2, \ldots, y_n)$.

$$d(X,\, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{7}$$

V tomto případě je výhodou rovněž nenáročné trénování – výpočet etalonů. Výpočetní náročnost nezáleží na velikosti trénovací množiny, ale na počtu etalonů. Nevýhodou této metody je horší úspěšnost rozpoznávání.

1.4.5 Support Vector Machine – SVM

Support Vector Machine [16] je poměrně nová a často využívána metoda pro potřeby klasifikace. Do češtiny se dá název přeložit jako metoda podpůrných vektorů. Podstatou této metody je oddělení dvou lineárně separabilních tříd pomocí rozdělující nadroviny. Tato nadrovina je stanovena pomocí pomocných vektorů, které jsou získány z trénovacích příkladů ležících co nejblíže dělící rovině. Tato metoda je oproti pravidlům nejbližšího souseda a rozhodovacím pravidlům vhodná především pro numerická data.

2 Získání trénovacích a testovacích dat

2.1 Zdroje dat

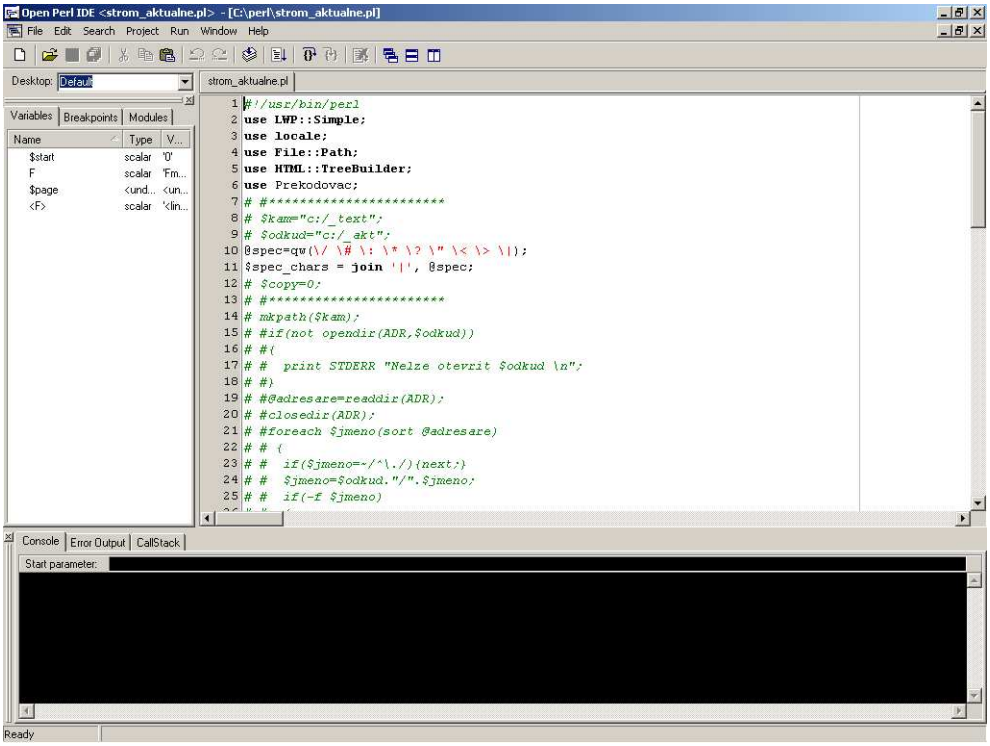
Jako zdroj trénovacích a testovacích dat se dají použít různé internetové noviny. V našem případě to jsou například Britské listy, které se zabývají nejen politickou situací v České republice a ve světě, ale věnují se i důležitým událostem, které leží mimo hlavní zájem nejrozšířenějších médií, Deník Sport a počítačový časopis PC World. Protože je potřeba poměrně velké množství dat, bylo by neúměrně náročné stahovat články ručně a ručně z nich kopírovat čistý text. Jako nejúčinější se ukázalo prostudovat strukturu příslušné webové stránky a vhodně napsaným programem z ní vybrat čistý text. Jako ideální programovací jazyk k tomuto účelu se jeví Perl s využitím balíků `LWP::Simple`, `File::Path`, `HTML::TreeBuilder` a s podporou `locales` [8], [10]. Perl je v praxi často používaný jazyk pro práci s textem. Především práce s regulárními výrazy, které umožňují vyhledávání v textu, nahrazování a zpracování velkého počtu souborů, je silnou stránkou Perlu. Samotný jazyk Perl je syntaxemi a zápisem velmi podobný jazyku C, což usnadňuje přechod k Perlu z jazyků podobných C.

2.2 Perl

Perl je rychlý interpretovaný jazyk nezávislý na platformě, je volně šiřitelný. Byl vyvinutý v prostředí Unixu, ale lze ho provozovat pod mnohými operačními systémy včetně Windows. Použít se dá například volně šiřitelná distribuce ActivePerl od společnosti ActiveState[2]. Perlovské programy se spouštějí na příkazové řádce, ale lze využít i integrovaného vývojového prostředí spolupracujícího se standardní instalací Perlu. V operačním systému Windows to je například OpenPerl IDE[3], viz obr. 1. Stejně jako Perl je i tato aplikace volně šiřitelná.

[2] http://www.activestate.com/Products/ActivePerl

[3] http://www.lost-sunglasses.de



Obrázek 2: Prostředí OpenPerl IDE

2.3 Balík LWP

LWP je zkratka – Library for WWW access in Perl. LWP je balík modulů, který obsahuje řadu funkcí, například jednoduché stažení stránky funkcemi `get($url)` nebo `getstore($url,$file)` z modulu `LWP::Simple`. Funkce `get` potřebuje k uložení souboru navíc ještě trojici funkcí – `open, print a close`. V případě neúspěchu vrací jen hodnotu `undef`. Při použití `getstore` nepotřebujeme výše uvedené funkce pro práci se soubory. Funkce `getstore` navíc vrací návratový kód, z něhož zjistíme, zda nedošlo k chybě. Pokud k chybě nedojde a soubor je uložen na disk, pak `getstore` vrací hodnotu `200`. Pokud není soubor v danén umístění nalezen, vrací hodnotu `404`. Návratové kódy se dají vyhodnotit funkcemi `is_success` a `is_error`.

2.4 Modul File::Path

Perl obsahuje standardní funkci `mkdir`, která nedokáže vytvořit více než jeden vnořený adresář najednou. Proto se nabízí modul `File::Path` umožňující pomocí příkazu `mkpath` vytvořit více než jeden vnořený adresář na jediném řádku. Toho se s výhodou využívá při vytváření složitejších cest obsahujích více dosud neexistujících vnořených podadresářů.

2.5 Modul HTML::TreeBuilder

`TreeBuilder` je modul obsahující metody, které dokážou rozložit strukturu HTML stránky na strom a vybírat z něho informace o HTML uzlech, vybírat text z webové stránky a podobně. Metoda `new_from_content($string)` vytvoří objekt obsahující stromovou strukturu webové stránky uložené v řetezci `$string`. Stejně pracuje metoda `new_from_file($file)`, která si navíc sama zajistí otevření a načtení obsahu souboru, jehož adresa je uložena v proměnné `$file`. V těchto stromech můžeme vyhledávat podle tagů – `find_by_tag_name()`, atributů HTML tagů `find_by_attribute()` a podobně. Výsledky hledání se převádí na text pomocí metody `as_text()`.

2.6 Generování adresy článku a jeho stažení

Nejvhodnější jsou takové webové stránky, jejichž adresa lze generovat v cyklu. V praxi to znamená, že soubory jsou pojmenovány čísly – názvy souborů obsahující články jsou například ve tvaru `http://www.blisty.cz/art/$cislo.html`. Tento tvar mají Britské listy, hodnota `$cislo` se momentálně pohybuje cca od 8000 do 32000 (listopad 2006).

2.7 Zdroje dat

Britské listy – www.blisty.cz

Britské listy patří mezi jednodušší varianty stahování. Mají volně přístupné veškeré vydané články od vzniku v roce 1996. Během let došlo několikrát ke změně

uspořádání článků. Na celý archív Britských listů tedy nelze vystačit s jediným stahovacím programem. Z hlediska počtu článků a jednoduchosti stahování se vyplatí zaobírat se pouze nejnovějším upořádáním archívu, takže stahovací program uvedený na obr. 2 umí stáhnout články od konce roku 2001 po současnost. Články jsou uloženy na `www.blisty.cz` v adresáři `/art` a jsou pojmenovány pouze číslem a příponou `*.html`, např. `www.blisty.cz/art/27898.html`.

```
use LWP::Simple;
use locale;
use File::Path;
use Time::Local;

$od=8800;
$do=31500;
$kam="./down";

for($cislo=$od;$cislo<=$do;$cislo++){
  $status=getstore("http://www.blisty.cz/art/$cislo.html",">$kam"."/"."$cislo.html");
  print STDERR "\n $cislo: Ok" if is_success($status);
  print STDERR "\n $cislo: Error" if is_error($status);
}
print "\n...konec";
```

Obrázek 3: Výpis programu na stahování článků z Britských novin

Na začátku programu se definují použité moduly `LWP::Simple`, `locale`, `File::Path`, definuje se adresář, do kterého se budou ukládat stažené soubory, a zadává se číselný interval článků. V cyklu `for` se generuje adresa článku a jméno souboru na disku. Funkce `getstore` se poté pokusí článek uložit. V proměnné `$status` je informace o úspěsnosti stažení, ta se dále vyhodnocuje funkcemi `is_success a is_error` a vypisuje na obrazovku.

Nevýhodou Britských listů je, že nemají články tříděné podle konkrétnějších témat – politika, historie, kultura, zajímavosti, apod. Články se tak nedají třídit jen v rámci Britských listů, i když rozsah témat, kterými se zabývají, je dost široký. K našemu účelu automatické detekce témat je tak potřeba ještě několik dalších serverů s odlišnou tématikou (sport, počítače, ...) k třídění článků podle témat.

Sport online – www.deniksport.cz

Sport online je internetový magazín deníku Sport a je velice podobný struktuře Britských listů. Články jsou postupně číslovány a jsou rozděleny do různých témat. Rozdíly mezi tématy nejsou moc výrazné, takže klasifikace článků podle témat v rámci tohoto zpravodajství se zdá velice obtížná. Stahovací program je téměř totožný s programem na stahování Britských listů, liší se pouze odlišnou webovou adresou – např. `www.deniksport.cz/Clanek484723.htm`

PC World – www.pcworld.cz

PC World zveřejňuje část svých článků z papírového vydání na svém serveru. Články jsou sice rozdělené do témat, ale protože zveřejných článků není mnoho, jsou staženy do jednoho společného adresáře. Struktura serveru PC World je poněkud složitější. Adresa `www.pcworld.cz/pcw.nsf/html/archiv.htm` obsahuje archiv starších vydání. Tyto odkazy obsahují adresy na články. Je tedy potřeba prozkoumat každé číslo zvlášť a ze zdrojového kódu stránky vyjmout přímé adresy na články a poté je stáhnout. Adresa článku se skládá z posloupnosti různých znaků a adresy článků tedy nelze jednoduše generovat v cyklu. Ukázka adresy článku: `www.pcworld.cz/pcw.nsf/c7a9e9cf79b1ab4bc1256fa1004c1a12/ff3e08dd4f8a8e42c12570c3003d78d7?OpenDocument`.

Atlas zprávy – zpravy.atlas.cz

Zpravodajství na serveru Atlas patří mezi typické představitele internetového zpravodajství. Články jsou opět postupně číslovány a navíc mají označené téma v záhlaví. HTML soubory obsahující články nejprve stáhneme do určeného adresáře. Do témat se články dělí později při předzpracování textu.

Aktuálně – www.aktualne.cz

Z pětice zpravodajských serverů je stahování z tohoto nejkomplikovanější. Články jsou rozděleny podle témat a názvy souborů jsou opět postupně číslované. Adresa článku vypadá přibližně následovně:

aktualne.centrum.cz/clanek.phtml?id=322106. Při stahování se vyskytlo několik problémů. Ne pod všemi čísly je uložen soubor s článkem. To znamená, že například ze sta vygenerovaných adres se podaří úspěšně stáhnout jen několik málo článků (odhadem 3÷5). Při stahování je třeba využít balík `LWP::UserAgent`, který nastaví správné kódování, jinak by texty neobsahovaly diakritiku. Protože tento server nevrací chybové hlášení při neexistenci stránky (kód 200 – OK, kód 404 – ErrorDocument, atp.), ale místo článku se pouze zobrazí hláška – Soubor neexistuje! – je při stahování zároveň zjišťováno, jedná-li se skutečně o soubor obsahující článek. Je-li tomu tak, je soubor uložen.

Další komlikace se vyskytne při rozložení struktury stránky, kdy LWP špatně rozloží stránku a program ukončí vybírání textu u prvního vnořeného tagu `<DIV>`, ve kterém se skrývá ilustrační obrázek. Do textového souboru je tak uložen jen první odstavec článku. Aby bylo vše v pořadku je třeba z HTML souboru vybrat část `id=hlavni_obsah` namísto `class=clanek`, což je vlastně nejbližší nadřazená vrstva. Po této úpravě program správně vybírá veškerý text obsažený pod tagy <P>.

2.8 Předzpracování článků

I v tomto případě patří Britské listy mezi jednodušší případy. Články s čistým textem se ukládají podle data vydání do podadresářů. Datum se zjišťuje příkazy:

```
$dotaz($tree->find_by_attribute('class','datum'));
$datum=$dotaz->as_text( );
```

Poté se upraví jeho formát a příkazem `mkpath()` se vytvoří příslušný podadresář.

Nejdůležitější částí je výběr samotného textu článku. Text článku má atribut `class='telo'`. Tento atribut se může vyskytovat v článku vícekrát – proto je hledán v cyklu funkcí `find_by_attributte()`.

```
$dotaz($tree->find_by_attribute('class','telo'));
$obsah=$dotaz->as_text( );
print OUT $obsah;
```

2.9 Použitelnost dat

Jako nejvhodnější pro potřeby klasifikace jsou články ze zpravodajství ze serveru Atlas. Britské listy, Sport online a PC World je možno třídit jen v rámci různého tématického zaměření serveru a jako nevhodné se ukázaly články serveru Aktuálně.

3 Program WEKA

WEKA[4] je volně dostupný program vyvinutý na univerzitě Waikato na Novém Zélandu. WEKA dokáže v aplikaci `Explorer` natrénovat klasifikátory pomocí různých algoritmů – `Classify`, umí shlukovou analýzu – `Cluster`, hledat příznaky – `Select attributes` a vizualizovat data – `Visualize`.

3.1 Tvar vstupních dat

WEKA pracuje se soubory těchto typů:

- ARFF (Attribute-Relation File Format) *.arff
- CSV (Comma Separated Values) *.csv
- C4.5 formát *.data, *.names
- BSI (Binary Serialized Instance) *.bsi.
- a další

Nejvhodnější formáty jsou ARFF a CSV. Z mnoha podporovaných formátů vstupních souborů programu WEKA jsou nejpoužívanější a dají se snadno vytvořit za pomoci textového nebo tabulkového editoru. Se soubory typu CSV umí pracovat tabulkový editor Microsoft Excel. Formát CSV souboru je následující: do prvního řádku se za sebou vypisují názvy atributů a do dalších řádků pod nimi hodnoty z vektorů. Řádek představuje právě jeden příznakový vektor. Hodnoty v řádcích jsou od sebe odděleny čárkou. Soubor je samozřejmě nutné uložit jako typ CSV s příponou *.csv.

Pro tvorbu souborů ve formátu ARFF lze využít jakéhokoliv textového editoru, ve Windows např. WordPad nebo NotePad. Soubor ARFF se skládá ze dvou částí – z hlavičky a datové části. Hlavička má následující strukturu: první řádek začíná návěštím `@RELATION` a za ním je uvedený název nebo označení souboru. Na dalších řádcích jsou uvedeny atributy. Za návěštím `@ATTRIBUTE` se zapisuje název atributu

[4] http://www.cs.waikato.ac.nz/ ml/weka

a jeho typ. Typů nabízí WEKA několik. Z číselných typů to jsou `INTEGER` a `REAL`, souhrnně je lze zapsat jako `NUMERIC`, dále typ řetězec – `STRING`, datum – `DATE` a typ `NOMINAL`, což je uživatelsky definovaný typ, který může obsahovat jakékoliv hodnoty. Na rozdíl od ostatních typů, které se zapisují přímo do hlavičky ARFF souboru (hlavička ARFF souboru přímo obsahuje slova `REAL`, `INTEGER` apod.), se typ `NOMINAL` zapisuje do složených závorek a do nich se vypíší všechny možné hodnoty, jichž může atribut nabývat. Typ `REAL` se používá pro desetinná čísla při frekvenční nebo TF-IDF reprezentaci. Na hodnoty atributů při binární reprezentaci je použit typ `NOMINAL` zapsaný jako `{0,1}`. Tento typ nese informaci pouze o tom, zda-li dokument obsahuje daný příznak či nikoliv. Na binární reprezentaci by šel použít i typ `INTEGER` a `REAL`, ale některé algoritmy (např. ID3) přímo vyžadují typ `NOMINAL`. Po výpisu atributů následuje datová část začínající návěštím `@DATA`. Za tímto návěštím jsou data z vektoru vypsána do řádku za sebou a jsou oddělena čárkou, přičemž jako poslední hodnota vektoru je zapisována příslušnost daného dokumentu ke své třídě. Vznikne tak matice příznaků, kdy v řádku je popsaný dokument a ve sloupcích jsou informace o vybraných příznacích. Ukázka ARFF souboru je na obrázku 4.

Soubory *.data, *.names, *.bsi jsou uvedeny jen pro úplnost. Tyto typy souborů nejsou často používány, a ani dokumentace k programu WEKA je podrobně nepopisuje [11].

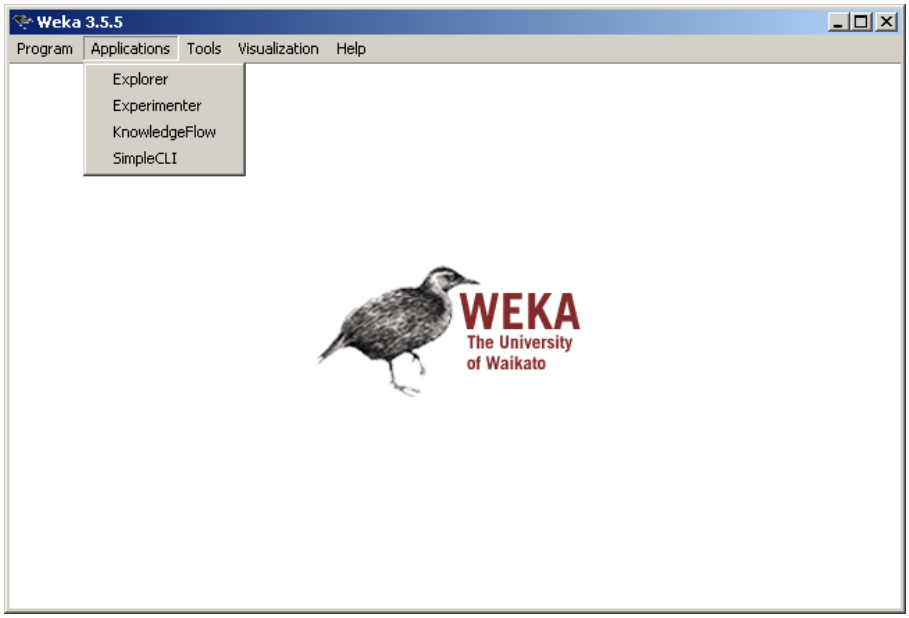
3.2 Instalace a spuštění programu WEKA

Instalační soubory programu WEKA jsou volně dostupné ke stažení. Po instalaci v operačního systému Windows se WEKA s grafickým rozhraním spouští z nabídky `Start→Programy`. Úvodní obrazovka programu WEKA je na obrázku 5. V menu `Applications` lze vybrat z několika podprogramů. Pro potřeby automatické detekce se využívá aplikace `Explorer`. Stejná aplikace se otevře při poklepání na soubor ARFF.

Je-li potřeba zpracovat opravdu velké množství dat, kdy ARFF soubor obsahuje rozměrnější příznakovou matici, pak WEKA nevystačí se standardně nastavenou velikostí paměti. Následkem nedostatečné velikosti paměti se WEKA stane nesta-

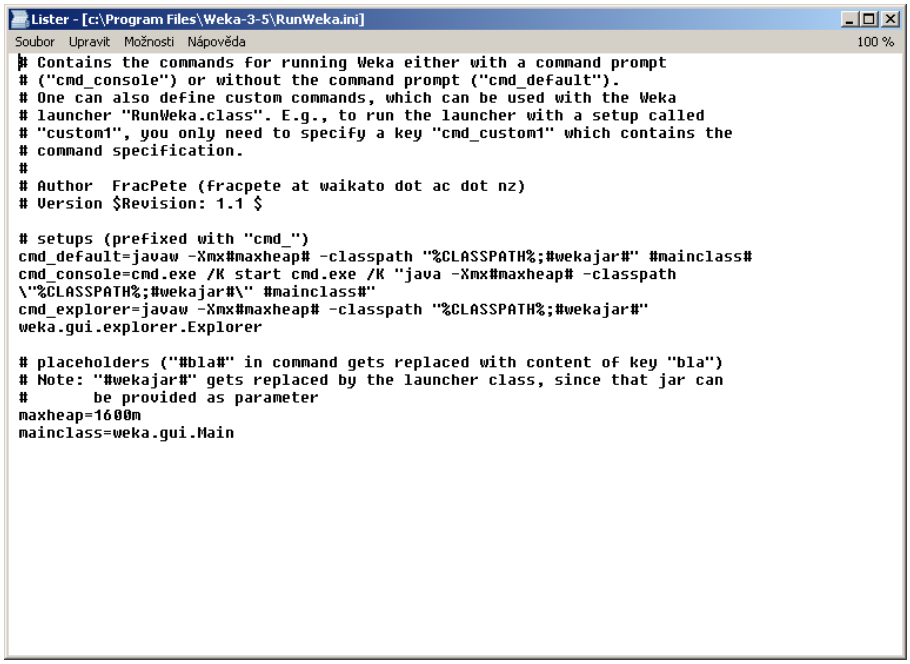
```
@RELATION název souboru
@ATTRIBUTE s1 NUMERIC %možno použít i INTEGER, REAL
@ATTRIBUTE třída {třída_1,třída_2} %typ NOMINAL
@DATA %matice příznaků
hodnota(s1),hodnota(s2),hodnota(s3),hodnota(s4),třída %soubor1
hodnota(s1),hodnota(s2),hodnota(s3),hodnota(s4),třída %soubor2
hodnota(s1),hodnota(s2),hodnota(s3),hodnota(s4),třída %...
hodnota(s1),hodnota(s2),hodnota(s3),hodnota(s4),třída %...
```

Obrázek 4: Ukázka struktury ARFF souboru



Obrázek 5: WEKA – úvodní obrazovka

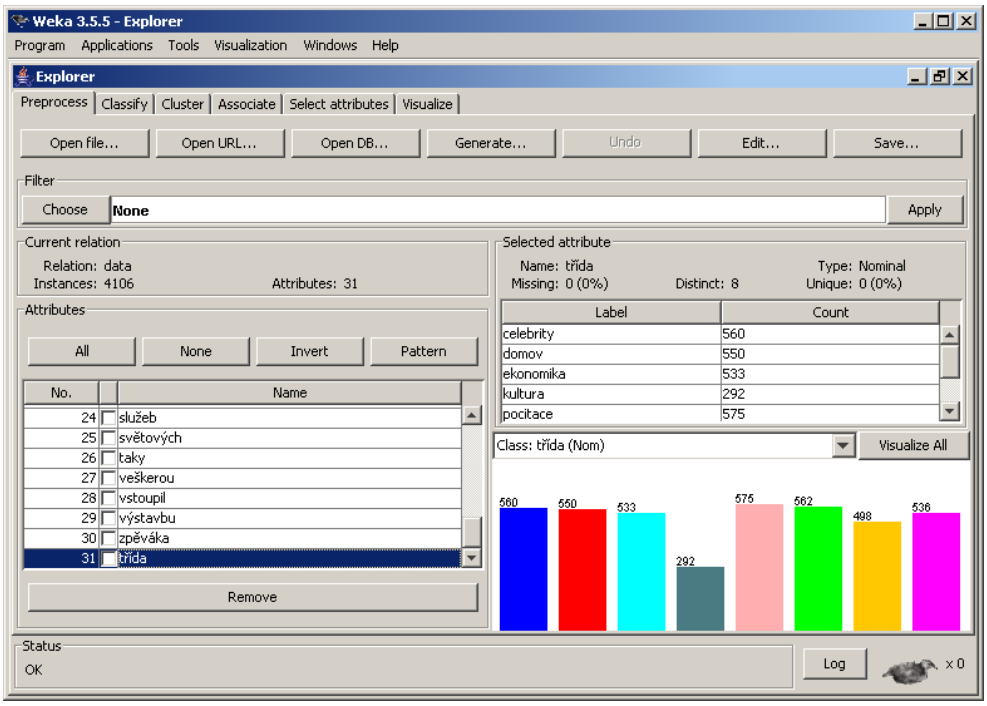
bilní. V kořenovém adresáři programu WEKA – například `C:/Program Files/weka-3-5` se musí editovat soubor `RunWeka.ini` a změnit hodnotu proměnné `maxheap` z implicitně nastavených 128 MB (`maxheap=128m`) paměti na vyšší hodnotu, např. 1024 MB (`maxheap=1024m`). Ukázka souboru `RunWeka.ini` je na obrázku 6. Jako největší použitelná velikost matice ARFF souboru ukázala velikost přibližně 9000x3000. Při ještě větší dimenzi matice nepomůže ani nastavení hodnoty 1600 MB (`maxheap=1600m`), která byla nastavena pro všechny testy.



Obrázek 6: RunWeka.ini

3.3 Načtení dat

Data se v prostředí `Explorer` načítají tlačítkem `Open file` viz obrázek 7. Pokud vše proběhne v pořádku, všechny záložky se stanou aktivními. V boxu `Attributes` vlevo dole vybereme položku, která značí třídu a poté se v grafu vpravo dole zobrazí statistika příznakové matice.



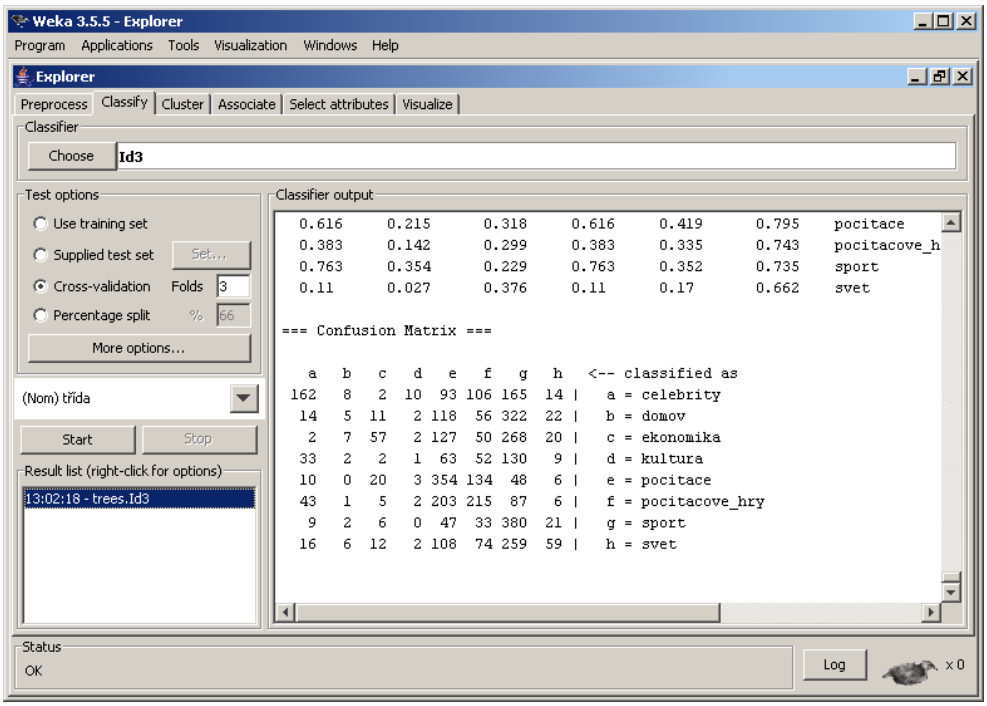
Obrázek 7: Načtení dat – zobrazení počtů dokumentů v třídách

3.4 Trénování klasifikátorů (Classify)

Na kartě `Classify` se trénují klasifikátory (obrázek 8). Po klepnutí na tlačítko `Choose` v horní části lze vybrat různé typy klasifikátorů, jejichž parametry se nastavují po klepnutí levým talčítkem myši na jméno algoritmu. Následně se otevře okno, ve kterém je stručný popis algoritmu a možnosti jeho nastavení. Nyní lze spustit test. WEKA nabízí 4 druhy testů:

- `Use training set` – test na tréninkové sadě dat.
- `Supplied test set` – test na testovací sadě dat určené uživatelem.
- `Cross validation` – n-násobná křížová validace, data jsou rozdělena do n-množin; každá množina je využita jako trénovací, zbytek je využit jako testovací data.
- `Percentage split` – procentní rozdělení.

Tlačítko `More options...` umožňuje podrobnější nastavení testu. Tlačítkem `Start` se spouští test (viz obrázek 8), výsledky se zobrazí v okně umístěném napravo. Pokud je třeba výsledky uložit do textového souboru, využívá se standardního postupu označení a kopírování textu s využitím schránky (clipboardu).



Obrázek 8: WEKA – výsledek hledání klasifikátoru

3.5 Nalezení příznaků (Select attributes)

V záložce `Select attributes` se vybírá algoritmus, metoda a typ rozdělení trénovací množiny – použít se dá celá množina anebo n-násobná křížová validace.

3.6 Shluková analýza (Cluster)

Data jsou rozdělena do vlastních tříd a výsledek je porovnán s původním rozdělením. Takto si lze ověřit, zda jsou data rozdělena do dostatečně odlišných množin. U vhodně vybraných dat bude rozdělení do nových tříd takřka shodné s původním

rozdělením. V ideálním případě by měly třídy určené shlukovou analýzou obsahovat stejná data jako původní třídy. Při analýze byl využit algoritmus `EM`, zaškrtnutá volba `Classes to clusters evaluation - (Nom) class`.

3.7 Testované algoritmy

Pro hledání vhodného klasifikátoru byly použity následující algoritmy:

NaiveBayes – Naivní Bayesův klasifikátor, zástupce pravděpodobnostních algoritmů.

ID3 – Iteral Dichotomizer 3, zástupce rozhodovacích stromů.

J48 – další zástupce rozhodovacích stromů, vychází z algoritmu ID3.

IB1 – Instant Based algorithm, zástupce algoritmu nejbližšího souseda.

SMO – Sequential Minimal Optimization algorithm, zástupce Support Vector Machine.

4 Praktické výsledky

4.1 Výběr dat

Ze zpravodajství na serveru Atlas bylo vybráno 5 nejpočetnějších rubrik. K nim byla přidána rubrika počítače obsahující články z časopisu PC WORLD. Data jsou tříděna do 6 tříd. Z těchto rubrik je v každé třídě v trénovací i testovací sadě dat náhodně vybráno přibližně 500 článků. Přesné počty článků a názvy vybraných rubrik jsou uvedeny v tabulce 1. Žádný z článků není použit ve více rubrikách a není obsažen v obou sadách dat zároveň.

	počet článků v trénovací sadě	**počet článků v testovací sadě**	**zdroj**
celebrity	584	522	zpravy.atlas.cz
z domova	543	543	zpravy.atlas.cz
ekonomika	485	496	zpravy.atlas.cz
počítače	500	500	www.pcworld.cz
sport	498	590	zpravy.atlas.cz
ze světa	536	580	zpravy.atlas.cz
celkem	3146	3231	

Tabulka 1: Rozdělení a počty článků v trénovací a testovací sadě dat

4.2 Hledání příznaků s využitím všech tříd

4.2.1 Binární reprezentace příznakového vektoru

K vytvoření globálního slovníku s binární reprezentací slouží program `bin.pl` (viz oddíl A.2, strana 80). V programu `bin.pl` lze nastavit minimální a maximální počet dokumentů, v nichž se slovo může vyskytovat. U binární reprezentace si lze vybrat dva typy popisu dat pro ARFF soubor. Je možné volit mezi typy NOMINAL s hodnotami 0,1 a nebo REAL, který nabývá stejných hodnot. Program automaticky odstraňuje jednopísmenná slova, slova obsahující numerické

znaky, znaky interpunkce a vylučuje slova obsažená ve stoplistu (viz 1.2, strana 17).

Nejprve byl vytvořen globální slovník zastupující slova obsažená v článcích všech 6 tříd. U každé položky ve slovníku je uveden počet dokumentů, ve kterých je slovo obsaženo. Při prvním testu byla snaha využít pro hledání příznaků co největší globální slovník. Byla odstraněna taková slova, jež se celkově nevyskytovala alespoň v 10 dokumentech. Jelikož byl ARFF soubor s daty pro systém WEKA příliš velký, byly podmínky upraveny. Použita byla slova obsažená alespoň v 15 dokumentech. Navíc byla přidána podmínka výskytu maximálně v 500 dokumentech. Výsledný soubor obsahoval 9220 slov a takto připravený soubor byl použit k hledání příznaků.

WEKA našla z 9220 slov 109 vhodných příznaků. Bylo použito standartní nastavení programu pro hledání příznaků (algoritmus CfsSubsetEval, metoda BestFirst). Na obrázku 9 je část výpisu, ze kterého je možno zjistit použitý algoritmus hledání včetně parametrů nastavení, počet vstupních příznaků, počet dokumentů apod. V posledním odstavci jsou vypsány pozice vybraných příznaků v globálním slovníku, za dvojtečkou je celkový počet vybraných příznaků.

```
=== Run information ===

Evaluator:    weka.attributeSelection.CfsSubsetEval
Search:       weka.attributeSelection.BestFirst -D 1 -N 5
Relation:     data
Instances:    3146
Attributes:   9221
              [list of attributes omitted]
Evaluation mode:    evaluate on all training data


=== Attribute Selection on all input data ===

Search Method:
        Best first.
        Start set: no attributes
        Search direction: forward
        Stale search after 5 node expansions
        Total number of subsets evaluated: 1026428
        Merit of best subset found:    0.416

Attribute Subset Evaluator (supervised, Class (nominal): 9221 #třída#):
        CFS Subset Evaluator
        Including locally predictive attributes

Selected attributes:
26,116,118,150,177,235,571,626,737,752,890,908,918,937,1001,1157,1215,1447,1522,1527,1750,1766,1770,1780,2084,2092,2124,2202,
2267,2278,2300,2340,2362,2367,2605,2841,3034,3149,3274,3449,3463,3535,3710,3717,3763,3774,3794,3827,3999,4143,4145,4260,4300,
4393,4429,4527,4615,4668,4729,4816,4925,4971,4993,5021,5033,5066,5068,5092,5150,5173,5231,5709,5850,5865,5939,5969,6078,6088,
6097,6136,6270,6302,6446,6589,6602,6758,6779,6897,7065,7211,7300,7519,7530,7682,7690,7806,7923,8019,8142,8188,8260,8265,8390,
8435,8835,8860,9010,9065,9091 : 109
```

Obrázek 9: Výpis hledání příznaků

Pod tímto odstavcem následuje výpis hledání příznaků a následují pozice vybraných příznaků. Příznaky se ručně zkopírují do textového souboru. Program

`bin_test.pl` (viz A.3, strana 93) z těchto příznaků vytvoří ARFF soubor, se kterým byly testovány různé klasifikační algoritmy. Klasifikátor se testuje na testovací množině dat, která je popsána pomocí příznaků nalezených na trénovací množině.

Úspěšnost klasifikátoru byla u dat popsaných jako REAL testována pomocí algoritmů NaiveBayes, J48, IB1 a SMO. U příznaků popsaných jako typ NOMINAL byl navíc vyzkoušen algoritmus ID3. Testy byly nastaveny jako 3-násobná křížová validace a s rozdělením 66 %. Pro zajímavost byl vyzkoušen i test na trénovací sadě dat.

Výsledky s popisem REAL jsou v tabulce 2. Nepatrně lepších výsledků dosahují algoritmy NaiveBayes a SMO. Ale i tak je jejich úpěšnost velice špatná. Výsledky na trénovací množině nejsou překvapivě lepší.

algoritmus	**NB [%]**	**J48 [%]**	**IB1 [%]**	**SMO [%]**
3-fold	44,8	35,9	35,5	44,8
split 66 %	45,6	35,9	36,0	44,3
training set	46,2	43,0	45,5	48,9

Tabulka 2: Binární reprezentace, popis REAL, počet příznaků v intervalu <15;500>

Výsledky s popisem NOMINAL jsou v tabulce 3. Dosažená úspěšnost je opět velice špatná. Dokonce výsledky algoritmu ID3 jsou horší než u J48. Algoritmus ID3 vyžaduje popis typem NOMINAL, proto byly očekávány lepší výsledky než s J48.

algoritmus	**NB [%]**	**J48 [%]**	**ID3 [%]**	**IB1 [%]**	**SMO [%]**
3-fold	46,5	44,7	42,2	35,5	44,8
split 66 %	47,2	43,7	42,2	36,0	44,3
training set	49,3	50,1	56,9	45,5	48,9

Tabulka 3: Binární reprezentace, popis NOMINAL, počet příznaků v intervalu <15;500>

Výsledky se zvoleným výskytem slov v dokumentech s počtem příznaků v intervalu <15;500> nejsou příliš uspokojivé. Též velikost ARFF souboru a časově

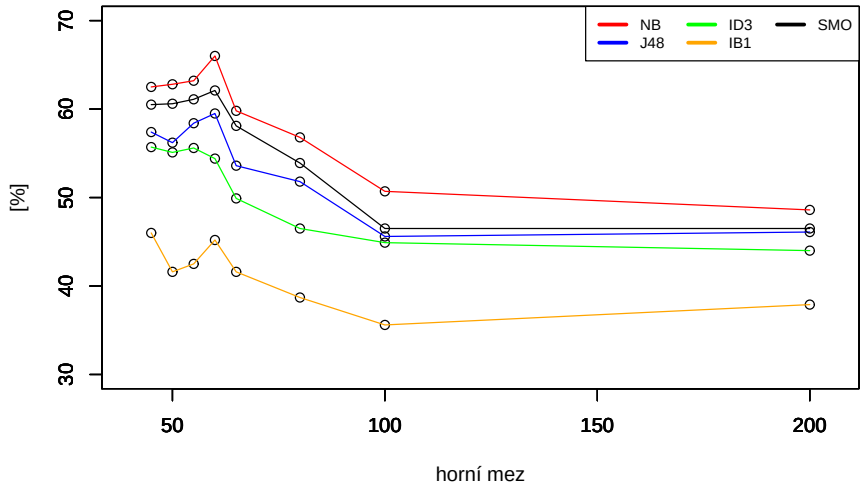
náročné hledání příznaků jsou nepříjemné. Došlo tedy k redukci počtu výskytů příznaků. Nejprve byla zkoušena horní mez intervalu. Díky lepším výsledkům při popisu dat typem NOMINAL byl použit tento popis. Výsledky jsou uvedeny v tabulce 4. Jako spodní mez intervalu byla zvolena hodnota 25. Nejlepších výsledků bylo dosaženo s počtem příznaků v intervalu <25;60>.

interval	**velikost slovníku**	**počet příznaků**
25-45	2746	353
25-50	3060	341
25-55	3320	341
25-60	3532	324
25-65	3719	280
25-80	4167	212
25-100	4508	138
25-200	5230	111
50-100	1508	115
50-200	2230	111

Tabulka 4: Hledání horní meze počtu příznaků, binární reprezentace, popis NOMINAL

Při hledání vhodné hodnoty spodního meze počtu příznaku byl byl použit nejlepší výsledek z předchozího testu hledání horní meze intervalu – viz tabulka 4. Použita tedy byla hodnota 60. Experimentálně byly vyzkoušeny i jiné intervaly. Pokusy se spodní mezí nepřinesly žádné zlepšení, výsledky jsou uvedeny v tabulce 5.

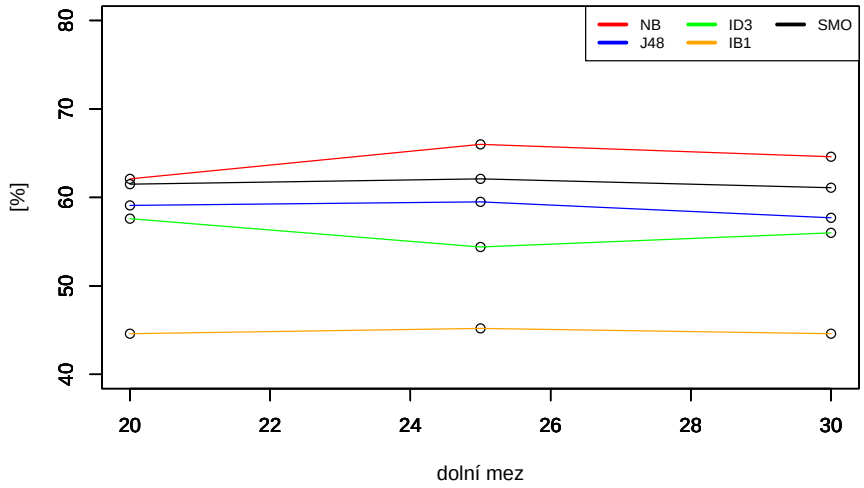
Pro většinu algoritmů vychází nejlépe interval se spodní mezí 25. Rozdíly mezi různými hodnotami jsou pouze v jednotkách procent, nelze přesně stanovit nejlepší interval. Jako užitečné se jeví nejít se spodní mezí pod hodnotu 20, ale spíše volit hodnotu kolem 30. Globální slovník se pak zmenší přibližně na polovinu, zrychlí se zpracovaní a přitom není znát vliv menšího počtu slov na úspěšnost klasifikace.



Obrázek 10: Hledání horní meze, binární reprezentace, popis NOMINAL

interval	**velikost slovníku**	**počet příznaků**	**NB** [%]	**J48** [%]	**ID3** [%]	**IB1** [%]	**SMO** [%]
20-45	4153	382	62,3	56,6	54,1	43,4	59,0
25-45	2746	353	62,5	57,4	55,7	46,0	60,5
30-45	1725	287	60,1	54,2	53,0	42,8	59,2
35-45	991	224	56,8	53,7	51,7	40,1	56,0
20-60	4940	356	62,1	59,1	57,6	44,6	61,5
25-60	3532	324	66,0	59,5	54,4	45,2	62,1
30-60	2512	279	64,6	57,7	56,0	44,6	61,1

Tabulka 5: Hledání spodní meze počtu výskytů příznaků, binární reprezentace, popis NOMINAL



Obrázek 11: Hledání spodní meze, binární reprezentace, popis NOMINAL

Úprava na základní tvar

V dalším kroku byla vyzkoušena úprava globálního slovníku na základní tvar. Byl použit Hajičův analyzátor. Vstupem do analyzátoru byl globální slovník, který obsahoval všechna slova bez omezení počtu výskytů. Pokud k danému slovu existuje základní tvar, je původní slovo nahrazeno základním tvarem. Může existovat i více než jeden základní tvar z důvodu existence více významů slova. Zdejší úprava do základního tvaru není zcela přesná, protože nebere v úvahu více možných významů slova. Z upraveného slovníku byly vybrány příznaky s počtem výskytů v intervalu <25;60>, výsledky jsou v tabulce 6. Počet slov ve slovníku klesnul z 324 na 186 slov.

Pokus s převodem do základního tvaru byl neúspěšný. Potvrdil se sice předpoklad snížení počtu příznaků – zde dokonce o jednu třetinu – ale výsledek je jen těžko použitelný. Dosažené výsledky jsou horší než ty původní, úspěšnost klesla o 15-20 %.

	velikost slovníku	počet příznaků	NB [%]	J48 [%]	ID3 [%]	IB1 [%]	SMO [%]
původní	3532	324	66,0	59,5	54,4	45,2	62,1
základní tvar	2363	186	41,9	40,5	39,3	28,9	40,9

Tabulka 6: Úprava na základní tvar, binární reprezentace, popis NOMINAL, počet výskytu příznaků v intervalu <25;60>

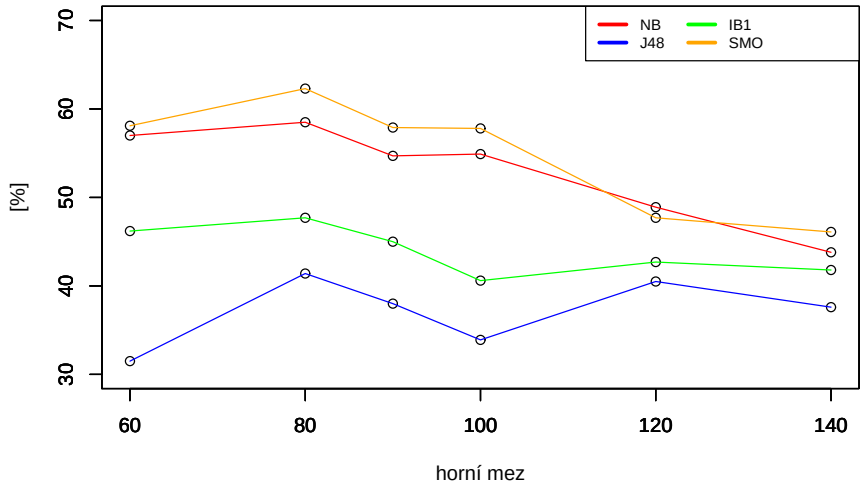
4.2.2 Frekvenční reprezentace příznakového vektoru

K převodu dat na frekvenční reprezentaci slouží program `tf.pl` (viz A.2, strana 84). Program umožňuje nastavení minimálního a maximálního počtu dokumentů, v nichž se slovo může vyskytovat. Oproti binární reprezentaci zde nelze použít popis dat typem NOMINAL, ale pouze typem REAL. Stejně jako již výše popsaný program automaticky odstraňuje jednopísmenná slova, slova obsahující numerické znaky, znaky interpunkce a vylučuje slova vložená ve stoplistu.

Dosažené výsledky jsou lepší než s binární reprezentací, algoritmy NaiveBayes a SMO dosahují úpěšnosti kolem 60 %. I přesto nejsou výsledky ani teď příliš uspokojivé.

interval	velikost slovníku	počet příznaků
25-60	4143	427
25-80	4911	417
25-90	5161	345
25-100	5364	318
25-120	5671	223
25-140	5901	203

Tabulka 7: Hledání horní meze počtu výskytu příznaků, frekvenční reprezentace



Obrázek 12: Hledání horní meze počtu výskytu příznaků, frekvenční reprezentace

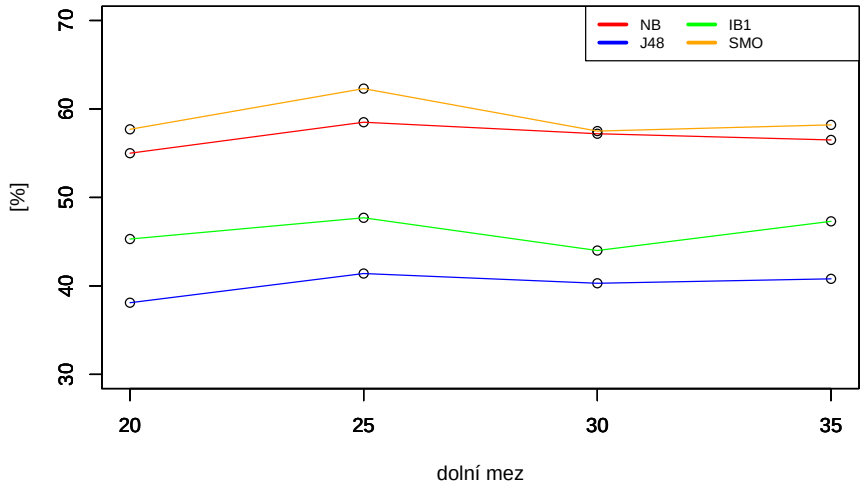
Při hledání spodní meze byl opět použit nejlepší výsledek předchozího testu a i tentokrát se ukázala jako nejlepší hodnota 25.

interval	**velikost slovníku**	**počet příznaků**
20-80	6476	427
25-80	4911	417
30-80	3782	401
35-80	2968	372

Tabulka 8: Hledání spodní meze počtu výskytů příznaků, frekvenční reprezentace

4.2.3 TF-IDF reprezentace příznakového vektoru

Program na převod dat do TF-IDF reprezentace `tfidf.pl` (viz A.2, strana 88) funguje stejně jako u předchozích dvou reprezentací. Od programu `tf.pl` (viz A.2, strana 84) se liší jen jiným způsobem ohodnocení příznaků. Data jsou popsána typem REAL.

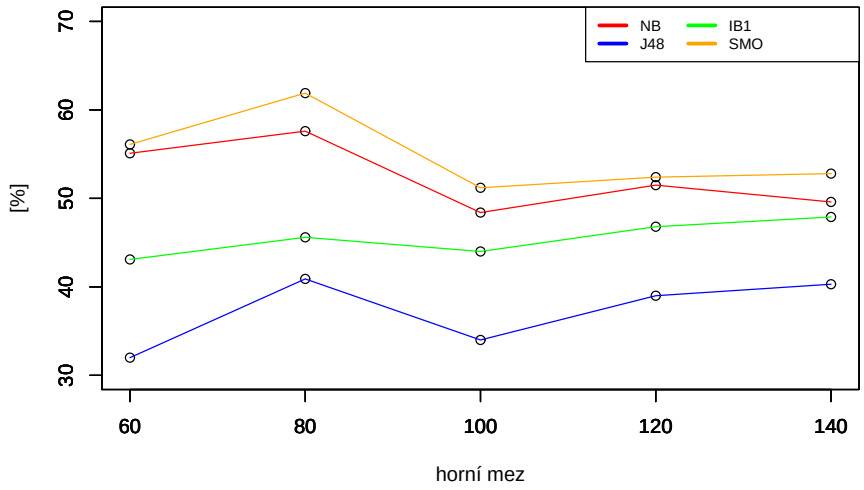


Obrázek 13: Hledání spodní meze počtu výskytů příznaků, frekvenční reprezentace

Výsledky jsou opět podobné s frekvenční reprezentací. Úspěšnost dosahuje maximálně 60 % u algoritmů NaiveBayes a SMO. NaiveBayes má sice nepatrně horší výsledky oproti SMO, ale pokud jde o rychlost, pak NaiveBayes nemá konkurenci. Oproti ostatním algoritmům je nezávisle na typu popisu i reprezentaci mnohonásobně rychlejší.

interval	**velikost slovníku**	**počet příznaků**
25-60	4144	434
25-80	4912	416
25-100	5366	302
25-120	5674	214
25-140	5906	198

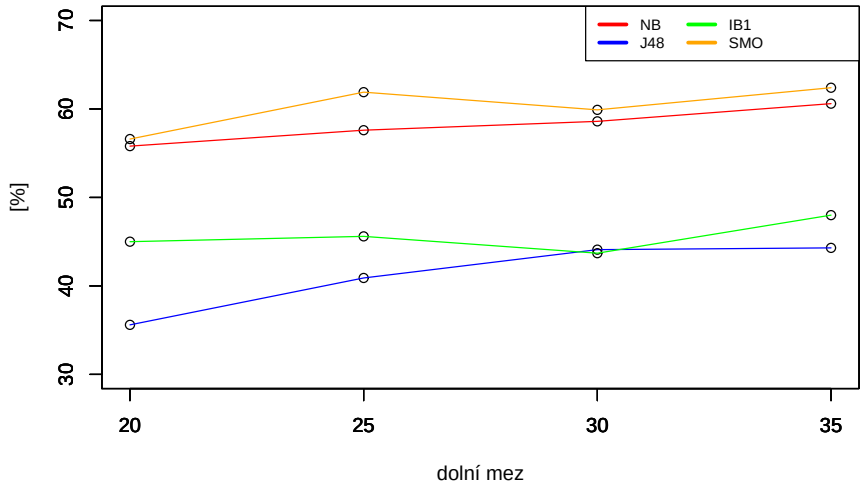
Tabulka 9: Hledání horní meze, TF-IDF reprezentace



Obrázek 14: Hledání horní meze, TF-IDF reprezentace

interval	**velikost slovníku**	**počet příznaků**
20-80	6477	424
25-80	4912	416
30-80	3783	408
35-80	2969	379

Tabulka 10: Hledání spodní meze, TF-IDF reprezentace



Obrázek 15: Hledání spodní meze, TF-IDF reprezentace

4.3 Hledání příznaků mezi dvojicemi tříd

4.3.1 Binární reprezentace příznakového vektoru

Jelikož hledání příznaků s využitím všech témat najednou nepřineslo dobré výsledky, byla data pokusně roztříděna do tří skupin po dvou tématech viz tabulka 11. Samostatně pro každou skupinu bylo provedeno hledání příznaků. Nalezené příznaky byly zkopírovány do jednoho souboru a byla odstraněna duplicitní slova. Takto připravené příznaky byly použity pro příznakový popis. Oproti předchozímu hledání se zvýšil počet příznaků pro příznakový vektor, ale hledání příznaků se znatelně zrychlilo. Postup byl stejný jako výše popsaný, akorát bylo potřeba většinu operací provést třikrát.

Nejprve bylo provedeno hledání pro výchozí interval počtu výskytů příznaků <15;500> – viz tabulka 12 a 13, aby bylo možno srovnat s předchozími výsledky. Vyzkoušen byl popis REAL i NOMINAL. NOMINAL opět vykazoval výsledky o pár procent lepší oproti REAL. Úspěšnost u některých klasifikátorů dosahuje až 75 %. Výsledky u algoritmu ID3 opět zaostávaly za J48. U testů na trénovací sadě se ukázaly výrazně lepší výsledky podle očekávání, až 98% úspěšnost.

skupina 1	celebrity
	počítače
skupina 2	z domova
	ze světa
skupina 3	ekonomika
	sport

Tabulka 11: Tabulka rozdělení témat

test options	NB [%]	J48 [%]	IB1 [%]	SMO [%]
3-fold	72,1	60,4	43,7	72,9
split 66%	73,2	63,1	42,8	72,1
training set	74,2	81,7	98,3	88,1

Tabulka 12: Dvojice tříd, binární reprezentace, popis REAL, interval <15;500>

test options	NB [%]	J48 [%]	ID3 [%]	IB1 [%]	SMO [%]
3-fold	74,1	64,0	52,8	43,7	72,9
split 66%	75,3	65,3	53,4	42,8	72,1
training set	78,4	83,1	98,7	98,3	88,1

Tabulka 13: Dvojice tříd, binární reprezentace, popis NOMINAL, interval <15;500>

Díky rozdělení do tří skupin a z toho vyplývajícího zmenšení matice příznaků bylo možné provést hledání i pro interval <8;∞). Výsledky jsou uvedeny v tabulkách 14 a 15. Bohužel jsou výsledky horší oproti intervalu <15;500>. Na testech na intervalu <8;∞> se ukazuje, že zahrnutí co největšího počtu slov při hledání příznaků nemá na úspěšnost klasifikátoru dobrý vliv. Je třeba jít cestou zúžení intervalu při hledání optimálního řešení.

test options	**NB [%]**	**J48 [%]**	**IB1 [%]**	**SMO [%]**
3-fold	64,0	53,7	41,9	64,7
split 66%	64,7	56,4	43,8	64,8
training set	65,7	73,0	91,8	78,2

Tabulka 14: Dvojice tříd, binární reprezentace, popis REAL, interval <8;∞)

test options	**NB [%]**	**J48 [%]**	**ID3 [%]**	**IB1 [%]**	**SMO [%]**
3-fold	66,7	59,8	48,8	41,9	64,7
split 66%	66,5	59,1	46,8	43,8	64,8
training set	71,3	76,5	93,8	91,8	78,2

Tabulka 15: Dvojice tříd, binární reprezentace, popis NOMINAL, interval <8;∞)

Při postupném zužování intervalu se úspěsnost vyšplhala až k 80 %. Výsledky hledání počtu výskytů příznaků jsou v tabulce 16 – horní mez – a v tabulce 17 – spodní mez.

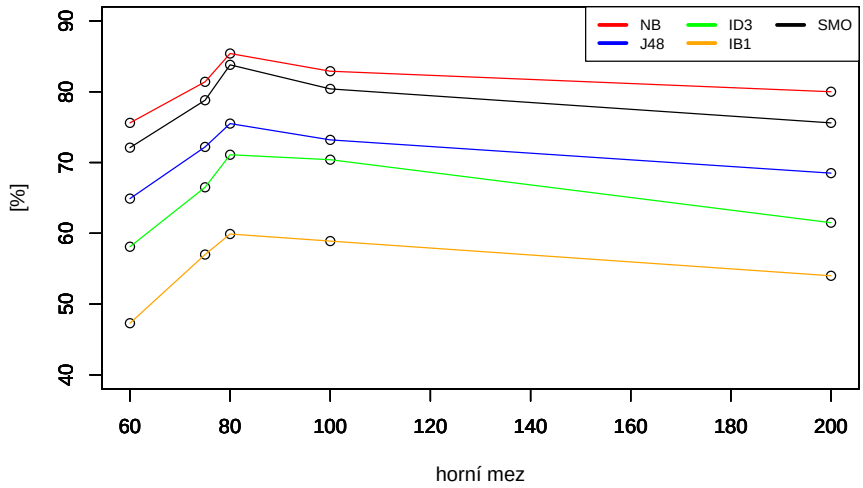
Na intervalu <25;80> byl proveden test s úpravou do základního tvaru. Výsledky jsou mnohem horší, zřejmě v důsledku nerespektování více významů slova.

Pro kontrolu výsledků hledání příznaků na dvojicích témat byly vyzkoušeny odlišně uspořádané skupiny témat. Dokumenty v rubrikách jsou stejné. V tabulce 19 je rozdělení témat, původní rozdělení je označeno jako trénovací sada 1, nové jako trénovací sada 2.

Jak je vidět na výsledcích v tabulkách 20 a 21 – intervaly ani úspěšnost klasifikátorů se příliš neliší. Na rozdělení zřejmě příliš nezáleží, WEKA se při hledání příznaků může věnovat jen rozdílům mezi dvěma třídami, které jsou pak dokonaleji oddělené. Proto bylo dosaženo mnohem vyšší úspěšnosti.

interval	počet příznaků	NB [%]	J48 [%]	ID3 [%]	IB1 [%]	SMO [%]
15-500	331	75,3	56,4	53,4	43,7	72,9
25-60	370	75,6	64,9	58,1	47,3	72,1
25-75	374	81,4	72,2	66,5	57,0	78,8
25-80	368	85,4	75,5	71,1	59,9	83,8
25-85	356	76,8	64,4	59,1	45,5	72,4
25-100	353	82,9	73,2	70,4	58,9	80,4
25-200	303	80,0	68,5	61,5	54,0	75,6
50-100	249	82,1	70,7	68,9	59,1	79,8
50-200	246	77,7	66,3	61,0	49,3	73,7

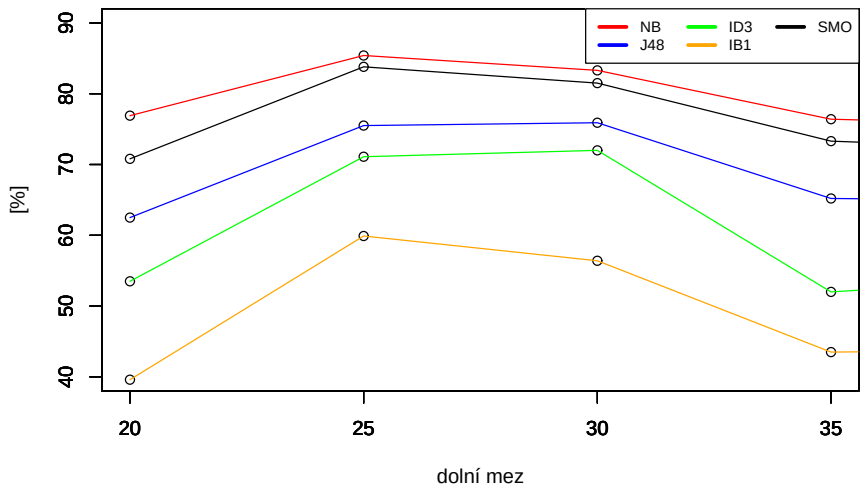
Tabulka 16: Hledání horní meze, dvojice tříd, binární reprezentace, popis NOMINAL, horní mez



Obrázek 16: Hledání horní meze, dvojice tříd, binární reprezentace, popis NOMINAL, horní mez

interval	počet příznaků	NB [%]	J48 [%]	ID3 [%]	IB1 [%]	SMO [%]
15-500	331	75,3	56,4	53,4	43,7	72,9
20-80	386	76,9	62,5	53,5	39,6	70,8
25-80	368	85,4	75,5	71,1	59,9	83,8
30-80	348	83,3	75,9	72,0	56,4	81,5
35-80	323	76,4	65,2	52,0	43,5	73,3
50-80	228	73,8	64,3	58,1	44,4	69,7

Tabulka 17: Hledání spodní meze, dvojice tříd, binární reprezentace, popis NOMINAL, spodní mez



Obrázek 17: Hledání spodní meze, dvojice tříd, binární reprezentace, popis NOMINAL, spodní mez

	počet příznaků	NB [%]	J48 [%]	ID3 [%]	IB1 [%]	SMO [%]
původní	368	85,4	75,5	71,1	59,9	83,8
základní tvar	151	60,0	40,8	45,7	57,6	50,9

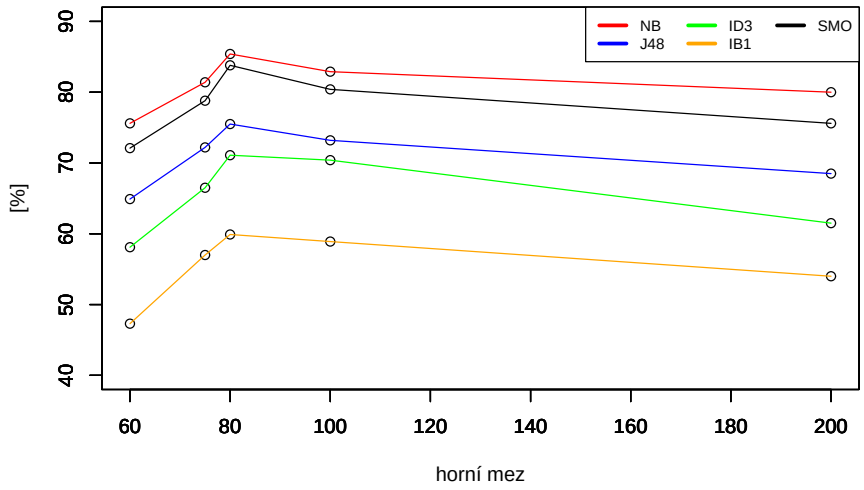
Tabulka 18: Úprava na základní tvar, dvojice tříd, binární reprezentace, interval <25;80>

	trénovací sada 1	trénovací sada 2
skupina 1	celebrity	celebrity
	počítače	z domova
skupina 2	z domova	ekonomika
	ze světa	ze světa
skupina 3	ekonomika	počítače
	sport	sport

Tabulka 19: Tabulka rozdělení témat

	počet příznaků	NB [%]	J48 [%]	ID3 [%]	IB1 [%]	SMO [%]
25-70	417	75,0	63,2	57,0	41,6	73,8
25-75	435	87,3	80,6	80,9	63,8	85,4
25-80	436	79,1	64,1	60,4	40,9	72,1
25-85	422	78,8	68,6	59,9	40,9	76,0

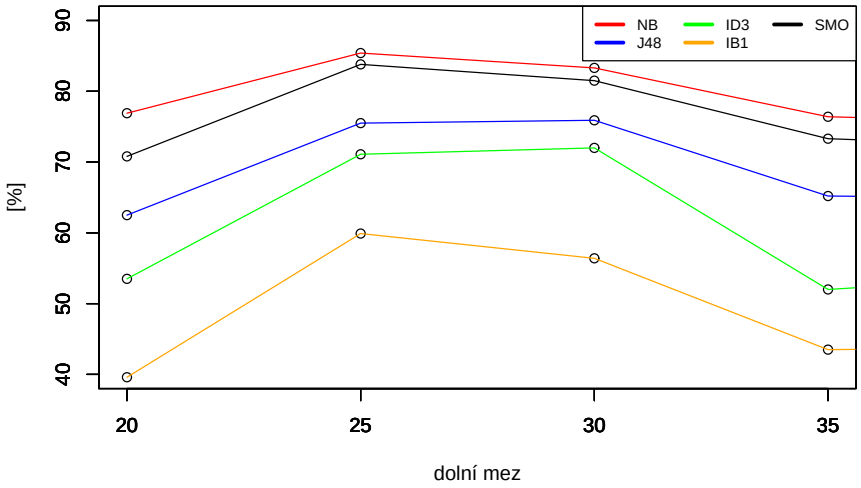
Tabulka 20: Hledání horní meze, dvojice tříd – jiné rozdělení, binární reprezentace, popis NOMINAL



Obrázek 18: Hledání horní meze, dvojice tříd – jiné rozdělení, binární reprezentace, popis NOMINAL

	počet příznaků	**NB [%]**	**J48 [%]**	**ID3 [%]**	**IB1 [%]**	**SMO [%]**
20-75	453	85,4	78,5	70,2	54,6	82,3
25-75	435	87,3	80,6	80,9	63,8	85,4
30-75	413	76,0	65,3	57,2	39,4	73,0

Tabulka 21: Hledání spodní meze, dvojice tříd – jiné rozdělení, binární reprezentace, popis NOMINAL



Obrázek 19: Hledání spodní meze, dvojice tříd - jiné rozdělení, binární reprezentace, popis NOMINAL

4.3.2 Frekvenční reprezentace příznakového vektoru

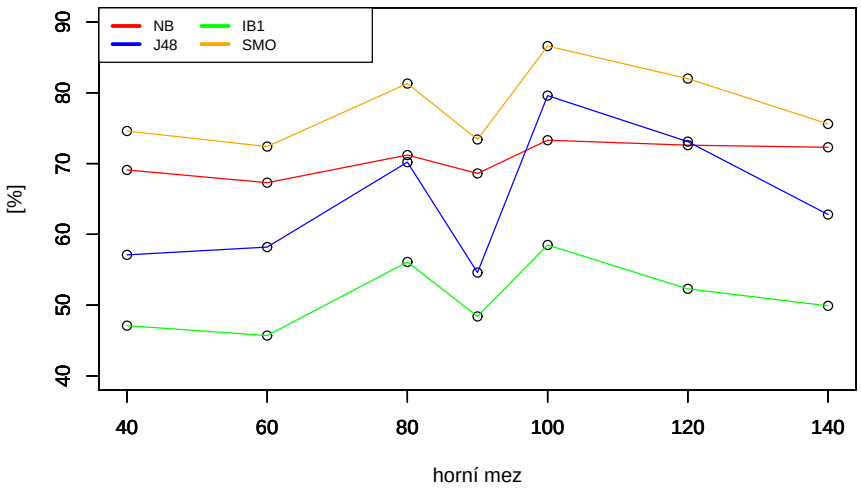
Frekvenční reprezentace se u hledání s dvojicí tříd oproti hledání s využitím všech tříd neosvědčila. Binární reprezentace zde vykazuje lepší hodnoty. Výhodou frekvenční reprezentace je menší citlivost na zvolený interval, výsledky mají konstantnější chování. Výsledky jsou v tabulkách 22 a 23.

4.3.3 TF-IDF reprezentace příznakového vektoru

U TF-IDF reprezentace je dosaženo podobných hodnot jako u frekvenční reprezentace, viz tabulky 24 a 25. Dá se říci, že mezi nimi není rozdíl.

interval	počet příznaků
25-40	610
25-60	471
25-80	452
25-90	448
25-100	457
25-120	461
25-140	446

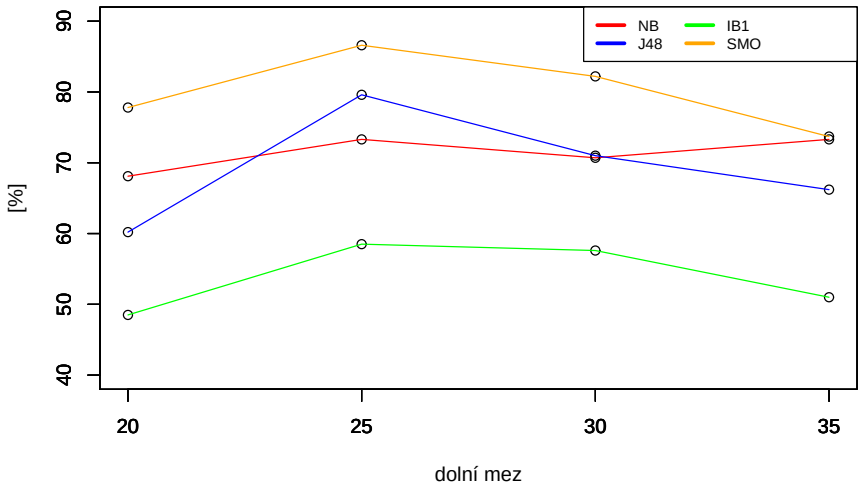
Tabulka 22: Hledání horní meze, dvojice tříd, frekvenční reprezentace



Obrázek 20: Hledání horní meze, dvojice tříd, frekvenční reprezentace

interval	počet příznaků
20-100	466
25-100	457
30-100	425
35-100	407

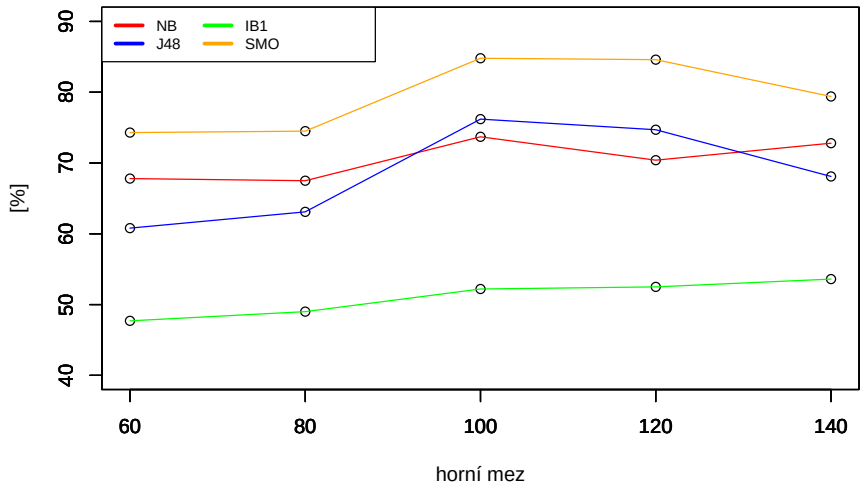
Tabulka 23: Hledání dolní meze, dvojice tříd, frekvenční reprezentace



Obrázek 21: Hledání dolní meze, dvojice tříd, frekvenční reprezentace

interval	počet příznaků
25-60	487
25-80	449
25-100	455
25-120	447
25-140	443

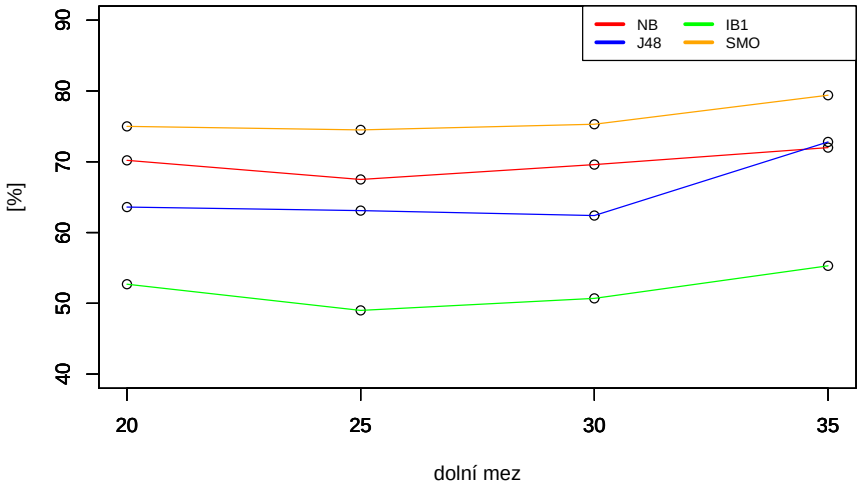
Tabulka 24: Hledání horní meze, dvojice tříd, TF-IDF reprezentace



Obrázek 22: Hledání horní meze, dvojice tříd, TF-IDF reprezentace

interval	**počet příznaků**
20-80	467
25-80	449
30-80	425
35-80	403

Tabulka 25: Hledání spodní meze, dvojice tříd, TF-IDF reprezentace



Obrázek 23: Hledání spodní meze, dvojice tříd, TF-IDF reprezentace

Závěr

Cílem diplomové práce bylo popsat a vyzkoušet metody pro automatickou detekci témat. Podstatná část práce je věnována seznámení s použitými algoritmy a s možnostmi, které nabízí systém WEKA. Dále je popsán postup získání a předzpracovaní textů.

Systém WEKA se ukázal jako velice silný a komplexní nástroj pro rozpoznávání, klasifikaci a shlukovou analýzu. Výhodou tohoto systému je velké množství rozsáhlých funkcí a algoritmů. Program WEKA je plně srovnatelný s jinými komerčními systémy, čímž se dá zmínit další nezanedbatelná výhoda – jedná se o freewareový program.

Jako problematické se jeví získání dostatečného množství kvalitních, volně dostupných testovacích dat roztříděných do témat vhodných k učení klasifikátoru. Při použití většího množství dat se dosti zásadně prodlužuje doba zpracování. Čas potřebný pro jednotlivé testy se pak pohyboval i v řádu hodin. Vyzkoušeny byly tři různé reprezentace dat (binární, frekvenční, TF-IDF) a čtyři různé algoritmy k hledání příznaků (J48, ID3, NaiveBayes a SMO).

Metoda hledání příznaků po dvojicích témat je přínosná z hlediska dosažených úspěšností. Není u ní třeba používat složitějších reprezentací příznakového vektoru, s jednodušší binární reprezentací dosahuje dokonce lepších výsledků než s frekvenční nebo TF-IDF reprezentací. Frekvenční a TF-IDF reprezentace je oproti binární reprezentaci méně náročná na volbu správného rozsahu intervalu výskytů slov v příznakovém vektoru, což může být výhodné. Z algoritmů se jako nejlepší jeví použití algoritmu NaiveBayes. Dosahovalo se s ním nejlepších výsledků, navíc byl bezkonkurenčně nejrychlejší. Dobrých výsledků dosahoval rovněž algoritmus SMO, který je bohužel mnohonásobně pomalejší oproti algoritmu NaiveBayes. Pro algoritmy NaiveBayes a SMO hovoří i malé výkyvy v úspěšnosti třídění. Algoritmy J48 a ID3 jsou více citlivé na výběr reprezentace příznakového vektoru a na výběr slov v příznakovém vektoru. Zklamáním byla úprava globálního slovníku do základního tvaru, která nepřinesla očekávané výsledky.

Při dalším testování by bylo dobré věnovat více prostoru zejména výběru trénovací sady dat, aby obsahovala skutečně kvalitní data, jinak nelze očekávat dobře

nacvičený klasifikátor. Ani výběr reprezentace příznakového vektoru a volba vhodného algoritmu klasifikátoru pak tento nedostatek nevyváží.

Literatura

Reference

[1] NOUZA, Jan. *Přednášky z předmětu MROX*

[2] *Definícia kategorizácie textov*. URL: alfa.intrak.tuke.sk/∼bodnarova/-ea/zadania/MR/prezentacia/final.doc

[3] *Současný stav a trendy automatické indexace dokumentů*. URL: <www.ikaros.cz/node/1300>

[4] HROZA, Jiří. *Automatizovaná podpora filtrace elektronických dokumentů metodami strojového učení*. MU FI, URL: www.fi.muni.cz/∼xhroza1/-hroza04_teze.pdf

[5] BERKA, Petr. *Aplikace systémů dobývání znalostí pro analýzu medicínských dat*. EuroMISE, URL: <euromise.vse.cz/kdd>

[6] SEDLÁČEK, Radek. *Morfologický analyzátor češtiny*. Diplomová práce, MU FI, URL: <nlp.fi.muni.cz/projekty/ajka/ajka.pdf>

[7] MATOUŠEK, Zbyněk. *Podobnost dokumentů*. URL: <nlp.fi.muni.cz/nlp/files/index_DB.html>

[8] SATRAPA, Pavel. *Perl pro zelenáče*. Neokortex, 2000, ISBN 80-86330-02-8

[9] *ActivePerl User Guide*, dokumentace ActivePerl

[10] *The Perl CD BookShelf v3.0*. O'Reilly & Associates, 2002, ISBN 0-596-00389-7

[11] *WEKA*. URL: <www.cs.waikato.ac.nz/∼ml/weka>

[12] *NLP*. URL: <nlp.fi.muni.cz/nlp>

[13] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J., kolektiv. *Umělá inteligence(1)*. Praha, Academia, 1993, ISBN 80-200-0496-3

[14] MAŘÍK, V., ŠTĚPÁNKOVÁ, O., LAŽANSKÝ, J., kolektiv. *Umělá inteligence(4).* Praha, Academia, 2003, ISBN 80-200-1044-0

[15] KOTEK, Z., CHALUPA, V., BRŮHA, I., JELÍNEK, J. *Adaptivní a učící se systémy.* Praha, SNTL, 1980, ISBN 04-515-80

[16] BERKA, P., *Dobývání znalostí z databází.* Praha, Academia, 2003, ISBN 80-200-1062-9

Software

- ActivePerl 5.8.7.813
- OpenIDE Perl 1.1
- WEKA 3.5.7
- Mozila Firefox 2.0, DOM Inspector

A Přílohy

A.1 Stahování a extrakce textu z HTML

blisty_stahovac.pl

Stahování článků ze serveru `www.blisty.cz`.

```
#!/usr/bin/perl
use LWP::Simple;
use locale;
use File::Path;
use Time::Local;

$od=25000;#14715;#8800;
$do=30000;
$kam="./down";
mkpath $kam;

for($cislo=$od;$cislo<=$do;$cislo++)
{
  $status=getstore("http://www.blisty.cz/art/$cislo.html","$kam"."/"."$cislo.html");
  print STDERR "$cislo : $status ";
  print STDERR "Ok\n" if is_success($status);
  print STDERR "Error\n" if is_error($status);
}
print "\n...konec";
```

blisty_vytahovac.pl

Extrahování textu z článků ze severu `www.blisty.cz`.

```
#!/usr/bin/perl
use locale;
use HTML::TreeBuilder;
use File::Path;
use Prekodovac;

$in="./down";
$out="./_texty";
#================================
@clanky=&list($in);
foreach $clanek(sort @clanky)
{
  print STDERR "$clanek\n";
  if(-e $clanek)
  {
    my $tree=HTML::TreeBuilder->new_from_file($clanek);
    $dotaz=$tree->find_by_attribute('class','datum');
    $datum=$dotaz->as_text( );
    @datum=split(/\./,$datum);    #rozkouskuje podle \. na pole
    foreach $dat(@datum){$dat=~s/ //;}
    if(length($datum[1])==1){$datum[1]="0".$datum[1];}
    $datum=$datum[2]."_".$datum[1];#."_".$datum[0];
    $cesta=$out."/".$datum;
    mkpath($cesta);
    $cesta=$cesta.substr($clanek,rindex($clanek,"/"));
    $cesta=~s/\.html/\.txt/;
    open(OUT,">$cesta");
    foreach $dotaz($tree->find_by_attribute('class','telo'))
    {
      $obsah=$dotaz->as_text( );
      print OUT $obsah;
    }
    close OUT;
  }
}
print STDOUT "\n...hotovo.";

### subs ###
sub list{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "Nelze otevrit!\n";
    return 0;
```

```
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  return @seznam;
}
```

sport_stahovac.pl

Stahování článků ze serveru `www.deniksport.cz`.

```
#!/usr/bin/perl
use LWP::Simple;
use locale;
use File::Path;
use Time::Local;

$od=410291;#14715;#8800;
$do=410000;
$kam="./down/410";
mkpath $kam;

for($cislo=$od;$cislo>=$do;$cislo--)
{
  $status=getstore("http://www.deniksport.cz/Clanek$cislo.htm","$kam"."/"."$cislo.html");
  print STDERR "$cislo : $status ";
  print STDERR "Ok\n" if is_success($status);
  print STDERR "Error\n" if is_error($status);
}
print "\n...konec";
```

sport_vytahovac_1250.pl

Extrahování textu z článků ze serveru `www.deniksport.cz`.

```
#!/usr/bin/perl
use locale;
use HTML::TreeBuilder;
use File::Path;
use Prekodovac;
use Cz2Cz;

$in="./down/435";
$out="./_texty";
#================================
@clanky=&list($in);
foreach $clanek(sort @clanky)
{
  print STDERR "$clanek\n";
  if(-e $clanek)
  {
    open(F,$clanek);
    $text=join ' ',<F>;
    $text=~s/<br>/!~br~!/gi;
    $text=Prekodovac::prekoduj($text);
    close(F);

    $text=preklad('iso88592to1250',$text);


    my $tree=HTML::TreeBuilder->new_from_content($text);
#    my $tree=HTML::TreeBuilder->new_from_file($clanek);

    $dotaz=($tree->find_by_attribute('class','perex'));
    if ($dotaz eq undef) {next;}
    $perex=$dotaz->as_text( );

    $dotaz=$tree->find_by_attribute('id','hdr2-vyroci');
        if ($dotaz eq undef) {next;}
    $datum=$dotaz->as_text( );

    @dat=split(/\./,$datum);
    foreach $dat(@dat){$dat=~s/\D//gi;}
    $datum=substr($dat[2],0,4)."_".$dat[1];

    $dotaz=$tree->find_by_attribute('id','hdr2-datum');
        if ($dotaz eq undef) {next;}
    $tema=$dotaz->as_text( );
    $tema=substr($tema,index($tema,':')+2);
    $tema=~s/\d//gi;
```

```
    $cesta=$out."/".$datum."/".$tema;
    mkpath($cesta);
    $cesta=$cesta.substr($clanek,rindex($clanek,"/"));
    $cesta=~s/\.html/\.txt/;

    open(OUT,">$cesta");
    $dot=($tree->find_by_attribute('class','obsah-in1'));
        if ($dot eq undef) {next;}
    foreach $dotaz($dot->find_by_tag_name('p'))
    {
      $obsah=$dotaz->as_text( );
       $obsah=~s/\!\~br\~\!/\n/g;
      print OUT $obsah;
    }
    close OUT;
  }
}
print STDOUT "\n...hotovo.";

### subs ###
sub list{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "Nelze otevrit!\n";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  return @seznam;
}
```

pcw_ stahovac.pl

Stahování článků ze serveru `www.pcworld.cz`.

```
#!/usr/bin/perl
use LWP::Simple;
use locale;
use File::Path;
use Time::Local;
use HTML::TreeBuilder;

$kam="./down";
rmtree $kam;
mkpath $kam;
#++++++++++++++++++++++++++++

$status=getstore("http://www.pcworld.cz/pcw.nsf/html/archiv.htm","./archiv.html");
print STDERR "$status ";
print STDERR " Ok \n" if is_success($status);
print STDERR " Error \n" if is_error($status);

my $tree=HTML::TreeBuilder->new_from_file("./archiv.html");
foreach $dotaz($tree->find_by_tag_name('a'))
{
  $ttt=$dotaz->attr('href');
  if ($ttt=~/OpenView&ExpandView&Count=200/)
  {
    push(@vydani,$ttt);
  }
}

foreach $vydani(@vydani)
{
  $status=getstore($vydani,"./pcw.html");
  my $tree=HTML::TreeBuilder->new_from_file("./pcw.html");
  $cesta=$vydani;
  $cesta=~s/\?OpenView&ExpandView&Count=200//;
  $cesta=$kam."/".substr($cesta,rindex($cesta,"/pcw.nsf/")+9);
  mkpath($cesta);
  foreach $dotaz($tree->find_by_tag_name('a'))
  {
    $soubor=$dotaz->attr('href');
    if (($soubor=~/\/pcw.nsf\//)and($soubor=~/\?OpenDocument/))
    {
      $s="http://www.pcworld.cz".$soubor;
      $soubor=~s/\?OpenDocument//;
      $soubor=substr($soubor,rindex($soubor,"/")+1);
      $cil=$cesta."/".$soubor.".html";
      print STDERR "$cil ";
```

```
      $st=getstore($s,$cil);
      print STDERR ": $st\n";
    }
  }
}
print "\n...konec";
```

pcw_vytahovac.pl

Extrahování textu z článků ze severu `www.pcworld.cz`.

```
#!/usr/bin/perl
use locale;
use HTML::TreeBuilder;
use File::Path;
use Prekodovac;

$in="./down";
$out="./_texty";
#================================
@clanky=&list($in);
foreach $clanek(sort @clanky)
{
  print STDERR "$clanek\n";
  if(-e $clanek)
  {
    my $tree=HTML::TreeBuilder->new_from_file($clanek);

    $text=$clanek;
    $text=~s/$in/$out/;
    $text=~s/.html/.txt/;
    $cesta=substr($text,0,rindex($text,"/"));
    mkpath($cesta);
    $dot=$tree->find_by_attribute('class','breadcrumbs');
    @dotaz=$dot->find_by_tag_name('p');
    open(OUT,">$text");
    $obsah=$dotaz[1]->as_text( );
    $obsah=~s/\!~BR~\!/\n/g;
    print OUT $obsah;
    close OUT;
  }
}
print STDOUT "\n...hotovo.";

  ### subs ###
sub list{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "Nelze otevrit!\n";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
```

```
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  return @seznam;
}
```

atlas_stahovac.pl

Stahování článků ze serveru `zpravy.atlas.cz`.

```
#!/usr/bin/perl
use LWP::Simple;
use locale;
use File::Path;
use Time::Local;

@cas=localtime(time);
$t1=time;

$od=4999;
$do=0000;#96050;
$kam="./down/000_2";
mkpath $kam;

for($cislo=$od;$cislo>=$do;$cislo--)
{
  $status=getstore("http://zpravy.atlas.cz/clanky/$cislo.aspx","$kam"."/"."$cislo.html");
  print STDERR "$cislo: $status ";
  print STDERR "Ok\n" if is_success($status);
  print STDERR "Error\n" if is_error($status);
}
printf STDERR "\n%02d:%02d:%02d",$cas[2],$cas[1],$cas[0];
@cas=localtime(time);
printf STDERR "\n%02d:%02d:%02d",$cas[2],$cas[1],$cas[0];
print "\n...konec";
```

atlas_ vytahovac.pl

Extrahování textu z článků ze severu `zpravy.atlas.cz`.

```
#!/usr/bin/perl
use locale;
use HTML::TreeBuilder;
use File::Path;
use Prekodovac;

$in="./down/065";
$out="./_texty065";
@spec=qw(\/ \# \: \* \? \" \< \> \| \. \,);
mkpath $out;
#==============================
@cas=localtime(time);
$t1=time;

@clanky=&list($in);
splice(@clanky,0,2240);

foreach $clanek(@clanky)
{
  print STDERR "\n$clanek ";
  if(-e $clanek)
  {
    open(F,$clanek);
    $text=join ' ',<F>;
    $text=~s/<br>/!~br~!/gi;
    $text=Prekodovac::prekoduj($text);
    close(F);

    my $tree=HTML::TreeBuilder->new_from_content($text);
    $dotaz=$tree->find_by_attribute('class','perex');
    if ($dotaz eq undef){next;}
    $perex=$dotaz->as_text( );

    $dotaz=$tree->find_by_attribute('class','time');
    $datum=$dotaz->as_text( );
    $datum=substr($datum,0,index($datum," ",8));
    @datum=split(/\./,$datum);    #rozkouskuje podle \. na pole
    foreach $dat(@datum){$dat=~s/ //;}
    if(length($datum[1])==1){$datum[1]="0".$datum[1];}
    $datum=$datum[2]."_".$datum[1];#."_".$datum[0];

    $dotaz=($tree->find_by_attribute('class','group'));
    if ($dotaz eq undef){next;}
    $group=$dotaz->as_text( );
    foreach $sp(@spec){$group=~s/$sp//g;}
```

```
    print STDERR "$datum $group";

    $cesta=$out."/".$group."/".$datum;
    mkpath($cesta);

#    eval{mkpath($cesta)};
#    if ($@){next;}

    $soubor=substr($clanek,rindex($clanek,"/")+1);
    $soubor=~s/.html/.txt/;
    open(OUT,">$cesta/$soubor");
    print OUT "$perex ";
    $dotaz = $tree->find_by_attribute('class','content');
    {
      $obsah=$dotaz->as_text( );
      $obsah=~s/\!\~br\~\!/\n/g;
      print OUT "$obsah ";
    }
    close OUT;
  }
}
printf STDERR "\n%02d:%02d:%02d",$cas[2],$cas[1],$cas[0];
@cas=localtime(time);
printf STDERR "\n%02d:%02d:%02d",$cas[2],$cas[1],$cas[0];
print STDOUT "\n...hotovo.";

### subs ###
sub list{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "Nelze otevrit!\n";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(@soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  return @seznam;
}
```

aktualne_stahovac.pl

Stahování článků ze serveru `aktualne.centrum.cz`.

```
#!/usr/bin/perl
use locale;
use HTML::TreeBuilder;
use LWP;
use File::Path;

$browser = LWP::UserAgent->new( );
$browser->agent('Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040206 Firefox/0.8');
$browser->default_header('Accept-Language' => "cs,en-us;q=0.8,en;q=0.5,fj;q=0.3");
# $browser->default_header('Accept-Charset' => "ISO-8859-2,utf-8;q=0.7,*;q=0.7");
$browser->default_header('Accept-Charset' => "windows-1250");

$od=$ARGV[0];
#$od=307484;
$do=0;

$out="./down";
mkpath $out;
@spec=qw(\/ \# \: \* \? \" \< \> \|);
#$clanek="aktualne.html";
#================================

for($cislo=$od;$cislo>$do;$cislo--)
{
  print STDERR "\n$cislo: ";
  $response = $browser->get("http://aktualne.centrum.cz/clanek.phtml?id=$cislo");
  $status = $response->{'_rc'};
  $content = $response->content;
  if ($status=200)
  {
    my $tree=HTML::TreeBuilder->new_from_content($content);
    $dotaz=($tree->find_by_tag_name('h2'));
    if ($dotaz eq undef) {last;}
    $dotaz=$dotaz->as_text( );
    if (not $dotaz=~/Článek nenalezen/)
    {
      $clanek="$out/$cislo.html";
      open FILE, ">$clanek";
      print FILE $content;
      close FILE;
      print STDERR "Ok"
    }
  }
}
print STDOUT "\n...hotovo.";
```

aktualne_vytahovac.pl

Extrahování textu z článků ze serveru `aktualne.centrum.cz`.

```
#!/usr/bin/perl
use locale;
use HTML::TreeBuilder;
use LWP;
use File::Path;

$in="./down";
$out="./_texty";
#mkpath $out;
@spec=qw(\/ \# \: \* \? \" \< \> \| \.);
#==============================
@clanky=&list($in);
foreach $clanek(@clanky)
{
    my $tree=HTML::TreeBuilder->new_from_file($clanek);
    $dotaz=($tree->find_by_attribute('id','hlavni-obsah'));
    if ($dotaz eq undef){next;}

    $topic=$dotaz->find_by_tag_name('span');

    if ($topic eq undef){next;}

    $topic=$topic->as_text( );
    foreach $sp(@spec){$topic=~s/$sp//g;}
#    print STDERR "\n$topic\n";

#    $soubor=substr($clanek,rindex($clanek,"/")+1);
#    print STDOUT "$soubor\n";
#    $soubor=~s/.html/.txt/;
#    print STDOUT "$soubor\n";
    $dotaz=$tree->find_by_attribute('class','datum');
    $datum=$dotaz->as_text();
    $datum=substr($datum,length($datum)-4);
    foreach $sp(@spec){$datum=~s/$sp//g;}
    $cesta=$out."/".$topic."/".$datum;
    mkpath($cesta);
    print STDERR "\n$clanek";
    open(OUT,">$cesta"."/".$cislo.".txt");
    $dotaz=$tree->find_by_attribute('class','clanek');
    if ($dotaz eq undef){close OUT;next;}
    @dot=$dotaz->find_by_tag_name('p');
    if ($dotaz eq undef){close OUT;next;}
    splice(@dot,0,2);

    foreach $dot(@dot)
```

```
    {
      $tema=$dot->as_text();
        substr($tema,0,index($tema,"-"),"");
#        $tema=~s/\!~BR~\!/\n/g;
      print OUT $tema;

    }
    close OUT;

}
print STDOUT "\n...hotovo.";

### subs ###
sub list{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "Nelze otevrit!\n";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  return @seznam;
}
```

A.2 Tvorba trénovacích ARFF souborů

Binární reprezentace – *bin.pl*

```
#!/usr/bin/perl
use locale;
#[1] vytvori seznam slov s poctem souboru, kde se vyskytuje

$stop="c:/perl/__data/stop_list_0.txt";
 $adr="c:/perl/__data/training3/3";
 $slv="c:/perl/__data/bin2.txt";
$arff="c:/perl/__data/bin2.arff";

@seznam=&list($adr);
#================================================
####################nacteni stop listu
open STOP,$stop;
$stoplist=join ' ',<STOP>;
$stoplist="\L$stoplist\E";
close STOP;
@stoplist=split(/[\W_]+/,$stoplist);
foreach (@stoplist){$stoplist{$_}=1;}
print STDERR "\n stop list nacten";

####################vytvoreni slovniku
foreach $data(sort @seznam)
{
  if(-e $data)
  {
    open(F,$data);
    $page=undef;
    $page=join ' ',<F>;
    close(F);
  }
  $page="\L$page\E";
  $page=~s/^bgn\s*//;#na hranici slova, dalsi radek, vsechny vyskyty
  @ss=();
  @ss=split(/[\W_]+/,$page);

  %podslovnik=();
  foreach (@ss)
  {
    if ($_=~/\d+/){next;}         #bez cisel

    if ($_!~/\w{2,}/){next;}     #ma alespon 2 znaky

    if (!exists($stoplist{$_}))   #porovnani se stoplistem
    {
```

```
      if (!exists($podslovnik{$_})) {$podslovnik{$_}=1;}  #pocet vyskytu v 1.dokumentu
      else {$podslovnik{$_}++;}

      if (!exists($slovnik{$_})) {$slovnik{$_}=0;}#vytvori novou polozku ve slovniku
    }                                             #glob slovnik
  }
  $i++;
  print STDERR "\n$i";
  foreach (keys %podslovnik) {$slovnik{$_}++;}  #pocet dokumentu, kde se vyskytuje ono slovo alespon 1x
}
$i=0;
print STDERR "\n slovnik vytvoren";

foreach (keys %slovnik)
{
#  if (($slovnik{$_}<25)or($slovnik{$_}>60)) {delete($slovnik{$_});}  #vyskyt vetsi nez..
  if ($slovnik{$_}<100) {delete($slovnik{$_});}
}

print STDERR "\n slovnik upraven";

open(SLV,">$slv");
foreach (sort keys %slovnik){print SLV "$slovnik{$_} $_\n";}
#foreach (sort keys %slovnik){print SLV "$_\n";}
close SLV;

####################vytvoreni *.arff a hlavicky
open(ARFF,">$arff")
  or die print "Nelze vytvorit data.arff!";
print ARFF "\@RELATION data\n\n";
foreach (sort keys %slovnik)
{
#  print ARFF "\@ATTRIBUTE $_ REAL\n";
  print ARFF "\@ATTRIBUTE $_ {0,1}\n";

}
if(not opendir(ADR,$adr))
{
  print STDERR "Nelze otevrit!\n";
}
@soubory=readdir(ADR);
closedir(ADR);
splice(@soubory,0,2);
$st="\@ATTRIBUTE #třída# {";
foreach (@soubory)
{
  $st.=$_.",";
```

```
}
$st=substr($st,0,rindex($st,',')).")";
print ARFF "$st\n\n";
print ARFF "\n\n\@DATA \n\n";

####################zapis vektoru
foreach $data(@seznam)
{
  $i++;
  print STDERR "\n$i";

  if(-e $data)
  {
    open(F,$data);#nacteni
    $page=undef;
    $page=join ' ',<F>;
    close(F);
    $page="\L$page\E";
  }
  @text=split(/[\W_]+/,$page);

  %pole=();
  foreach (@text)
  {
    $pole{$_}=1;
  }
  $vektor=undef;
  foreach $slovo(keys(%slovnik))
  {
    if (exists($pole{$slovo}))
    {
      $vektor.="1,"
#      print ARFF "1,";
    }
    else
    {
      $vektor.="0,"
#      print ARFF "0,";
    }
  }
  print ARFF $vektor;
  $tetema=$data;
  $tetema=~s/$adr//;
  $tetema=substr($tetema,1,index($tetema,'/',1)-1);
  print ARFF "$tetema\n\n";
}
close ARFF;
```

```
print STDERR "\n hotovo...";


### subs ###
sub list{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "Nelze otevrit!\n";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  return @seznam;
}
```

Frekvenční reprezentace – *tf.pl*

```
#!/usr/bin/perl
use locale;

$od=40;
$do=200;
$stop="c:/perl/__data/stop_list.txt";
    $adr="c:/perl/__data/training1";
$file="c:/perl/__data/tf_training1_6t_".$od."_".$do;

@seznam=&list($adr);  #nacte prislusny adresar
&stop;                #nacteni stoplistu
&glob_slovnik;        #vytvori seznam vsech slov, u slova je uveden celkovz pocet vyskytu
&zapis_slovniku($od,$do);
&arff_hlavicka;
&zapis_vektoru;
print STDERR "\nkonec";

####################nacteni stop listu
sub stop
{
  print STDERR "\nnacteni stoplistu";
  open STOP,$stop;
  $stoplist=join ' ',<STOP>;
  $stoplist="\L$stoplist\E";
  close STOP;
  @stoplist=split(/[\W_]+/,$stoplist);
  foreach (@stoplist){$stoplist{$_}=1;}
}

####################vytvoreni globalniho slovniku
sub glob_slovnik
{
  print STDERR "\nvytvoreni glob. slovniku";
  $i=0;
  foreach $data(sort @seznam)
  {
    if(-e $data)
    {
      open(F,$data);
      $page=undef;
      $page=join ' ',<F>;
      close(F);
    }
    $page="\L$page\E";
    $page=~s/^bgn\s*//;#na hranici slova, dalsi radek, vsechny vyskyty
    @ss=();
```

```
    @ss=split(/[\W_]+/,$page);

    foreach (@ss)
    {
      if ($_=~/\d+/){next;}         #bez cisel
      if ($_!~/\w{2,}/){next;}     #ma alespon 2 znaky
      if (!exists($stoplist{$_}))   #porovnani se stoplistem
      {
        if (!exists($slovnik{$_})) {$slovnik{$_}=1;}  #vytvori glob. slovniku
        else {$slovnik{$_}++;}
      }
    }
    $i++;
    print STDERR "\n$i";
  }
}

####################zapis slovniku
sub zapis_slovniku
{
  my $od,$do;
  $od=shift;
  $do=shift;
  print STDERR "\nzapis slovniku";
  foreach (keys %slovnik) #uprava poctu vyskytu - neni stejne jako v pripade binarni reprezentace!!!!
  {
    if (($slovnik{$_}<$od)or($slovnik{$_}>$do)) {delete($slovnik{$_});}  #vyskyt vetsi nez..
  #  if ($slovnik{$_}<$od) {delete($slovnik{$_});}
  }
  open(SLV,">$file.txt");
  foreach (sort keys %slovnik){print SLV "$slovnik{$_} $_\n";}
  #foreach (sort keys %slovnik){print SLV "$_\n";}
  close SLV;
}

####################vytvoreni *.arff a hlavicky
sub arff_hlavicka
{
  print STDERR "\narff hlavicka";
  open(ARFF,">$file.arff")
    or die print "\nNelze vytvorit .arff!";
  $nazev=substr($file,rindex($file,'/')+1);
  print ARFF "\@RELATION $nazev\n\n";
  foreach (sort keys %slovnik)
  {
    print ARFF "\@ATTRIBUTE $_ REAL\n";
  }
```

```
  if(not opendir(ADR,$adr))
  {
    print STDERR "\nNelze otevrit!";
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  splice(@soubory,0,2);
  $st="\@ATTRIBUTE #třída# {";
  foreach (@soubory)
  {
    $st.=$_.",";
  }
  $st=substr($st,0,rindex($st,',')).}";
  print ARFF "$st\n\n";
  print ARFF "\n\n\@DATA \n\n";
}

####################zapis vektoru
sub zapis_vektoru
{
  print STDERR "\nzapis vektoru";
  $i=0;
  foreach $data(@seznam)
  {
    if(-e $data)
    {
      open(F,$data);#nacteni
      $page=undef;
      $page=join ' ',<F>;
      close(F);
      $page="\L$page\E";
    }
    @text=split(/[\W_]+/,$page);

    %podslovnik=();

    foreach (@text)
    {
      if ($_=~/\d+/){next;}         #bez cisel
      if ($_!~/\w{2,}/){next;}      #ma alespon 2 znaky
      if (!exists($stoplist{$_}))   #porovnani se stoplistem
      {
        if (!exists($podslovnik{$_})) {$podslovnik{$_}=1;}
        else {$podslovnik{$_}++;}
      }
    }
```

```
    $dok=0;
    foreach (keys %podslovnik)
    {
      $dok+=$podslovnik{$_};        #pocet pouzitelnych slov v dokumentu
    }

    $vektor=undef;
    foreach (keys(%slovnik))
    {
      if (!exists($podslovnik{$_})) {$hod=0;}
      else {$hod=$podslovnik{$_}/$dok;} #vypocet hodnoty TF
      $vektor.="$hod,";
    }
    print ARFF $vektor;
    $tetema=$data;
    $tetema=~s/$adr//;
    $tetema=substr($tetema,1,index($tetema,'/',1)-1);
    print ARFF "$tetema\n\n";             #zapis tridy na konec vektoru
    $i++;
    print STDERR "\n$i";
  }
  close ARFF;
}

####################seznam souboru
sub list
{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "\nNelze otevrit!";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  print STDERR "\nnacteni souboru";
  return @seznam;
}
```

TF-IDF reprezentace – *tfidf.pl*

```
#!/usr/bin/perl
use locale;

$od=40;
$do=200;
$stop="c:/perl/__data/stop_list.txt";
       $adr="c:/perl/__data/training2/3";
$file="c:/perl/__data/tfidf_training2_3_".$od."_".$do;

@seznam=&list($adr);  #nacte prislusny adresar
#&stop;                #nacteni stoplistu
&glob_slovnik;        #vytvori seznam vsech slov, u slova je uveden celkovy pocet vyskytu
&zapis_slovniku($od,$do);
&arff_hlavicka;
&zapis_vektoru;
print STDERR "\n$file";
print STDERR "\nkonec";

####################nacteni stop listu
sub stop
{
  print STDERR "\nnacteni stoplistu";
  open STOP,$stop;
  $stoplist=join ' ',<STOP>;
  $stoplist="\L$stoplist\E";
  close STOP;
  @stoplist=split(/[\W_]+/,$stoplist);
  foreach (@stoplist){$stoplist{$_}=1;}
}

####################vytvoreni globalniho slovniku
sub glob_slovnik
{
  print STDERR "\nvytvoreni glob. slovniku";
  $i=0;
  foreach $data(sort @seznam)
  {
    if(-e $data)
    {
      open(F,$data);
      $page=undef;
      $page=join ' ',<F>;
      close(F);
    }
    $page="\L$page\E";
    $page=~s/^bgn\s*//;#na hranici slova, dalsi radek, vsechny vyskyty
```

```
    @ss=();
    @ss=split(/[\W_]+/,$page);

    foreach (@ss)
    {
      if ($_=~/\d+/){next;}         #bez cisel
      if ($_!~/\w{2,}/){next;}      #ma alespon 2 znaky
      if (!exists($stoplist{$_}))   #porovnani se stoplistem
      {
        if (!exists($slovnik{$_})) {$slovnik{$_}=1;}  #vytvori glob. slovniku
        else {$slovnik{$_}++;}
        $pod_df{$_}=0;
      }
    }
    foreach (keys %pod_df) {$df{$_}++;}     #df - pocet dokumentu, ve kterych se vyraz vyskytuje
    %pod_df=();
    $i++;
    print STDERR "\n$i";
  }
}

####################zapis slovniku
sub zapis_slovniku
{
  my $od,$do;
  $od=shift;
  $do=shift;
  print STDERR "\nzapis slovniku";
  foreach (keys %slovnik) #uprava poctu vyskytu - neni stejne jako v pripade binarni reprezentace!!!!
  {
    if (($slovnik{$_}<$od)or($slovnik{$_}>$do)) {delete($slovnik{$_});}  #vyskyt vetsi nez..
  #  if ($slovnik{$_}<$od) {delete($slovnik{$_});}
  }
  open(SLV,">$file.txt");
  foreach (sort keys %slovnik){print SLV "$slovnik{$_} $_\n";}
  #foreach (sort keys %slovnik){print SLV "$_\n";}
  close SLV;
}

####################vytvoreni *.arff a hlavicky
sub arff_hlavicka
{
  print STDERR "\narff hlavicka";
  open(ARFF,">$file.arff")
    or die print "\nNelze vytvorit .arff!";
  $nazev=substr($file,rindex($file,'/')+1);
  print ARFF "\@RELATION $nazev\n\n";
```

```
  foreach (sort keys %slovnik)
  {
    print ARFF "\@ATTRIBUTE $_ REAL\n";
  }
  if(not opendir(ADR,$adr))
  {
    print STDERR "\nNelze otevrit!";
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  splice(@soubory,0,2);
  $st="\@ATTRIBUTE #třída# {";
  foreach (@soubory)
  {
    $st.=$_.",";
  }
  $st=substr($st,0,rindex($st,',')."}";
  print ARFF "$st\n\n";
  print ARFF "\n\n\@DATA \n\n";
}

####################zapis vektoru
sub zapis_vektoru
{
  print STDERR "\nzapis vektoru";
  $i=0;
  foreach $data(@seznam)
  {
    if(-e $data)
    {
      open(F,$data);#nacteni
      $page=undef;
      $page=join ' ',<F>;
      close(F);
      $page="\L$page\E";
    }
    @text=split(/[\W_]+/,$page);

    %podslovnik=();

    foreach (@text)
    {
      if ($_=~/\d+/){next;}         #bez cisel
      if ($_!~/\w{2,}/){next;}      #ma alespon 2 znaky
      if (!exists($stoplist{$_}))   #porovnani se stoplistem
      {
        if (!exists($podslovnik{$_})) {$podslovnik{$_}=1;}
```

```
        else {$podslovnik{$_}++;}
      }
    }

    $dok=0;
    foreach (keys %podslovnik)
    {
      $dok+=$podslovnik{$_};        #pocet pouzitelnych slov v dokumentu
    }

    $celk_dok=$#seznam+1;
    $vektor=undef;
    foreach (keys(%slovnik))
    {
#      if (!exists($podslovnik{$_})) {$hod=0;}
#      else {$hod=$podslovnik{$_}/$dok;} #vypocet hodnoty TF

#      $df=
#      $tf=$podslovnik{$_}/$dok;

      if (!exists($podslovnik{$_})) {$tfidf=0;}
      else
      {
        $tfidf=($podslovnik{$_}/$dok)*(log($celk_dok/$df{$_})/log(10)); #vypocet hodnoty TF-IDF
      }
      $vektor.="$tfidf,";
    }
    print ARFF $vektor;
    $tetema=$data;
    $tetema=~s/$adr//;
    $tetema=substr($tetema,1,index($tetema,'/',1)-1);
    print ARFF "$tetema\n\n";             #zapis tridy na konec vektoru
    $i++;
    print STDERR "\n$i";
  }
  close ARFF;
}

####################seznam souboru
sub list
{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "\nNelze otevrit!";
    return 0;
```

```
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  print STDERR "\nnacteni souboru";
  return @seznam;
}
```

A.3 Tvorba testovacích ARFF souborů

Binární reprezentace – *bin_test.pl*

```
#!/usr/bin/perl
use locale;

#[3]

#$adr="c:/perl/_slovnik/ttt";
     $adr="c:/perl/__data/test";
    $arff="c:/perl/__data/test_6t_50_200_nom.arff";#nastavit spravný typ!!!!
$priznaky="c:/perl/__data/priznaky_6t_50_200.txt";
#$priznaky="c:/perl/__data/priznaky_human.txt";
@seznam=&list($adr);
#================================================
####################nacteni slovniku

open(SLV,$priznaky);
$slv=join ' ',<SLV>;
#$slv=~s/\n//gi;
@slovnik=split(/\n/,$slv);
foreach (@slovnik){s/ //gi;}
close SLV;

####################vytvoreni *.arff a hlavicky

open(ARFF,">$arff")
  or die print "Nelze vytvorit data.arff!";
print ARFF "\@RELATION data\n\n";

foreach (@slovnik)
{
#  print ARFF "\@ATTRIBUTE $_ REAL\n";
  print ARFF "\@ATTRIBUTE $_ {0,1}\n";
}
if(not opendir(ADR,$adr))
{
  print STDERR "Nelze otevrit!\n";
}
@soubory=readdir(ADR);
closedir(ADR);
splice(@soubory,0,2);
$st="\@ATTRIBUTE #trida# {";
foreach (@soubory)
{
  $st.=$_.",";
}
```

```
$st=substr($st,0,rindex($st,',')).'}';
print ARFF "$st\n\n";
print ARFF "\n\n\@DATA \n\n";

####################zapis vektoru
foreach $data(@seznam)
{
  $i++;
  print STDERR "\n$i";

  if(-e $data)
  {
    open(F,$data);#nacteni
    $page=undef;
    $page=join ' ',<F>;
    close(F);
    $page="\L$page\E";
  }
  @text=split(/[\W_]+/,$page);

  %pole=();
  foreach (@text)
  {
    $pole{$_}=1;
  }
  foreach $slovo(@slovnik)
  {
    if (exists($pole{$slovo}))
    {
      print ARFF "1,";
    }
    else
    {
      print ARFF "0,";
    }
  }
  $tetema=$data;
  $tetema=~s/$adr//;
  $tetema=substr($tetema,1,index($tetema,'/',1)-1);
  print ARFF "$tetema\n\n";
}
close ARFF;
print "hotovo...";


### subs ###
sub list{
```

```
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "Nelze otevrit!\n";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(@soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  return @seznam;
}
```

TF reprezentace – *tf_test.pl*

```
#!/usr/bin/perl
use locale;

#$stop="c:/perl/__data/stop_list.txt";
# $adr="c:/perl/__data/training2/2";
         $adr="c:/perl/__data/test";
        $arff="c:/perl/__data/test_tf_123_40_200.arff";
$priznaky="c:/perl/__data/priznaky_tf_123_40_200.txt";

@seznam=&list($adr);  #nacte prislusny adresar
&priznaky;
&arff_hlavicka;

####################zapis vektoru
$i=0;
foreach $data(@seznam)
{
  if(-e $data)
  {
    open(F,$data);#nacteni
    $page=undef;
    $page=join ' ',<F>;
    close(F);
    $page="\L$page\E";
  }
  @text=split(/[\W_]+/,$page);

  %podslovnik=();
  foreach (@text)
  {
    if (!exists($podslovnik{$_})) {$podslovnik{$_}=1;}
    else {$podslovnik{$_}++;}
  }
  $dok=0;
  foreach (keys %podslovnik)
  {
    $dok+=$podslovnik{$_};
  }

  $vektor=undef;
#  foreach (sort keys(%slovnik))
  foreach (@slovnik)
  {
    if (!exists($podslovnik{$_})) {$hod=0;}
    else {$hod=$podslovnik{$_}/$dok;}
    $vektor.="$hod,";
```

```
  }
  print ARFF $vektor;
  $tetema=$data;
  $tetema=~s/$adr//;
  $tetema=substr($tetema,1,index($tetema,'/',1)-1);
  print ARFF "$tetema\n\n";
  $i++;
  print STDERR "\n$i";
}
close ARFF;

print STDERR "\nkonec";
################################################################################
################################################################################
################################################################################
### subs ###


####################nacteni priznaku
sub priznaky
{
  open(ATT,$priznaky);
  $att=join ' ',<ATT>;
  @slovnik=split(/\n/,$att);
  foreach (@slovnik){s/ //gi;}
  close ATT;
}

####################vytvoreni *.arff a hlavicky
sub arff_hlavicka
{
  print STDERR "\narff hlavicka";
  open(ARFF,">$arff")
    or die print "Nelze vytvorit .arff!";
  $nazev=substr($arff,rindex($arff,'/')+1);
  print ARFF "\@RELATION $nazev\n\n";
#  foreach (sort keys %slovnik)
  foreach (@slovnik)
  {
    print ARFF "\@ATTRIBUTE $_ REAL\n";
  }
  if(not opendir(ADR,$adr))
  {
    print STDERR "\nNelze otevrit!";
  }
  @soubory=readdir(ADR);
  closedir(ADR);
```

```
  splice(@soubory,0,2);
  $st="\@ATTRIBUTE #třída# {";

  foreach (@soubory)
  {
    $st.=$_.",";
  }
  $st=substr($st,0,rindex($st,',')).")";
  print ARFF "$st\n\n";
  print ARFF "\n\n\@DATA \n\n";
}

####################seznam souboru
sub list
{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "\nNelze otevrit!";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
  print STDERR "\nnacteni souboru";
  return @seznam;
}
```

TF-IDF reprezentace – *tfidf_ test.pl*

```
#!/usr/bin/perl
use locale;

#$stop="c:/perl/__data/stop_list.txt";
# $adr="c:/perl/__data/training2/2";
         $adr="c:/perl/__data/test";
        $arff="c:/perl/__data/test_tfidf_6t_40_200.arff";
$priznaky="c:/perl/__data/priznaky_tfidf_6t_40_200.txt";

@seznam=&list($adr);  #nacte prislusny adresar
&priznaky;
&arff_hlavicka;
&zapis_vektoru;
print STDERR "\nkonec";

####################nacteni priznaku
sub priznaky
{
  open(ATT,$priznaky);
  $att=join ' ',<ATT>;
  @slovnik=split(/\n/,$att);
  foreach (@slovnik)
  {
    s/ //gi;
    $x=$_;
    $slovnik{$x}=0;
  }
  close ATT;
}

####################vytvoreni *.arff a hlavicky
sub arff_hlavicka
{
  print STDERR "\narff hlavicka";
  open(ARFF,">$arff")
    or die print "Nelze vytvorit .arff!";
  $nazev=substr($arff,rindex($arff,'/')+1);
  print ARFF "\@RELATION $nazev\n\n";
#  foreach (sort keys %slovnik)
  foreach (@slovnik)
  {
    print ARFF "\@ATTRIBUTE $_ REAL\n";
  }
  if(not opendir(ADR,$adr))
  {
    print STDERR "\nNelze otevrit!";
```

```
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  splice(@soubory,0,2);
  $st="\@ATTRIBUTE #třída# {";

  foreach (@soubory)
  {
    $st.=$_.",";
  }
  $st=substr($st,0,rindex($st,',')).}";
  print ARFF "$st\n\n";
  print ARFF "\n\n\@DATA \n\n";
}

####################zapis vektoru
sub zapis_vektoru
{
  print STDERR "\nzapis vektoru";
  $i=0;
  foreach $data(@seznam)
  {
    if(-e $data)
    {
      open(F,$data);#nacteni
      $page=undef;
      $page=join ' ',<F>;
      close(F);
      $page="\L$page\E";
    }
    @text=split(/[\W_]+/,$page);

    %podslovnik=();
    foreach (@text)
    {
      if ($_=~/\d+/){next;}          #bez cisel
      if ($_!~/\w{2,}/){next;}     #ma alespon 2 znaky
      if (!exists($stoplist{$_}))   #porovnani se stoplistem
      {
        if (!exists($podslovnik{$_})) {$podslovnik{$_}=1;}#pocty a slova v dokumentu
        else {$podslovnik{$_}++;}
        $pod_df{$_}=0;
      }
    }
    foreach (keys %pod_df) {$df{$_}++;}     #df - pocet dokumentu, ve kterych se vyraz vyskytuje
    %pod_df=();
```

```
    $dok=0;     #pocet slov v dokumentu
    foreach (keys %podslovnik)
    {
      $dok+=$podslovnik{$_};    #pocet slov v dokumentu
    }

    $celk_dok=$#seznam+1;
    $vektor=undef;
    foreach (keys(%slovnik))
    {
      if (!exists($podslovnik{$_})) {$tfidf=0;}
      else
      {
        $tfidf=($podslovnik{$_}/$dok)*(log($celk_dok/$df{$_})/log(10)); #vypocet hodnoty TF-IDF
      }
      $vektor.="$tfidf,";               #string k zapsani do file
    }
    print ARFF $vektor;
    $tetema=$data;
    $tetema=~s/$adr//;
    $tetema=substr($tetema,1,index($tetema,'/',1)-1);
    print ARFF "$tetema\n\n";
    $i++;
    print STDERR "\n$i";
  }
  close ARFF;
}

####################seznam souboru
sub list
{
  my $adresar=shift;
  my (@soubory,$jmeno,$cesta);
  if(not opendir(ADR,$adresar))
  {
    print STDERR "\nNelze otevrit!";
    return 0;
  }
  @soubory=readdir(ADR);
  closedir(ADR);
  foreach $jmeno(sort @soubory)
  {
    if($jmeno=~/^\./){next;}   #tecka na zacatku, vynechani "o adresar vys"
    $cesta=$adresar."/".$jmeno;
    if(-d $cesta){&list($cesta);}
    if(-f $cesta){push(@seznam,$cesta);}
  }
```

```
  print STDERR "\nnacteni souboru";
  return @seznam;
}
```