Mendelova univerzita v Brně
Provozně ekonomická fakulta

Aplikování paralelismu při dolování znalostí z textových dat

Diplomová práce

Vedoucí práce:

doc. Ing. František Dařena, Ph.D.

Bc. Zdeněk Novák

Brno 2013

list originalniho zadani prace

Děkuji vedoucímu práce, doc. Ing. Františku Dařenovi, Ph.D., za vedení při zpracování diplomové práce a za poskytnutí nezbytných podkladů, pomoci a cenných rad. Děkuji také doc. Ing. Janu Žižkovi, CSc., za seznámení s problematikou text miningu a pomoci při vyprácování této práce.

Prohlašuji, že jsem tuto práci vypracoval zcela samostatně s použitím výhradně těch zdrojů, které jsou souhrnně uvedeny v seznamu literatury.

Brno, 23. května 2013 .....

Abstract

Zdeněk Novák, Application of parallel approach to text mining. Diploma thesis. Brno, 2013.

The diploma thesis deals with mining knowledge from large collections of textual data using parallelism as a possible solution of problems related to high computational complexity of this process. It describes issues associated with preprocessing of textual data and usage of machine learning algorithms. The outcome is represented by the results of experiments that compare a traditional approach to text classification with a parallel approach. For the purpose of realizing these experiments an application (using some popular libraries (sometimes modified) and software) was implemented. The applied classification algorithms include neural networks, support vector machines, and k-nearest neighbours. The results demonstrate that a parallel approach enables to reduce the time complexity of the process while maintaining sufficient values of classification performance metrics.

Keywords

text mining, machine learning, classification, parallelism, neural networks, support vector machines, k-nearest neighbours

Abstrakt

Zdeněk Novák, Aplikování paralelismu při dolování znalostí z textových dat. Diplomová práce. Brno, 2013.

Diplomová práce se zabývá problematikou dolování znalostí z rozsáhlých kolekcí textových dat a možným řešením tohoto výpočetně náročného procesu v podobě paralelizace. Popisuje problematiku spojenou s přípravou textových dat a jejich následným zpracováním algoritmy strojového učení. Výstupem jsou výsledky experimentů, které porovnávají tradiční sekvenční přístup klasifkiace dat s paralelním řešením. Pro účely realizace těchto experimentů byla implementována aplikace (s využitím známých knihoven (částečně upravených) a softwaru). Použité klasifikační algoritmy zahrnují neuronové sítě, podpůrné vektory a k-nejbližších sousedů. Výsledky ukazují, že použití paralelního přístupu umožňuje snížení časové náročnosti procesu se současným zachováním dostatečných hodnot metrik pro zhodnocení přesnosti klasifikace.

Klíčová slova

dolování znalostí z textových dat, strojové učení, klasifikace, paralelismus, neuronové sítě, porpůrné vektory, k-nejbližších sousedů

Obsah

1 Úvod a cíl práce

1.1 Úvod

Již několik desítek let jsou známé modely a navržené inteligentní algoritmy, jejichž použití ale nebylo do nedávné doby možné. Kupříkladu model umělého neuronu byl poprvé představen na začátku druhé poloviny minulého století (Rosenblatt, 1957), ale jeho praktická realizace nebyla v dané době kvůli neodstatečné výpočetní technice možná. Teprve v nedávné době umožnilo neustálé zvyšování výpočetních možností jeho reálné využití a obecně rozvoj umělé inteligence, která si klade za cíl stvořit inteligentní program či stroj. Jde o velice zajímavý obor v oblasti počítačové vědy, který přitahuje značné množství lidí. Často jsou však současné možnosti stále omezeny dostupnou výpočetní kapacitou, a proto některá úsilí směřují k hledání způsobů, jak tyto nároky na počítačovou techniku snížit. Jednou z mnoha oblastí umělé inteligence je dolování znalostí z dat, jehož princip spočívá v prohledávání velkého množství sebraných údajů s cílem odhalit v těchto datech nové, skryté informace. Zvlášťním případem je dolování znalostí z textových dat. S reálnými aplikacemi z této oblasti se většina lidí setkává každý den, např. při vyhledávání přes celosvětový vyhledávač *Google*. Jde tedy o velice zajímavou oblast hromadného zpracování dat, protože díky dnešnímu rozšíření internetu a možnostech, které nejrůznější webové aplikace nabízejí, je k dispozici obrovské množství textových dat. Některé aplikace nemusí díky malému množství dat dostupných ke zpracování dosahovat kvalitních výsledků a jedním ze způsobů, jak výsledky vylepšit, je právě větší množství dat (Banko, Brill, 2001). S množstvím dat není u dolování z textů problém, ale narozdíl od klasického dolování jsou nároky na výpočetní výkon vyšší. Může za to právě povaha textových dat. Důsledkem je tedy nemožnost zpracovávat v rozumném čase velká množství textových dat, přestože jsou dostupná a mohla by tak být nějakým způsobem využita. Možným řešením, kterým se zabývá také tato diplomová práce, je aplikování paralelního přístupu. Ten umožňuje snížit výpočetní nároky kladené na počítačovou techniku při zpracování rozsáhlých kolekcí textových dokumentů (Žižka, Dařena, 2012). Především jde o zjištění možností nasazení konkrétních algoritmů při paralelním zpracování a jaký bude mít tento přístup dopad na jejich schopnost poskytovat správné výsledky v rozumném čase.

1.2 Cíl práce

Cílem práce je prozkoumat možnosti aplikování paralelního přístupu při dolování znalostí z rozsáhlých kolekcí textových dat. Hlavní snažení bude směřovat na porovnání vlivu tohoto přístupu na přesnost a výpočetní náročnost různých algoritmů strojového učení, které jsou používány pro klasifikaci textových dat. Pro tyto účely bude nutné realizovat sadu experimentů, které poskytnou výsledky vhodné pro tato srovnání. V práci bude navržen formát těchto experimentů a získané výsledky vyhodnoceny. Pro jejich realizaci bude také implementována aplikace.

2 Současný stav

Umělá inteligence je v dnešní době samostatným vědním oborem, který si od svého vzniku v roce 1987 prošel obdobími velikého nadšení a vkládání mnoha nadějí i neutuchající skepse a odmítání. Samotné základy položil v roce 1956 na univerzitě v Princetonu John McCarthy, a to uspořádáním první mezinárodní konference, jejímž tématem byla právě umělá inteligence. Za své přispění na poli tohoto nového vědního oboru pak v roce 1985 obdržel Turingovu cenu (AI Basics). Mezi účastníky této konference byl i Marvin Minsky, který je také považován za jednoho z otců moderní umělé inteligence. On sám ji pak definuje jako vědu, jejímž cílem je tvorba strojů na dělání věcí, pro jejichž vykonávání je vyžadována určitá inteligence, provádí-li je člověk (Whitby, 1996). Toto samozřejmě není jediná z definic, na které lze při nahlédnutí do různých učebnic, vědeckých článků či internetu narazit. Dokonce je možné se na umělou inteligenci dívat nikoliv jako na vědní obor, nýbrž jako na určitou vlastnost umělého (ve smyslu neživého) objektu, která mu umožňuje se inteligentně rozhodovat nejen na základě vjemů, které získává ze svého okolí, ale také na základě vlastních znalostí, které má uchovány ve své paměti. Samotný pojem inteligentní rozhodování je však poměrně vágní. Intuitivně si pod tímto lze představit například to, že inteligentně se rozhodující stroj by měl docházet k podobným závěrům, ke kterým by za stejných okolností dospěl člověk.

Definitivní rozhodnutí o tom, zda je stroj inteligentní nebo nikoliv, je možné učinit na základě tzv. Turingova testu, pojmenovaném po britském matematikovi Alanu Turingovi. Ten se ve svém článku *Computing machinery and intelligence* (1950) zaobíral otázkou, zda je možné uvažovat nad tím, jestli mohou stroje myslet. Místo pokusů o její zodpovězení, kvůli vícevýznamnosti slov *stroj* a *myslet,* navrhl jiný přístup k formulaci tohoto problému, a to ve formě hry. Této se účastní celkem tři osoby: muž, žena a tazatel. Tazatel zůstává v oddělené místnosti a oběma osobám pokládá otázky, přičemž jejich odpovědi pak dostává v tištěné podobě. Úkolem tazatele je rozpoznat, kdo je muž a kdo žena, a oba dva se snaží tazatele přesvědčit, že jsou žena, tzn. že muž záměrně lže, zatímco žena říká pravdu. K tomuto je však třeba určitým způsobem myslet a inteligentně odpovídat. Muž je poté v průběhu hry nahrazen umělou inteligencí a pokud stroj zvládne svou úlohu a obelže tazatele, tak tímto testem prošel. Jeho chování tudíž lze označit za inteligentní. Dodnes se to však žádnému stroji zcela nepodařilo. Nejblíže tomu byl program Josepha Weizenbauma (1966),[1] který je i některými lidmi považován za první, který testem prošel. Princip spočívá v jednoduchém hledání klíčových slov v průběhu konverzace, na základě kterých pak sestavuje svoje odpovědi a nové otázky. Pokud není k dispozici dostatek informací pro sestavení odpovědi, program opakuje dřívější výroky nebo používá obecné odpovědi – tento postup je založen na tzv. rogeriánské psychoterapii. Díky těmto principům program dokázal tazatele v Turingově testu přesvědčit o tom, že je člověk.

[1] dnes jsou podobné programy známé pod názvem chatterbot

Umělá inteligence – jako vědní obor – v dnešní době v podstatě zastřešuje celou řadu různých přístupů řešení této problematiky. Důvodem je to, že narozdíl od konkrétních implementací umělé inteligence hrající například šachy, tak vytvoření obecné umělé inteligence, která by svými kvalitami byla alespoň srovnatelná s lidským myšlením, se velice brzo ukázalo jako téměř neřešitelný problém. Proto se současná snaha zaměřuje zejména na hledání dílčích řešení. Technologie a přístupy, které se se pro tyto účely využívají jsou (Russell, Norvig, 1995):

- strojové učení,
- expertní systémy,
- fuzzy logika,
- neuronové sítě,
- genetické programování,
- procházení stavového prostoru,
- dolování z dat.

Expertní systémy jsou jedním z typů tzv. znalostních systémů, což jsou ve své podstatě programové systémy, které mají v nějaké konkrétní podobě uloženy znalosti z určité problémové oblasti a využívají je k hledání řešení na problémy spadající do stejné oblasti (Konečný, Trenz, 2010). Expertní systémy se však odlišují tím, že dokáží nalezené řešení zdůvodnit. Důvodem jejich tvorby je především nedostatek expertů, jejichž služby bývají jednak drahé, ale také nemusí být kvůli časové vytíženosti experta vůbec dostupné. Vytvořením systému jsou pak znalosti snadno kdykoliv dostupné, jeho údržba je levnější, zaručuje určitou úroveň bezchybnosti a uchovává data po neomezeně dlouhou dobu. Asi největší problém však spočívá právě v získání potřebných znalostí od expertů. Ti musí formulovat tzv. rozhodovací pravidla, která jsou následně převedena do podoby srozumitelné pro expertní systém. Princip fungování systému spočívá v použití inferenčního (odvozovacího) mechanismu, který na základě báze znalostí sestávající z faktů (atomická pravdivá tvrzení) a pravidel, metodou zpětného řetezení odvodí jednotlivé podcíle a pořadí jejich řešení.

Cílem fuzzy logiky je umožnit práci s tzv. vágními pojmy, což jsou hodnoty vyjádřené slovně nikoliv konkrétní číselnou hodnotou. Své uplatnění nalezla ve fuzzy expertních systémech, u kterých jsou pravidla formulována slovními hodnotami (Konečný, Trenz, 2010). Tyto hodnoty zastupují určitý interval číselných hodnot a zároveň definují i jistotu, s jakou konkrétní prvek nebo hodnota do daného intervalu patří či ne. Jiným označením pro tyto intervaly hodnot jsou fuzzy množiny.

Genetické programování je založené na principech probíhajících v přírodě na úrovni biologické evoluce. Zakladní myšlenka je inspirována Darwinovou teorií o vývoji živočišných druhů, tedy že zjednodušeně přežijí jen ti nejsilnější, potažmo nejlepší, jedinci. Princip spočívá ve vygenerování populace nějakých řešení pro určitý

problém. Důležité je, aby počáteční populace byla dostatečně početná, jinak by mohlo dojít k degenerování řešení. Na členy populace jsou pak v každé iteraci, představujicí jednu generaci populace, aplikovány tři základní operace, a to křížení, mutace a výběr nejlepších jedinců (Koza, 1990). Je tedy nutné nějakým způsobem zajistit srovnatelnost jednotlivých členů populace. Ti jsou reprezentováni chromozomy skládající se z určitého počtu genů. Kvalitu jedince pak určuje tzv. funkce přizpůsobivosti.[2] Lépe ohodnocení jedinci mají pak vyšší šanci, že budou vybráni do procesu křížení, a tím případně přenesou alespoň část své kvality na potomky. Hůře ohodnocení jedinci nejsou z výběru zcela vyřazeni, ale jejich šance na participaci jsou výrazně menší. Proces mutace, představující náhodnou změnu genů jedince, je do jednotlivých generací populace zaveden kvůli tomu, aby dřív nebo později nedocházelo k degeneraci, což odpovídá uvíznutí v lokálním extrému. Ideální řešení se tedy naproti tomu nachází v globálním extrému. Výhodou genetických algoritmů je, že se při hledání řešení nemusí vyhýbat lokálním extrémům, narozdíl od mnoha jiných algoritmů jako neuronové sítě, které v lokálním extrému snadno uvíznou. Někteří jedinci populace naopak obsadí tyto extrémy a jejich potomci pak budou představovat řešení jiná. Genetické algoritmy lze použít na řešení celé řady problémů, kupříkladu aproximace funkce, problém obchodního cestujícího nebo i generování zdrojového kódu.

Procházení stavového prostoru je jednou z důležitých částí umělé inteligence, jelikož představuje jeden ze základních úkonů, které je potřeba řešit. Ani se současným výpočetním výkonem není možné, aby programy prohledávaly všechny možné kombinace hodnot jednotlivých proměnných, dokud nenarazí na správné řešení. Z tohoto důvodu je nutné hledat řešení inteligentnějším způsobem, a to pomocí stavového prostoru. Jednotlivé kombinace hodnot proměnných představují konkrétní stavy, přičemž existují dva význačné stavy, a to počáteční, odkud začíná proces prohledávání stavového prostoru, a stav koncový, který vyhovuje podmínkám na požadované řešení problému. Aby bylo možné přecházet mezi jednotlivými stavy, tak jsou zavedeny tzv. operace, jejichž realizací se stroj pohybuje stavovým prostorem a prohledává dostupné stavy. Celý statový prostor je možné znázornit pomocí orientovaného grafu, ve kterém uzly představují stavy a hrany operace. Zpravidla se stavový prostor nikdy neuchovává v paměti stroje celý, protože pro komplexnejší typ úloh to zkrátka není možné. Místo toho se následující stavy generují na základě aktuálního stavu a operací, které je možné z tohoto stavu realizovat. Stroj si pak vybírá, do kterého stavu přejde na základě hodnotící funkce, která stanoví, jaký užitek[3] daný stav přinese. Typickým příkladem úlohy, jejíž řešení je založené právě na procházení stavového prostoru, je hra *misionáři a kanibalové*. Ta spočívá v přepravení tří kanibalů a tří misionářů na druhý břeh pomocí loďky, která uveze maximálně dvě osoby. Důležité však je aby, na jednom z břehů nebyli kanibalové v početní převaze. Stavy zde představují pozici loďky a počty kanibalů a misionářů na obou březích. Operace pak jsou možné převozy z jednoho břehu na druhý. Zde

[2] v originále fitness function

[3] programy pracující tímto způsobem jsou označovány jako užitkově zaměření agenti

je stavový prostor natolik malý (pokud ukládáme pouze unikátní stavy), že nic nebrání tomu, najít řešení prohledáním všech možných stavů. Oproti tomu hra v šachy tento přístup již neumožňuje. Stavy zde představují rozestavení figurek na šachovnici a operace možné tahy. Odhaduje se, že pokud by byl pro uložení každého stavu použit atom ve vesmíru, pak stejně nebude možné uchovat všechny možné stavy. Počítačový hráč tedy může zjednodušeně fungovat tak, že nějakým způsobem hodnotí kvalitu, kterou mi přenesou jednotlivé tahy, a rozhoduje se na základě procházení pouze části stavového prostoru. Náročnost počítačového hráče pak může být určena počtem budoucích tahů, které může při svém rozhodování prohledat.

Dolování z dat nelze zcela jednoznačně označit jako součást umělé inteligence, ale v podstatě s ní úzce souvisí. V procesu dolování z dat jsou totiž velice často používány právě technologie a přístupy známé z umělé inteligence, potažmo strojového učení. Samotné dolování je tedy zaměřeno na hledání nebo odkrývání znalostí, které mohou být v datech skrytě obsažené. V knize *Data Mining: Practical Machine Learning Tools and Techniques* (Witten, Eibe, 2005) je dolování z dat definováno jako proces odhalování vzorů v datech, přičemž proces samotný musí být automatický nebo alespoň poloautomatický a odhalené vzory musí být natolik smysluplné, aby dokázaly poskytnout určitou výhodou, zpravidla ekonomickou.

Tato definice zohledňuje i ekonomický rozměr získaných výsledků. Ve firmením světě je totiž dolování již poměrně běžnou praxí, a to v rámci business inteligence aplikací, kdy se z dat, které firma získala například od svých zákazníků, mohou zjišťovat nové skutečnosti, členit zákazníky do různých skupin podle jejich preferencí nebo vzorů v jejich nákupním chování. Při tomto přístupu se zpravidla využívají všechny možná dostupná data, která má firma k dispozici. Tato data jsou strukturovaná a převážně vyjadřují nějaké číslené hodnoty, například celkové peníze utracené zákazníkem za určité skupiny produktů, četnost jeho nákupů, počty nakoupených položek v dílčích nákupech nebo příslušnost produktu do určité kategorie vyjádřená číselným identifikátorem. Část dat však zůstává v rámci klasického dolování z dat nevyužita a je tedy nutné hledat jiné přístupy či metody, které by tuto mezeru dokázaly zaplnit. Jednou takovou zvláštní oblastí je pak dolování z textových dat, které se běžně označuje jako *text mining,* případně text data mining. Téma této práce spadá právě do oblasti dolování z textových dat, a proto bude několik kapitol věnováno bližšímu představení současných postupů používaných při text miningu.

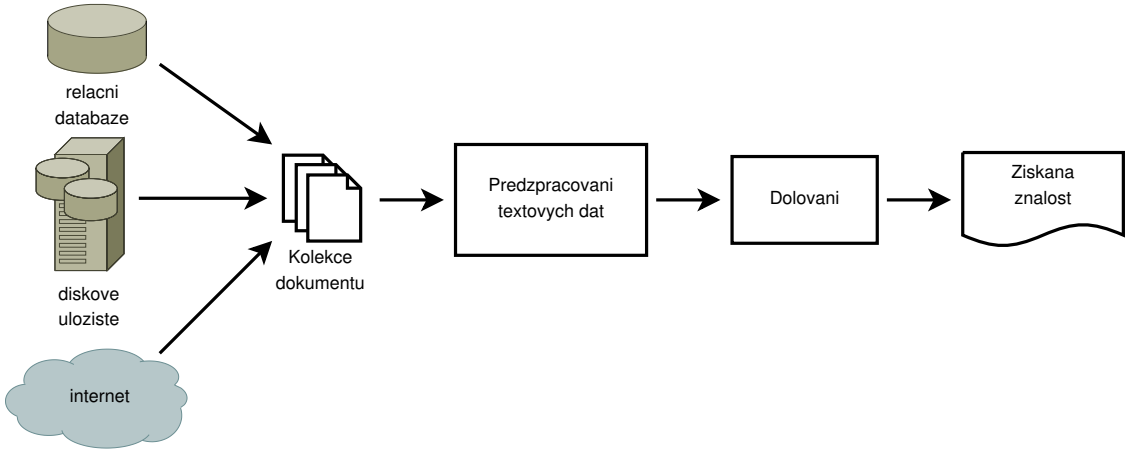
Účelem tohoto obecného úvodu bylo alespoň velice povrchově čtenáři nastínit historický vývoj v oblasti inteligentních algoritmů, v jakém kontextu se současné technologie nachází, k čemu a jak se využívají a především poskytnout přibližnou představu o postavení text miningu mezi těmito přístupy. Další kapitoly se již čistě zaměří právě na problematiku dolování z textových dat.

2.1 Text mining

Text mining, neboli též dolování z textových dat, je chápán jako proces odvozování vysoce kvalitních informací z textu, přičemž vysoká kvalita je chápána jako kombinace významnosti, novoty a zajímavosti nově získaných informací (Konchady, 2006). Své uplatnění nalézá v řadě různých oblastí jako je bioinformatika, webové služby, řízení vztahu se zákazníkem nebo pro akademické účely (Text Mining Science, 2012). Jak již bylo zmíněno, jde o určitou odnož data miningu, která se zaměřuje čistě na textová data. Toto je hlavní rozdíl oproti klasickým přístupům dolování z dat, které vyžadují dobře strukturovaná data. Neznamená to však, že jde o naprosto odlišné oblasti, opak je pravdou. Při práci s textovými daty jsou totiž velice často používány tytéž postupy a metody jako při zpracování strukturovaných dat. Aby to však bylo možné, je nutné textová data vhodně předzpracovat a převést je do podoby požadované těmito metodami, tedy do číselné reprezentace.

Pokud se na data pro dolování člověk podívá jako na tabulku hodnot, známou například z programu Microsoft Excel, jednotlivé řádky představují konkrétní instance dříve zaznamenaných případů. Každá buňka tabulky pak určuje jeden atribut, který má jasně definovanou svou doménu. Jeden ze základních problémů, který se musí při práci se strukturovanými daty řešit, je právě nutnost dodržení příslušnosti hodnot každého atributu do odpovídající domény, a to samozřejmě u všech uchovávaných záznamů. Stejně tak problematické mohou být (a zpravidla také jsou) neúplné záznamy, kdy některé atributy nemají zadány svou hodnotu. Oba dva je třeba nějakým způsobem řešit. Většinou se to děje v rámci přípravy dat pro samotné dolování, kdy neúplné záznamy mohou být například zcela vyřazeny nebo doplněny průměrnými hodnotami. Při práci s texty není nutné těmto komplikacím čelit. Je to dáno tím, že textová data představují tzv. kolekci dokumentů a každý jednotlivý dokument se bere jako jedna instance, tedy řádek v tabulce. Jako atributy dokumentů jsou pak používány všechna slova, která se v rámci celé kolekce dokumentů objeví. Tím pádem odpadá problém s neuplnými záznamy, jelikož slovo se vždy v daném dokumentu buď vyskytuje nebo nevyskytuje. Stejně tak nejsou komplikace se sjednocováním domén jednotlivých atributů napříč všemi záznamy, protože informace o výskytu slova se vždy vyjadřuje stejnou hodnotou. Zde je navíc možné vypozorovat další zajímavou charakteristiku textových dat. Hodnoty jsou ve celé tabulce pouze kladné (Konchady, 2006), opět narozdíl od dat, se kterými se pracuje při klasickém dolování, kde například stav účtu může nabývat i záporných hodnot. I přes tyto určité výhody, které s sebou přináší práce s textovými daty, stále je samotný proces předzpracování textů často mnohem náročnější, než je tomu u strukturovaných dat (Neto, Santos, Kaestner, Freitas, 2000). Jako další úskalí textů a práce s nimi je uváděna například už samotná abstrakce či nejednoznačnost, která se za textem skrývá. To v podstatě souvisí s dalším problémem, který skýtá různorodost vyjadřování se pomocí přirozeného jazyka, kdy stejnou věc je možné pojmenovat různými slovy se stejným nebo velice podobným významem. Z hlediska počítače jsou však tato synonyma automaticky brána jako různé atributy.

Stejně tak zhoršují řešení problému homonyma, což jsou naopak mnohovýznamová slova. Příkladem může být české slovo měsíc, které lze chápat jako měsíc kalendářní či jako vesmírné těleso. Člověk snadno pochopí význam slova na základě kontextu, v jakém se slovo nachází, a rozpoznal by jej tedy jako dva odlišné atributy dokumentu, počítač už nikoliv. Základní pracovní postup při dolování z textových dat je



Obrázek 1: Schéma postupu při dolování z textových dat

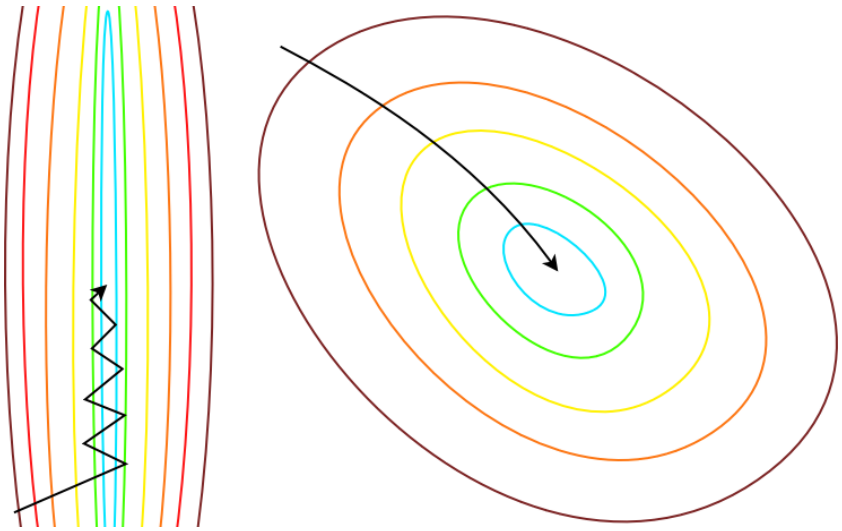
schématicky znázorněn na obrázku 1. Prvním krokem je získání kolekce dokumentů. Zdrojem mohou být relační databáze, internetové stránky, elektronické články atd. Po ukončení sběru dat je třeba veškeré dokumenty předzpracovat dle požadavků technologií a algoritmů, které budou použity při samotném dolování. Tento proces může být poměrně přímočarý a jednoduchý, kdy dojde pouze k transformaci nestrukturovaných textů do strukturované podoby (jak bylo ve zkratce popsáno dříve). Samozřejmě je také možné zvolit mnohem sofistikovanější přístup, kdy budou odstraněna některá slova či celé dokumenty, provedeny různé výpočty pro transformaci číselných hodnot charakterizující jednotlivé dokumenty, obohacování či naopak zmenšování dokumentů apod. Nejčastěji používané přístupy pro tuto fázi procesu budou popsány v následujících kapitolách. Posledním krokem pak už je realizace samotného dolování, jehož výstupem by měla být určitá netriviální znalost, například naučený klasifikátor nebo odhalení společných vzorů mezi jednotlivými dokumenty.

Nově získanou znalost je většinou potřebné nějakým způsobem zhodnotit. Ke změření kvality znalosti se používá několik různých způsobů, které budou také blíže představe ve zvláštní kapitole. Ve zkratce probíhá proces zhodnocení kvality znalosti tak, že se vytvoří tzv. testovací množina dokumentů a znalost je na této množině otestována. Z výsledků testu jsou následně vypočítány různé metriky přesnosti, které pomáhají zhodnotit kvalitu odhalené znalosti (Manning, Raghavan, Schütze, 2008).

2.2 Reprezentace dokumentů

Textová data je nezbytné transformovat do podoby, která je vhodná pro algoritmus používaný ve fázi dolování. Tato část procesu zpracování textů je velice důležitá

a zvolené postupy při transformaci dat do strukturované podoby mají značný vliv na výslednou kvalitu získaných znalostí. Příkladem, na kterém lze ilustrovat dopad nevhodně zvolené podoby vstupních dat na výsledek a samotný průběh hledání výsledku, může být algoritmus *slézání kopce* (anglicky gradient descent). Pro jednoduchost můžeme uvažovat, že použijeme pouze data o dvou atributech, přičemž jejich hodnoty se nacházejí v rozdílných intervalech (např. 1–5 a 1800–2400). Tyto hodnoty slouží jako vstup pro hodnotící funkci, která přiřadí dané dvojici hodnot konkrétní cenu. Cenové ohodnocení se vypočítá pro určité množství kombinací hodnot v daných rozsahech. Pokud tato data zobrazíme pomocí konturového grafu, budou mít jednotlivé izolinie tvar velice úzkých elips (Ng, 2012). Pro algoritmus bude v takovémto případě mnohem obtížnější nalézt cestu do hledaného minima. Pokud jsou však hodnoty přepočítány do shodného měřítka (rozsah hodnot bude pak pro oba atributy stejný), dostanou izolinie přibližně kruhový tvar a algoritmus zkonverguje mnohem rychleji. Tento jev je ilustračně znázorněn na obrázku 2. Podobným způsobem se lze dívat na přípravu textových dat. Nevhodná reprezentace může negativně ovlivnit proces dolování a získáné výsledky nemusí dosahovat dostatečné kvality. Proto je důležité věnovat přípravě dostatečnou pozornost a snažit se o nalezení co nejvhodnější reprezentace dat.



Obrázek 2: Zjednodušená ilustrace vlivu normalizace dat na průběh algoritmu gradient descent. Levý konturový graf znázorňuje výrazně zúžené izolinie, které jsou typické pro hodnoty atributů z výrazně odlišných intervalů. Pravý graf znázorňuje možnou podobu izolinií po normalizaci hodnot do shodných intervalů.

Aby bylo možné jednotlivé textové dokumenty transformovat do strukturované podoby, je třeba určit nějakou základní stavební jednotku, pomocí které bude možné dokument vyjádřit. Přirozeným a zároveň nejpoužívanějším řešením je použití slov, která jsou dostatečně srozumitelná. Výhodou je navíc i fakt, že zpracování textů do tohoto způsobu reprezentace je poměrně jednoduchá. Není to však jediný přístup a stejně jako při klasickém dolování, i zde je možné vytvářet z datových složek

složky nové. Pokud data obsahují např. údaje o rozloze domu a jeho prodejní ceně, je možné na základě těchto hodnot vypočítat novou složku, a to cenu za metr čtvereční. Ta pak může teoreticky pomoci při samotném dolování, kupříkladu by mohla tato informace dobře posloužit při shlukování. Obdobně je možné rozšiřovat možnosti pro strukturovaný popis textových dat.

Model bag-of-words

Základním přístupem je tzv. *bag of words* reprezentace a použití nalézá nejen při práci s textovými dokumenty, ale také s obrazovými daty nebo s video daty. Každý dokument je reprezentován skupinou slov. Tato množina se také nazývá slovník. Nejjednodušší reprezentací je vyjádření dokumentů pomocí příznaku výskytu slova. Mnohem častěji se však místo této binární hodnoty (slovo se v dokumentu nachází nebo nenachází) používá počet výskytu daného slova v dokumentu. Pak je možné dívat se na reprezentaci pomocí bag-of-words jako na histogram slov pro konkrétní dokument. Důležité je, že každý dokument, jakožto uspořádaná struktura slov, je transformován na neuspořádánou množinu slov, čímž dochází ke ztrátě určitého množství informace. V podstatě se naivně předpokládá, že výskyty jednotlivých slov jsou na sobě vzájemně nezávislé a není tedy nutné uchovávat informaci o pořadí jednotlivých slov (Zhang, Jin, Zhou, 2012).

Obdobným způsobem je bag-of-words uplatňováno i pro obrazová data, kde jsou slova nahrazena tzv. *vizuálními slovy,* což jsou malé části obrazu, které bývají také označovány jako příznaky (angl. features). Je tedy možné se setkat i s označením *bag-of-features.*

Základní podoba bag-of-words, ve které jsou použita pouze slova z dokumentů v původním tvaru, je schopna poskytnout poměrně dobré výsledky (Zhang, Jin, Zhou, 2012). Často je však využíváno různých možností, jak reprezentaci dokumentu pomocí bag-of-words zlepšit a dosáhnout tak kvalitnějších výsledků. Těchto postupů existuje nemalé množství, a tak se může proces transformace textu velice snadno skládat hned z několika dílčích kroků. Zároveň je také třeba brát v potaz různé problémy, které je třeba při transformaci textu řešit.

Tokenizace

Tokenizace je proces rozdělení textu na jednotlivá slova a jde o důležitou část procesu zpracování textu. (Manning, Raghavan, Schütze, 2008) Text dokumentu je třeba rozdělit na tzv. *tokeny,* což jsou instance prvků ze slovníku. Ačkoliv by se mohlo řešení jevit jako poměrně jednoduché a přímočaré, tak i zde je možné narazit na několik problémů. Pokud se pracuje kupříkladu s doslovným přepisem řeči nějakého člověka, může tento text obsahovat nedořečená slova nebo neartikulované zvuky používané při pauzách mezi řečí. Budou i tyto řetězce považovány za slova či nikoliv? Další problém mohou způsobit obecně víceslovné názvy např. měst nebo institucí. Budou *Královo pole* nebo *ústavní soud* brány jako jeden token nebo jak tokeny dva?

Různé jazyky mohou navíc díky svým vlastním charakteristikám přinést další aspekty, které může být vhodné nějakým způsobem vyřešit. V angličtině se například mohou objevit tzv. *hovorové stažené tvary.* Jde o zkrácený zápis (nejčastěji podmětu a přísudku). Při zpracování je třeba rozhodnout, jakým způsobem se bude zacházat s podobnými zápisy. Zda-li se apostrof nahradí za mezeru a vzniknou tak nekoreketní slova, nebo se vše ponechá v původním tvaru, případně se zápis rozšíří na kompletní slova.

```
I'm     --->  I am
You're  --->  You are
don't   --->  do not
```

Dalším zajímavým příkladem může být japonština, pro kterou je charakterisické psaní slov bez oddělení mezerou. Při zpracování japonsky psaného textu je tedy nutné nějakým způsobem rozdělit psanou větu jednotlivé tokeny. Pro tyto účely dobře slouží *longest fit* algoritmus (funguje na hladovém principu), který postupně prochází text znak po znaku a hledá nejdelší možnou shodu se některým ze slov ve slovníku. Využívá se zde další zajímavé charakteristiky japonštiny, a to průměrné délky slova, která je 2.4 znaku. Tento poměrně jednoduchý přístup však funguje velice dobře. V japonsky psaném textu je ještě možné narazit na celkem tři typy abeced, z nichž každá slouží pro jiný účel.

- Kanji – používá se pro slova propůjčená z Číny, vždy jde o obsahová slova jako podstatná jména, přídavná jména nebo slovesa
- Hiragana – používá se pro funkční slova, gramatické značky, skloňovací koncovky a některá japonská slova nepsaná v Kanji
- Katakana – tato abeceda se používá pro slova vypůjčená z cizích jazyků (kromě čínštiny)

Některá slova mohou být navíc psána jako kombinace více abeced. Také se pro technické převzaté názvy bez vlastního překladu používá klasická latinka (Fung, Kan, Horita, 1996). V japonštině se navíc (narozdíl od angličtiny) vyskytuje velké množství homonym a synonym, což opět vyžaduje věnovat zvýšenou pozornost při zpracování a reprezentovat různá slova se stejným významem jedním tokenem nebo stejná slova s různým významem více tokeny.

Jak je vidět na několika málo příkladech, ačkoliv se může zdát samotné rozdělení textu na slova velice snadnou úlohou, pro získání opravdu kvalitní reprezentace dokumentu je nutné ošetřit řadu mnohoznačností a naopak nejednoznačností textu psaného v přirozeném jazyce. Různé jazyky pak mohou navíc celý proces ještě zkomplikovat svými charakteristickým vlastnostmi, na které je také třeba brát při zpracování zřetel.

Normalizace

Stejně jako v případě klasického dolování, je i u textových dat nutné provést určitou míru normalizace. Jedná se o proces, ve kterém dochází ke sjednocení způsobu reprezentace určité informace (Manning, Raghavan, Schütze, 2008), a to napříč celou datovou základnou. Zatímco u strukturovaných dat se během normalizce jednotným způsobem vyjadřují například data, peněžní položky, adresy nebo jména, u textových dat dochází k normalizaci zápisu slov se stejným významem, ale rozdílným zápisem. Typickým příkladem mohou být různé zkratky nebo názvy.

```
Narodil jsem se v České republice.
Po vystudování jsem se rozhodl odcestovat z ČR do zahraničí.
Let Č.R.--U.S.A. bude odlétat za 15 minut.
```

Ve výše uvedených větách se vyskytují tři různá označení pro Českou republiku. Během normalizce by tato slova měla být rozpoznána a nadále již vyjádřena vždy stejným jedním slovem. Tím dojde jednak ke zmenšení velikosti slovníku, ale především budou slova vyjadřující naprosto shodnou informaci zápsána stejným způsobem, což by mělo mít pozitivní dopad na algoritmy použité pro samotné dolování.

Case folding

Volně psaný text bývá velice často *neuhlazený.* Jelikož bývají zpracovávaná data většinou kolekcí dokumentů z různých zdrojů, tak je velice pravděpodobné, že jednotlivé dokumenty pocházejí od různých lidí, s různými návyky při psaní. Stejná slova tak mohou být zapsána různě a jedním z rozdílů může být použitá forma písma. V textových datech se tak vyskytují slova psaná jak minuskami, tak i verzálkami. Některé texty mohou být napsány celé pomocí minusek, jinde mohou být některá slova nebo i celé texty psané naopak pouze verzálkami, někdy se vyskytnou i slova obsahující písmena psaná různou formou zcela nahodile (někdy označováno jako mixed-case). Pokud by texty zůstaly ve své původní podobě, mohou být stejná slova se stejným významem vyjádřena hned několika unikátními prvky slovníku, čemuž se snaží case folding zabránit. Jde ve své podstatě o poměrně konkrétní případ předcházející normalizace, protože dochází k sjednocování použité formy písma. Nejjednodušším přístupem je celou množinu dokumentů sjednotit buď do lower-case nebo upper-case podoby. Tím však může dojít i sjednocení způsobu zápisu slov, která vyjadřují různé věci a měly by tedy zůstat rozlišitelné.

```
Jack Black (jméno herce) ---> jack black (překlad: hever černý)
F.E.D.     (jméno úřadu) ---> fed        (překlad: nakrmený)
```

Příkladem mohou být nejrůznější názvy nebo jména, která bývají často odlišena od klasických slov právě pomocí formy písma. Je možné použít heuristický přístup, kdy ke změně formy písma dojde pouze u nadpisů nebo začátků vět a slova uvnitř vět jsou ponechána v původním tvaru. Ve většině případů by tento postup

měl dosáhnout poměrně dobrých výsledků, ale na první pohled je vidět, že rozhodně nejde o bezchybný přístup. Pokud budou jména nebo názvy uvedeny na začátku vět nebo v nadpisech, tak dojde k jejich chybnému přepsání do lower-case podoby a spoléhá se zde na správné používání formy písma samotnými autory textu. Z tohoto důvodu bývá transformace veškerého textu do lower-case nejefektivnější (Manning, Raghavan, Schütze, 2008).

Oprava pravopisu

Oprava pravopisu je dnes již poměrně rozšířenou součástí celé řady programů nebo aplikací, které vetšina lidí běžně využívá. Na kontrolu pravopisu je možné narazit při vyhledávání pomocí internetového vyhledávače Google, který podtržením zvýrazní nesprávně napsaná slova v dotazu a samotné vyhledávání provede pro dotaz s opravami. Textový procesor MS Word, který je součástí kancelářského balíku od firmy Microsoft, podobným způsobem zvýrazňuje nesprávná slova, případně překlepy přímo opravuje. A stejně tak se oprava pravopisu nachází i v chytrých telefonech.

V rámci dolování z textových dat je motivací pro použití korekce pravopisu zredukování velikosti slovníku, kdy nejrůznější překlepy mohou vytvořit celou řadu neplatných slov, která budou mít nízkou frekvenci výskytu (Dařena, 2011) a může tak dojít ke snížení kvality získané znalosti.

Celý proces opravy pravopisu je možné rozdělit na dvě úlohy, a to nalezení chyby a její následná oprava (Manning, Raghavan, Schütze, 2008). Jednotlivé chyby v textových datech lze také rozdělit do dvou kategorií. První jsou chyby, jejichž důsledkem vznikají nová neplatná (smyšlená) slova.

```
graffe  --->    giraffe
```

Nejjednodušším přístupem pro odhalení podobných chyb je použití rozsáhlého slovníku. Pro provedení opravy je vytvořen seznam možných kandidátů a vybrán je ten nejpravděpodobnější. Pravděpodobnost je možné určit na základě Levenshteinovi vzdálenosti (označováno také jako minimal edit distance) nebo na základě tzv. *noisy channel* modelu (Kernignan, Church, Gale, 1990).

Druhou skupinou jsou chyby, u kterých dojde k záměně jednoho slova za slovo jiné. To se může stát u tzv. homofonních slov, která poslechově znějí totožně, ale mají různý zápis, nebo pouhým prohozením písmen nebo záměnou písmen.

```
too     --->    two
beer    --->    bear
table   --->    label
fee     --->    free
```

U této skupiny chyb je jejich oprava velice podobná, ale o něco komplikovanější, protože nelze jednoznačně určit, které slovo bylo zapsáno chybně. Pro každé slovo je tedy vygenerován seznam kandidátů, do kterého je zahrnuto jednak slovo sa-

motné, všechna korektní slova, která lze získat vložením, smazáním nebo substitucí jednoho písmene. Je možné zahrnout i slova homofonní. Výběr správného (nejpravděpodobnějšího) slova pak probíhá opět na základě noisy channel modelu.

Lemmatizace

Lemmatizace je proces, při kterém dochází k převedení slova do základního (též slovníkového nebo kanonického) tvaru, tzv. *lemma.* Lemmatizace probíhá na základě morfologické analýzy, pomocí které se určí, do jakého konkrétního tvaru má být určité slovo převedeno. Během analýzy se tedy nepracuje pouze s jedním slovem, ale je zahrnuta větší část textu, aby bylo možné správně pochopit význam slova, a tím pádem určit i jeho základní tvar. Slova se totiž ve většině jazyků používají buď v odvozených nebo skloňovaných tvarech. Tyto tvary slov mohou mít například shodný zápis s jinými slovy, která však mají odlišný význam (Manning, Raghavan, Schütze, 2008).

```
I saw your saw in garage. ---> i see your saw in garage
```

Příkladem může být tvar nepravidelného anglického slovesa see (vidět) ve tvaru pro minulý čas saw, který by měl být přepsán na tvar infinitivu. Kdežto podstatné jméno saw (pila) by mělo zůstat nezměněno. Implementace lematizéru je však poměrně náročná. Proto se často místo lemmatizace používá pro převádění slov do základního tvaru jednodušší přístup stemming.

Stemming

Cílem stemmingu je, stejně jako v případě lemmatizace, přepis slov do jejich základního tvaru. Snahou je tedy odstranit odvozené tvary a skloňované koncovky slov. Toho však již není dosahováno morfologickou analýzou, ale prostým aplikováním sady přepisovacích pravidel. Jednoduchost tohoto postupu se samozřejmě odráží i na celkovém čase zpracování, kdy při morfologické analýze je třeba pracovat s větším množstvím textových dat, aby bylo možné správně určit základní tvar slova. Při stemmingu se pracuje pouze s jednotlivými slovy a pokud vyhovuje určitému tvaru, aplikuje se přepisovací pravidlo. To však může vést k mnoha chybným úpravám, které mohou vytvořit nekorektní slova.

```
walking     --->    walk
king        --->    k
sing        --->    s
```

Velice tedy záleží na pořadí a správné formulaci pravidel pro provedení úpravy. Příklad se slovy *king* a *sing* ukazuje chybně zapsané pravidlo, které odstraní poslední tři písmena, pokud má slovo koncovku *ing.* Lepší tvar pravidla by slovo přepsal pouze v případě, že se někde ve slově nachází také samohláska.

Nejpoužívanějším implementací používanou pro stemming anglického textu je Porterův algoritmus. Celý proces je rozdělen na celkem čtyři části, přičemž v prvním

kroku se provádí přepis slov v množném čísle a příčestí minulém. Následující pravidla[4] tvoří jednu ze skupin, která je aplikována právě v této fázi (Porter, 1980).

```
SSES -> SS                      caresses  ->  caress
IES  -> I                       ponies    ->  poni
                                ties      ->  ti
SS   -> SS                      caress    ->  caress
S    ->                         cats      ->  cat
```

Odstranění stop slov

Stop slova tvoří zvláštní skupinu slov. Všechna slova, které by bylo možné označit jako stop slova, nemají z hlediska určení celkového významu psaného textu příliš velkou informační hodnotu. Přesto se však jedná o slova, která se v textu vyskytují velice často, protože jde o obecná slova. Typickým příkladem stop slov mohou být předložky nebo spojky. Tato slova bývají proto odstraňována ze slovníku, čímž dojde k redukci jeho velikosti a sníží se nároky na paměť potřebou při dolování. Dále je uveden příklad stop slov pro anglický a český jazyk.

```
a, an, the, be, and, so, to, then, with, where, why, ...

a, byl, tak, tato, kdy, pak, proto, nebo, kdyz, jen, ...
```

Tato slova bývají definována v tzv. *seznamu stop slov* (angl. stop words list). Jedním z možných způsobů, jak podobný seznam získat, je spočítat frekvenci výskytu všech slov napříč celou kolekcí dokumentů a do seznamu zařadit ty s největším počtem výskytu. Problém však spočívá v tom určit, kolik nejčastějších slov začlenit do seznamu. Někdy může být odstranění stop slov také nežádoucí. Pokud se pracuje s frázemi, často bývají právě stop slova poměrně důležitá pro získání korektní fráze (Manning, Raghavan, Schütze, 2008).

Při konstrukci seznamu stop slov bývá také vhodné zohlednit oblast, které se zpracovávaná textová data týkají. Typický seznam slov je tak možné rozšířit o další často používaná obecná slova bez významové hodnoty, které jsou pro danou oblast specifické, nebo naopak některá častá, ale významově hodnotná slova ze seznamu vyřadit.

Odstranění slov dle frekvence výskytu

Stejně jako v případě odstranění stop slov, kdy jsou ze slovníku vyřazena nejčastěji se vyskytující obecná slova, je možné zredukovat velikost slovníku odstraněním dalších slov, čistě dle frekvence jejich výskytu. Tím opět dojde ke snížení náročnosti na paměť a může odjít ke zlepšení průběhu dolování. Odebrat ze slovníku je možné buď slova s příliš velkou četností, což bývají zpravidla obecnější slova s menší významovou nebo informační hodnotou, nebo naopak slova s příliš nízkou frekvencí výskytu, kdy

[4] převzato z článku An algorithm for suffix stripping

může jít o velice specifická slova použitá ve velmi malém množství dokumentů, která také nebudou z hlediska co nejefektivnějšího odhalení (obecné) znalosti příliš přínosná. Často se také může jednat o překlepy (Dařena, 2011).

Rozšíření o fráze

Většina předchozích operací prováděných při předzpracování textových dat se zaměřuje na zredukování velikosti slovníku a současném zvýšení kvality získané znalosti. Někdy však může být prospěšné slovník naopak rozšířit o fráze, neboli n-tice slov. Nejzákladnější podobou frází jsou tzv. *bigramy* složené ze dvou slov, případně *trigramy* složené ze tří slov. Je samozřejmě možné vytvářet i delší fráze obecně z n slov, tzv. *n-gramy*. Toto rozšíření slovníku však není dobré aplikovat globálně a vytvořit všechny možné varianty, protože čím větší bude slovník, tím náročnější bude následné hledání znalosti. Proto se zpravidla používájí pouze významné fráze, které jsou pro použitá textová data typické. Použití frází může být také vhodné při práci s velmi krátkými dokumenty, kdy se za účelem zlepšení reprezentace dat jednotlivé dokumenty uměle rozšiřují dalším textem.

Model vektorového prostoru

Při dolování je cílem odhalit nějakou obecnou znalost, kterou pak lze aplikovat na další dokumenty, které nebyly při hledání použity. Získanou znalostí mohou být například společné znaky pro určitou skupinu dokumentů, na základě kterých pak lze identifikovat příslušnost určitého dokumentu do dané skupiny. Jednotlivé dokumenty je tedy třeba mezi sebou nějakým způsobem porovnávat a pro tyto účely je také nutné zvolit vhodnou reprezentaci dokumentu.

Model vektorového prostuoru je jednou z reprezentací, která je založena na bag-of-words. Poprvé byl uveden v článku z roku 1975 (Salton, Wong, Yang, 1975). Všechny dokumenty z celé kolekce použité při dolování tvoří společně prostor dokumentů. Pokud by slovník vytvořený nad těmito dokumenty obsahoval celkem tři slova, bylo by možné každý takový dokument znázornit jako bod pomocí kartézského souřadnicového systému, přičemž tři slova ze slovníku by představovala tři základní osy x, y a z. Dokument lze tedy vyjádřit jako vektor, jehož jednotlivé složky představují četnost výskytu každého slova ze slovníku v daném dokumentu (Turney, Pantel, 2010). Dokumenty však zpravidla bývají mnohem delší a velikost vygenerovaného slovníku bývá tedy mnohonásobně větší než pouhá tři slova. Místo trojrozměrného prostoru se pracuje s prostorem o n rozměrech, přičemž n je rovna velikosti slovníku. Každý dokument je pak vyjádřen pomocí vektoru o n složkách.

$$n = |vocabulary|$$

$$d \in \mathbb{N}^n$$

$$d = (w_1, w_2, w_3, ..., w_n)$$

Tomuto vektoru se také někdy říká příznakový vektor (angl. feature vector). Jako hodnoty jednotlivých příznaků vektoru se nejčastěji používají frekvence výskytu jednotlivých slov v daném dokumentu, kterým jsou však přiřazeny statistické váhy pro určení důležitosti každého slova (Wang, Zhang, Vassileva, 2010).

Každý vektor reprezentující konkrétní dokument je však velmi řídký. To je typickou charakteristikou pro reprezentaci textových dat pomocí vektoru. Tuto skutečnost lze ukázat na vzorku uživatelských recenzí dostupných u produktů prodávaných elektronickým obchodem Amazon. Nad kolekcí obsahující celkem sto tisíc recenzí byl vygenerován slovník o velikosti 13 026 slov. Nejkratší dokument obsahoval celkem dvě slova, nejdelší 187 slov a průměrná délka dokumentu byla dvacet slov. Řídkost vektoru reprezentující dokument o průměrné délce tak činí zhruba 99,85 %, takže drtivá většina složek vektoru je rovna nule.

Na základě způsobu uchovávání hodnot ve vektorech je tedy možné rozlišit dva hlavní typy reprezentace:

- hustá (angl. dense),
- řídká (angl. sparse).

U hustých vektorů jsou ukládány všechny složky vektoru. Pokud by tedy byl dokument z kolekce výše popsaných recenzí reprezentován pomocí hustého vektoru, obsahoval by celkem 13 026 složek, ze kterých by pouhých dvacet bylo nenulových. V případě řídkého vektoru jsou nulové hodnoty vypuštěny a nenulové hodnoty jsou ukládány jako dvojice index složky vektoru a hodnota dané složky. Aby byl tento způsob reprezentace efektivní, musí být alespoň polovina složek vektoru nulová. U ukázkové kolekce dokumentů je více jak 99 procent hodnot nulových, takže reprezentace textových dat řídkými vektory je mnohem efektivnější než u hustých vektorů.

Celou kolekci dokumentů je možné znázornit pomocí tzv. *term-document* matice, ve které řádky představují jednotlivé dokumenty a sloupce jednotlivá slova ze slovníku, označované také jako termy (viz tabulka 1).

Vážení frekvence výskytu slov v dokumentu má za úkol lépe vyjádřit důležitost daného slova. Pokud se slovo vyskytne v dokumentu desetkrát, bude pro určení dokumentu důležitější více, než kdyby se v dokumentu vyskytlo pouze jednou. Důležitost tohoto slova však není desetkrát větší. Již zmíněná stop slova jsou velice obecná, a proto se v dokumentech vyskytují ve velkých počtech. Jejich informační hodnota je však velice malá. To samé do určité míry platí i pro ostatní slova. Čím častěji se v dokumentu vyskytují, tím bývají obecnější a jejich informační hodnota spíše klesá. Důležitost slova tedy neroste proporcionálně s počtem jejich výskytu. Způsob, jak tuto skutečnost promítnout do vektorové reprezentace dokumentu, ukazuje následující rovnice.

$$weight_{d,t} = \frac{local\_weight_{d,t} \cdot global\_weight_t}{normalization_{d,t}}$$

Tabulka 1: Reprezentace kolekce dokumentů pomocí Term-document count matrix

	t_1	t_2	t_3	$\cdots$	t_j	$\cdots$	t_n
d_1	w_{11}	w_{12}	w_{13}	$\cdots$	w_{1j}	$\cdots$	w_{1n}
d_2	w_{21}	w_{22}	w_{23}	$\cdots$	w_{2j}	$\cdots$	w_{2n}
d_3	w_{31}	w_{32}	w_{33}	$\cdots$	w_{3j}	$\cdots$	w_{3n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
d_i	w_{i1}	w_{i2}	w_{i3}	$\cdots$	w_{ij}	$\cdots$	w_{in}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
d_m	w_{m1}	w_{m2}	w_{m3}	$\cdots$	w_{mj}	$\cdots$	w_{mn}

Výsledná váha pro slovo t v dokumentu d je dána jako součin jeho lokální váhy (v rámci daného dokumentu) a jeho globálního váhy (v rámci celé kolekce dokumentů). Získaný součin vah je pak možné pomocí podílu normalizovat. Mezi nejčastěji používané funkce pro určení lokální váhy termu patří (Singhal, 1997):

- *term frequency:* tato funkce vrací počet výskytů termu v daném dokumentu.

```
function term-frequency(term, dokument) returns frekvence
vstupy:              term      // počítaný term
                     dokument  // seznam slov
lokální proměnné:    frekvence // počet výskytu termu
frekvence <- 0
for each slovo in dokument
do
    if slovo shodné s term then navýšit frekvence
done
return frekvence
```

- *term presence:* tato binární funkce zcela ignoruje počet výskytů a vrací pouze příznak, zda se term v dokumetu vyskytuje nebo ne.

$$local\_weight_{d,t} = \begin{cases} 1 & \text{pokud term-frequency}_{d,t} > 0 \\ 0 & \text{jinak} \end{cases}$$

  Pokud by byla použita pouze tato metoda vážení, tak by celá kolekce dokumentů byla reprezentována zvláštním případem matice – incidenční matice term-document.

- *logarithmic term frequency:* výsledek této funkce je v základní podobě počítán pomocí následujícího vztahu

$$local\_weight_{d,t} = \begin{cases} \log_{10} \text{term-frequency}_{d,t} & \text{pokud term-frequency}_{d,t} > 0 \\ 0 & \text{jinak} \end{cases}$$

  Pomocí tohoto způsobu lokálního vážení výskytu termu je možné zohlednit výše popsaný problém s neproporcionálním růstem důležitosti termu vzhledem k frekvenci jeho výskytu. Díky charakteristice logaritmické funkce se tedy s rostoucí frekvencí výskytu zpomaluje růst důležitosti. Je možné použít i lehce upravenou variantu

$$local\_weight_{d,t} = \begin{cases} 1 + \log_{10} \text{term-frequency}_{d,t} & \text{pokud term-frequency}_{d,t} > 0 \\ 0 & \text{jinak} \end{cases}$$

  využívající tzv. *aditivního vyhlazování* (angl. add-one smoothing).

- *augmented term frequency:* při použití této funkce dochází k zredukování rozsahu přírůstků důležitosti do určitého intervalu, nejčastěji do rozmezí hodnot 0,5 a 1,0. Pracuje se s předpokladem, že pouhá přítomnost slova je ohodnocena základní hodnotou k (obvykle 0,5). Za každý další výskyt je tato hodnota lineárně navýšena až po nejvyšší frekvenci, pro kterou bude vrácena maximální hodnota intervalu (1,0). Výpočet vypadá následovně

$$local\_weight_{d,t} = k + (1 - k) \cdot \frac{\text{term-frequency}_{d,t}}{\text{max-term-frequency}_d}$$

- *okapi's term frequency:* tato funkce je založena na aproximaci 2-Poissonova modelu a její charakteristika je velice podobná funkci logarithmic term frequency.

$$local\_weight_{d,t} = \frac{\text{term-frequency}_{d,t}}{2 + \text{term-frequency}_{d,t}}$$

Pro určení celkové váhy však pouhé lokální vážení frekvence výskytu termu nestačí. Při hodnocení důležitosti daného slova je nutné zohlednit nejen jeho významnost v rámci dokumentu, ale také v rámci celé kolekce dokumentů. V opačném případě by došlo k tomu, že všechna slova, i když se vyskytnou v rámci celé kolekce pouze v jednom dokumentu, jsou brána jako informačně stejně přínosná. K tomuto rozlišení slouží globální vážení termu. Možností je opět několik:

- *žádné vážení:* zohlednění globální váhy slova při výpočtu jeho celkové váhy je možné vyrušit nastavením jednotkové váhy

$$global\_weight_t = 1$$

- *inverse document frequency:* základem pro výpočet globální váhy termu je počet dokumentů, ve kterých se daný term vyskytuje.

```
function document-frequency(term, dokumenty) returns frekvence
vstupy:             term      // počítaný term
                    dokumenty // seznam seznamů slov
lokální proměnné:   frekvence // počet výskytu termu
frekvence <- 0
for each dokument in dokumenty
do
    if dokument obsahuje term then navýšit frekvence
done
return frekvence
```

  Cílem však je snížit významnost termu, který lze považovat za obecný, tedy v čím větším množství dokumentů se term vyskytuje, tím nižší by měla být jeho významnost. Frekvence výskytu slova v rámci dokumentu by tedy měla být redukována hodnotou, která roste s nižším výskytem termu v rámci kolekce dokumentů. Toho je možné dosánout použitím inverzní relativní hodnoty výskytu termu v kolekci.

$$global\_weight_t = \log \frac{m}{\text{document-frequency}_t}$$

  Logaritmus je zde, stejně jako u frekvenci výskytu termu v dokumentu, použit pro zredukování růstu hodnot. Tento způsob vážení je většinou označován zkratkou IDF.

- *probabilistic inverse document frequency:* tato funkce je velice podobná funkci IDF, ale výpočet důležitosti termu je založen na pravděpodobnosti relevance daného termu, což je podíl počtu dokumentů neobsahujících term vůči počtu dokumentů obsahujících term.

$$global\_weight_t = \log \frac{m - \text{document-frequency}_t}{\text{document-frequency}_t}$$

- *global frequency inverse document frequency:* tato funkce přiřazuje nejmenší možné hodnoty termům, které se vyskytují ve všech dokumentech.

$$global\_weight_t = \log \frac{\text{global-frequency}_t}{\text{document-frequency}_t}$$

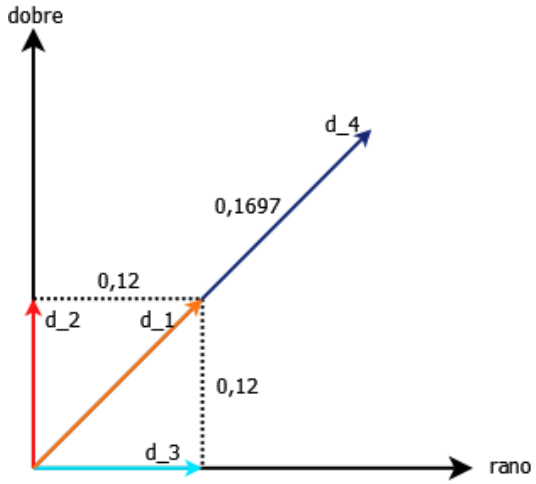
Pomocí globálních vah je tedy možné zohlednit frekvenci výskytu termu v celé kolekci dokumentů a přidat tak na důležitosti méně používaným slovům a zároveň ubrat na důležitosti mnohem obecnějším slovům, která se objevují ve většině dokumentů z kolekce. Výsledkem součinu lokální a globální váhy je tedy celková váha termu. Posledním krokem ve výpočtu vektorů je jejich normalizace. Důvodem

provádění normalizace je zajištění toho, aby dokumenty používající stejná slova byly ve vektorovém prostoru reprezentovány body ležícími blízko sebe. Bez provedení normalizace totiž mohou být dokumenty obsahující podobná slova od sebe značně vzdáleny. Důvodem mohou být jejich rozdílné délky, díky kterým jsou stejná slova vyjádřena rozdílnými (násobnými) hodnotami. Navíc delší dokumenty obsahují více slov a důsledkem je opět prodlužování vzdálenosti mezi dokumenty. Vliv použití či nepoužití normalizace je možné ukázat na jednoduchém příkladu, kdy je k dispozici zjednodušená kolekce čtyř dokumentů, ve kterých jsou použita pouze dvě slova.

```
vocabulary = (dobre, rano)

d_1 = "dobre rano"              -> d_1 = (1, 1) -> d_1 = (0.12, 0.12)
d_2 = "dobre"                   -> d_2 = (1, 0) -> d_2 = (0.12, 0.00)
d_3 = "rano"                    -> d_3 = (0, 1) -> d_3 = (0.00, 0.12)
d_4 = "dobre rano dobre rano" -> d_4 = (2, 2) -> d_4 = (0.24, 0.24)
```

Z hlediska podobnosti by měl být první dokument nejblíže ke čtvrtému dokumentu, protože obsahují více stejných slov, zatímco s dalšími dvěma dokumenty má společné pouze jedno slovo. Na obrázku 3 jsou dokumenty zobrazeny ve dvourozměrném



Obrázek 3: Znázornění vzdálenosti dokumentů ve vektorovém prostoru vyjádřených pomocí TF-IDF

prostoru, spolu s jejich vzdáleností od prvního dokumentu. Provedením normalizace dojde k přepočítání jednotlivých složek vektorů, které pak budou náležet do prostoru vyhrazeném jednotkovým vektorem, čímž dojde k přiblížení vektorů se stejnými slovy, ale různými počty výskytu těchto slov. V případě ukázkové kolekce dokumentů budou vektory reprezentující první a čtvrtý dokument po normalizaci totožné. Dále jsou uvedeny některé techniky pro provedení normalizace délky vektoru:

- *žádná normalizace:* stejně jako u globální váhy je možné potlačit i efekt nor-

malizace a ponechat vektory v původní podobě

$$normalization_{d,t} = 1$$

- *cosine normalization:* kosinová normalizace patří mezi nejpoužívanější přístupy pro normalizaci délek vektorů. Pro převedení vektoru do jednotkového prostoru se používá Eukleidovská délka daného vektoru. Aplikováním normalizace je pak každá složka vektoru podělena jeho délkou. Čím jsou vyšší hodnoty výskytu slov v dokumentu, tím se celková délka vektoru prodlužuje a vyšší dělitel tak více snižuje váhu slova. Stejně tak větší množství slov v dokumentu prodlužuje délku jeho vektoru a opět navyšuje hodnotu dělitele. Tento způsob normalizace tedy zohledňuje oba důsledky delších dokumentů.

$$normalization_{d,t} = \sqrt{\sum_{n=1}^{n} w_{d,i}^2}$$

- *maximum term-frequency normalization:* normalizace využívající pro převod váh slov do jednotkového prostoru maximální váhu v dokumentu je principem totožná s již zmíněným způsobem výpočtu globální váhy pomocí funkce *augmented term frequency.* Ta přiřazovala určitou základní váhu minimální hodnotě frekvence slova v kolekci dokumentů a zbytek z jednotkového intervalu rovnoměrně rozdělila mezi ostatní hodnoty výskytu. Slovo s maximálním výskytem tak bylo vždy přepočítáno na hodnotu 1. Stejný princip je možné použít i při normalizaci délky vektoru.

$$normalization_{d,t} = \text{max-weight}_d$$

  Nevýhodou tohoto přístupu je, že při normalizaci jsou ve výpočtu zohledněny pouze vyšší hodnoty výskytu slov v dokumentu. Problém s prodlužováním vzdálenosti v důsledku většího počtu slov u delších dokumentů tento způsob normalizce neřeší.

Uvedený výčet způsobů výpočtu váhy slova rozhodně nepředstavuje kompletní výčet. Cílem je spíše poskytnout základní představu o možnostech reprezentace textových dat pomocí vektorů a nastínit výhody a nevýhody různých přístupů. Obecně se pro určení vah nejčastěji používá schéma *Term frequency-Inverse Document Frequency*, zkráceně TF-IDF, a při nutnosti normalizace délek výsledných vektorů je použita kosinová normalizace.

2.3 Klasifikace dokumentů

Klasifikace textových dokumentů je jedním z typických problémů, které lze v rámci dolování znalostí z textových dat řešit. Mimo klasifikaci jde také o predikci, vyhledávání a extrakci informací (Weiss et. 2010). Před samotným formálním popisem problému klasifikace dokumentů bude uvedeno několik reálných problémů, pro jejichž řešení se využívá právě klasifikace.

Zřejmě nejrozšířenějším příkladem, se kterým si dnes a denně setkává většina lidí, je detekování nevyžádané pošty označované jako *spam.* Většina elektronické pošty obsahuje pouze textová data, někdy však bývají obohaceny o HTML tagy. To samo o sobě už může sloužit jako atribut dokumentu, pomocí kterého je možné určit, zda se jedná o spam či nikoliv. Při klasifikaci elektronické pošty je možné kromě

```
From:    benjyx556iii1990@seznam.cz
Subject: Máme pro Vás peníze zdarma - kdo si hraje, nezlobí!
Date:    15 November 2012 08:57:51 CET
To:      Zdeněk Novák
-----------------------------------------------------------------
Využijte speciální nabídky,
jak získat peníze.

Čeká vás propracovaná grafika, napětí a zábava.
Zahrajte si 50x zdarma oblíbenou hru na internetu.

Vše, co vyhrajete si můžete nechat!

ZAČÍT HRÁT
```

Obrázek 4: Ukázka elektronické pošty ozančováné jako spam

samotného textu zprávy využít i další atributy, jako adresa odesílatele nebo text předmětu zprávy. Na ukázkovém mailu (viz obrázek 4) je vidět, že předmět obsahuje slova jako *peníze, zdarma,* a také vykřičník. Stejně tak adresa odesílatele nemá úplně typický tvar a nenachází-li se již v příjemcově kontaktním adresáři, může přispět ke klasifikaci zprávy jako nevyžádané. Z hlediska klasifikace textu je však nejdůležitější text zprávy. Ten opět obsahuje slova jako *speciální nabídka, peníze, zábava, zdarma*, které jsou typické spíše pro spam. Všechny tyto atributy mailu mohou být využívány klasifikátorem, který na základě adresy odesílatele a přítomnosti určitých slov ve zprávě či jejím předmětu rozhodne, zda nově příchozí pošta skončí ve spamovém koši.

Dalším příkladem klasifikační úlohy je analýza mínění. Typickým zástupcem analýzy mínění je kategorizace psaných recenzí na pozitivní a negativní. Princip je stejný jako v případě odhalování nevyžádané pošty. V textu recenze se hledají určitá slova, která svým významem pomohou určit, jaký je význam celého textu. Ve dvou ukázkových recenzích převzatých z Česko-Slovenské filmové databáze (viz obrázek 5) jsou tato slova na první pohled snadno identifikovatelná. V první recenzi jsou použita slova jako *skvělý, výtečný, perfektní* nebo *líbit,* která dávají vcelku jasný signál o spokojenosti uživatele s filmem. Oproti tomu v druhé recenzi jsou naopak slova s negativním sentimentem jako *nuda, strašné* či *přecenit.* Tyto příklady

```
Skvělý film, vítečný J.Nicholson. jak se dokázal pohybovat mezi
blázny a pefektně zahrát. Moc se mi to líbilo :-) 92%

Nuda, nuda, strašná nuda. Nechápem, ako sa podarilo takto
preceniť tento film u takej veľkej masy divákov... Zázrak.

-----------------------------------------------------------------

Film je zcela brilantním dílem, naprostý skvost ve své kategorii,
dosahuje téměř stejných kvalit jako Catwoman s Halle Berry.
```

Obrázek 5: Ukázka uživatelských recenzí.
Zdroj: `http://www.csfd.cz/film/2982-prelet-nad-kukaccim-hnizdem/`

slouží však pouze pro ilustraci, protože text může často obsahovat slova, která obrací význam některých přídavných jmen. Nejnáročnější mohou pro automatizovanou klasifikaci textu být ironické věty, které sice obsahují pozitivní slova, ale jejich význam je přesně opačný. Pro člověka by neměl být problém rozpoznat negativní tón smyšlené recenze (viz obrázek 5), avšak pokud není seznámen s velice negativním popkulturním míněním o filmu Catwoman, na který se text recenze odvolává, může i člověk dojít k mylnému závěru. Pro klasifikátor je situace v podstatě velice podobná a bez zahrnutí znalosti o kvalitách ostatních filmů je odkázán pouze na text recenze, který obsahuje převážně pozitivní slova. Je tedy velice pravděpodbné, že podobný text by byl chybně označen jako pozitivní.

Klasifikace je však velice obecnou úlohou a aplikovat ji lze na celou řadu problémů, nejen v oblasti zpracování textových dat. Zde jsou uvedeny některé typické klasifikační úlohy s textovými daty (Manning, Raghavan, Schütze, 2008):

- detekce spamu,
- analýza mínění,
- přiřazení autora,
- identifikace jazyka,
- rozlišení pohlaví autora,
- kategorizace článků dle tématu.

Cílem klasifikace textových dokumentů je umožnit automatické přířazování dokumentu d do jedné z možných tříd c. Pokud definujeme množinu možných tříd jako $\mathcal{C} = \{c_1, c_2, c_3, ..., c_{|\mathcal{C}|}\}$ a množinu dokumentů jako $\mathcal{D} = \{d_1, d_2, d_3, ..., d_{|\mathcal{D}|}\}$, tak je možné klasifikaci formálně definovat jako funkci, která přiřazuje každé uspořádané dvojici $(d_i, c_j) \in \mathcal{D} \times \mathcal{C}$ logickou hodnotu *pravda,* pokud d_i náleží

do c_j a *nepravda* v opačném případě. Po definovaní množiny logických hodnot $\mathcal{L} = \{pravda, nepravda\}$ je tedy možné zapsat klasifikační funkci v následujícím tvaru:

$$\Phi : \mathcal{D} \times \mathcal{C} \rightarrow \mathcal{L}$$

Řešením klasifikační úlohy je však nalezení funkce $\hat{\Phi}$, která je co nejpřesnější aproximací funkce Φ. Tato funkce je označována jako *klasifikátor* (Sebastiani, 2005).

Důvodem, proč se zabývat řešením automatické klasifikaci textových dokumentů, je především náročnost ručního třídění textů. Člověk si především musí celý text přečíst, aby plně porozuměl jeho obsahu a mohl jej správně zařadit do některé ze tříd. To sice nemusí být problém například u recenzí, které bývají většinou krátké a jejich přečtení tak zabere maximálně několik málo minut, většinou jich však bývá velké množství. Klasifikace velkého množství textových dokumentů čistě lidskými silami je tedy velice náročná (Manning, Raghavan, Schütze, 2008).

Pokud by bylo nutné dosáhnout výsledků v co nejkratším čase, je třeba do klasifikace zařadit větší množství lidí, mezi které by byly jednotlivé dokumenty rozděleny. Tento přístup v sobě však skrývá problém související s jednou z charakteristik klasifikace. Rozhodnutí o příslušnosti dokumentu do určité třídy je ovlivněn zkušenostmi jednotlivých lidí a konečné zařazení dokumentu je výsledkem subjektivního rozhodnutí (Sebastiani, 2005). Různí lidé se tak velice často nemusí na příslušnosti dokumentu do třídy shodnout. Rozdělením klasifikace skupiny dokumentů mezi více lidí tedy dojde ke ztrátě konzistence při přidělování tříd. Použitím automatického klasifikátoru je tento problém eliminován.

Dokument je v jednodušší variantě klasifikace zařazován vždy výhradně do jedné ze tříd. Pro každý dokument $d_i \in \mathcal{D}$ tedy platí, že má přiřazenu pouze jednu třídu $c_j \in \mathcal{C}$. Dokument je stejně tak možné klasifikovat do více tříd, včetně nezařazení do žádné ze tříd. Nejčastěji bývá klasifikační úloha řešena jako binární. Úkolem klasifikátoru je tedy zařadit dokument pouze do jedné ze dvou tříd, což lze přeformulovat na problém přiřazení či nepřiřazení do jedné třídy (druhou třídu představuje doplněk třídy první). Funkce binárního klasifikátoru je tím pádem definována jako aproximace funkce $\Phi : \mathcal{D} \rightarrow \mathcal{L}$. Binárního klasifikátoru je možné využít i pro řešení klasifikační úlohy s více třídami, a to tak, že se pro každou třídu vytvoří zvláštní binární klasifikátor. Konečný klasifikátor pro celou množinu tříd $\mathcal{C}$ tvoří skupina těchto binárních klasifikátorů. Důvodem pro řešení vícetřídní klasifikace tímto způsobem je menší složitost tvorby binárního klasifikátoru.

Pro nalezení aproximační funkce a vytvoření klasifikátoru se používají různé algoritmy strojového učení. Ty mají za úkol automaticky odhalit na množině tzv. *trénovacích dat* pravidla, pomocí kterých se následně rozhoduje o příslušnosti dokumentu do určité třídy. Aby bylo možné tato pravidla v textových datech nalézt, je nutné poskytnout dostatečně velkou množinu trénovacích dokumentů, ale zároveň co nejvíce vyváženou z hlediska počtu dokumentů zastupujících jednotlivé třídy. Z toho také plyne, že všechny dokumenty z trénovací množiny musí mít přiřazenu některou ze tříd, které chceme pomocí klasifikátoru rozeznávat.

Bayesovský klasifikátor

Naivní Bayesův klasifikátor patří k základním algoritmům strojového učení, které se používají při klasifikaci textových dat. Jde o jednoduchých přístup ke klasifikaci, jehož základem je tzv. Bayesovský teorém (Manning, Raghavan, Schütze, 2008)

$$P(c_j|d_i) = \frac{P(d_i|c_j)P(c_j)}{P(d_i)}$$

Při výpočtu se pracuje s apriorní pravděpodobností výběru klasifikační třídy c $P(c_j)$, tedy že náhodně vybraný dokument bude patřit do třídy c, dále s pravděpodobností výběru dokumentu d $P(d_i)$ a s pravděpodobností výskytu dokumentu d ve třídě c $P(d_i|c_j)$. Výsledkem výpočtu je podmíněná pravděpodobnost, že při výběru dokumentu d bude jeho klasifikační třídou třída c.

Aby tedy bylo možné určit třídu, do které náhodně vybraný dokument patří, je třeba z trénovacích dat vypočíta jednotlivé pravděpodobnosti $P(c_j)$, $P(d_i)$ a $P(d_i|c_j)$. Pro určení třídy neznámého dokumentu se následně spočítají pravděpodobnosti $P(c|d)$, a to pro všechny možné třídy.

$$\begin{aligned} c &= \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} P(c_j|d_i) \\ &= \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} \frac{P(d_i|c_j)P(c_j)}{P(d_i)} \\ &= \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} P(d_i|c_j)P(c_j) \end{aligned}$$

Dokument je zařazen do té třídy, která maximalizuje aposteriorní pravděpodobnost $P(c|d)$. Při výpočtu byl vypuštěn dělitel s pravděpodobností dokumentu, protože jde o konstantní hodnotu a nemá tudíž na výsledek žádný vliv. Jelikož jsou texty reprezentovány pomocí příznakových vektorů, je ignorováno pořadí slov v daném dokumentu. Zároveň se (naivně) předpokládá, že jednotlivé příznaky jsou na sobě vzájmně nezávislé. Pravděpodobnost $P(d_i|c_j)$ je tedy dána jako součin pravděpodobnosti výskytu slov w ve třídě c.

$$P(w_1, w_2, w_3, ..., w_n|c_j) = \prod_{w_k \in d_i} P(w_k|c_j)$$

$$c = \underset{c_j \in \mathcal{C}}{\operatorname{argmax}} P(c_j) \prod_{w_k \in d_i} P(w_k|c_j)$$

Rozhodovací stromy

Intuitivním přístupem ke klasifikaci textových dat jsou rozhodovací pravidla, kdy se na základě přítomnosti konkrétních slov rozhodne, do jaké třídy dokument patří. Rozhodovací stromy jsou implementací tohoto přístupu. Jde o význačný typ grafu,

který obsahuje dva typy uzlů, prvním jsou rozhodovací uzly, které představují jednotlivá rozhodovací pravidla. Dle možných odpovědí na dané pravidlo vede z rozhodovacího uzlu stejný počet hran. Druhým typem uzlu jsou listy uchovávající třídu, do které je dokument klasifikován. Při klasifikaci se postupuje od prvního pravidla (kořene stromu) a postupným vyhodnocováním pravidel se přes hrany dokument přesunuje do podstromů, až dorazí k jednomu z listů, čímž dojde k připřazení dokumentu do třídy reprezentované daným listem (Russell, Norvig, 1995).

Z trénovacích dat se tedy při dolování extrahují pravidla a odpovědi, které vytvoří rozhodovací strom. Snahou je, aby bylo pravidel co nejméně a strom byl co nejjednodušší, tzn. aby odpovědi co nejvíce rozdělovaly data do jednotlivých tříd. Pokud je k dispozici množina dokumentů obsahující 50 příkladů za každou ze dvou tříd, tak by špatné pravidlo tuto množinu rozdělilo na podmnožiny, ve kterých budou obě třídy zastoupeny opět stejným počtem dokumentů. Dobré pravidlo by naopak vytvořilo více homogenní podmnožiny, ve kterých by výrazněji převažovala jedna třída. Využití zde tedy nalézá *teorie informace.* Pravidla jsou vytvářena tak, aby pokud možno co nejvíce minimalizovala míru nevědomosti či chaosu počítanou pomocí *entropie.*

$$H(X) = -\sum p_i \log_2 p_i$$

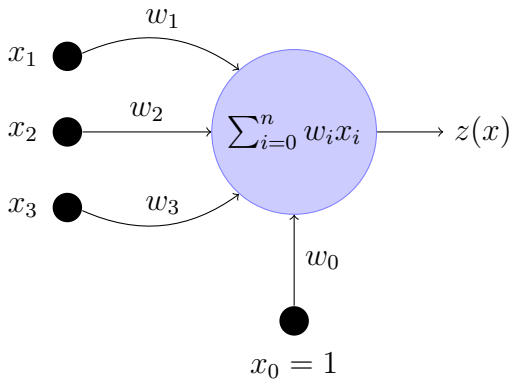
Atribut dat (v případě textu slovo) zvolený jako rozdělovací parametr musí analogicky maximalizovat informační zisk. Pro každý atribut je tedy spočítána tato hodnota a pro rozdělení se zvolí ten, který poskytne maximální informační zisk. Ten se počítá jako vážený součet entropií jednotlivých podmnožin odečtený od nynejší entropie.

Neuronové sítě

Snahou algoritmu neuronových sítí je napodobit princip fungování lidského mozku, konkrétněji způsobu, jakým se mozek učí novým znalostem. Základní stavební jednotkou v mozku je buňka zvaná *neuron.* Jeho účelem je přijímání nervových impulsů pomocí *dendritů* a naopak výsílání nervových impulsů k ostatním neuronům skrze *axony.* Výstupní signál je však neuronem vyslán pouze ve chvíli, kdy jeho vstupní signály překročí určitou prahovou hodnotu (Šíma, Neruda, 1996). Umělé neuronové sítě modelují biologickou neuronovou síť a jejím základním prvkem je tedy umělý neuron, označován také jako *perceptron.* Princip fungování je zcela totožný s biologickým neuronem. Perceptron přijímá na vstupech hodnoty s nimiž provede vážený součet (viz obrázek 6). Výsledek součtu je použit jako vstup pro aktivační funkci neuronu, která převede součet vstupních hodnot na velikost výstupního signálu, který bude neuron vysílat do sítě. Nejčastěji se používá logistická přechodová funkce

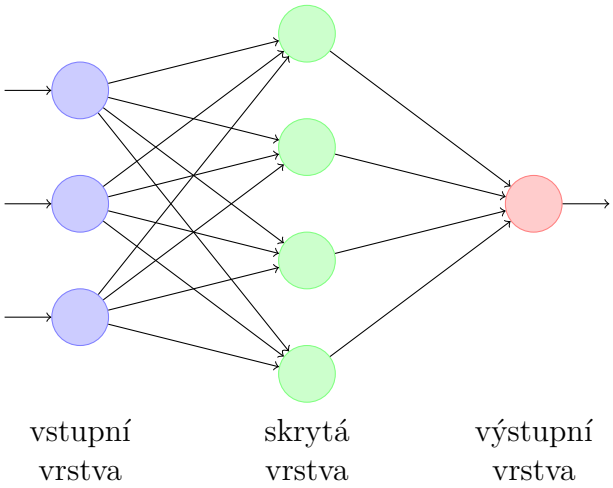
$$z(x) = \frac{1}{1 + e^{-x}},$$

je ale možné použít i funkce jiné (mezi další často používané patří např. skoková nebo lineární).



Obrázek 6: Model perceptronu

Pomocí perceptronu je možné nalézt lineární hranici mezi dvěma skupinami prvků v prostoru. Pro nalezení nelineární hranice, jsou neurony zapojovány do sítí. Nejčastějším zapojením neuronů je třívrstvá architektura (viz obrázek 7). Síť je rozdělena na tři vrstvy, první je vstupní vrstva, poslední výstupní vrstva a mezi nimi se nachází tzv. skrytá vrstva. Pokud se na dvě sousední vrstvy v neuronové síti bude nahlížet jako na graf s neurony reprezentující uzly a vazby mezi neurony hrany, tak způsob zapojení neuronů odpovídá úplnému bipartitnímu grafu, tzn. že každý neuron z jedné vrstvy je vždy propojen s každým neuronem z druhé vrstvy. Neuronovou síť



Obrázek 7: Schématické znázornění třívrstvé neuronové sítě

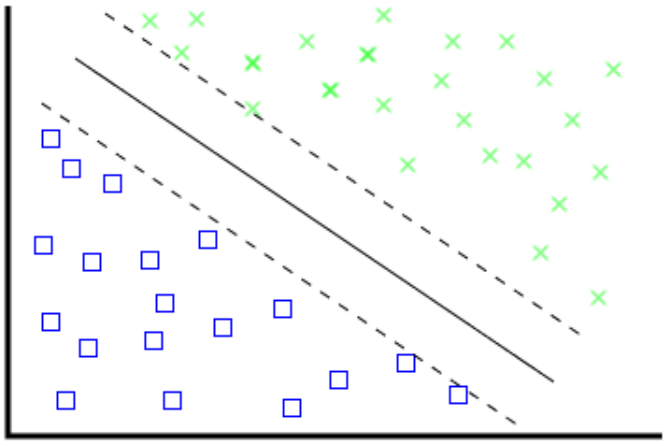
je třeba naučit nějaké znalosti. V současné době existují různé přístupy k procesu učení sítě, nejzákladnějším a také nejčastěji používaným algoritmem je *zpětné šíření chyby* (angl. backpropagation). Během učení dochází k aktualizaci vah vstupů pro jednotlivé neurony tak, aby se minimalizovala chybová funkce.

Při práci s textovými daty jsou na vstup neuronové sítě předávány jednotlivé příznakové vektory. Díky dříve uvedeným charakteristikám textových dat reprezen-

tovaných pomocí vektorů tak musí vzniknout síť s řádově několika tisíci neuronů na vstupní vrstvě. Počet neuronů ve výstupní vrstvě se odvíjí od klasifikační úlohy. Pokud jsou rozlišovány pouze dvě třídy, postačí jediný neruon. V případě vícetřídní klasifikace už musí být přítomno tolik neuronů, kolik je rozlišováno tříd. Velikost skryté vrstvy se volí experimentálně, podle kvality dosahovaných výsledků. Čím vyšší je celkový počet neuronů, tím roste časová náročnost na proces učení neuronové sítě, takže pro klasifikaci textových dat nemusí být použití sítí nejlepší možnou variantou.

Podpůrné vektory

Metoda podpůrných vektorů (angl. support vector machines, zkráceně SVM) slouží pro rozdělení bodů ve vícerozměrném prostoru do dvou skupin, a to buď lineární nebo nelineární hranicí. Jednotlivé body jsou od sebe odděleny pomocí hyperroviny, která je umístěna tak, aby hraniční pásmo mezi oběma skupinami bylo co největší (viz obrázek 8), čímž se dosáhne snížení chybovosti při klasifikování náhodně vybraného neznámého prvku (Vapnik, 1995). Problém tedy spočívá v nalezení optimálního umístění roviny, protože způsobů jejího umístění je teoreticky nekonečně mnoho. Pro hledání rozdělující hyperroviny se používájí tzv. jádrové (kernel) funkce, kterých existuje více druhů. Základní funkce umožňuje nalézt pouze linearní hranici, podobně jako např. perceptron. Použitím polynomiální nebo radiální bázové funkce je možné nalézt i hranici nelineární. Ta je nalezena pomocí transformace



Obrázek 8: Ukázka hraničního pásma a hyperroviny rozdělující dvě skupiny prvků.
Zdroj: `http://phaedrusdeinus.org/svm-lightning/`

prvků do nového prostoru s více dimenzemi, který umožní nalezení lineární hranice. Zpětnou transformací se dostane hranice nelineární. Algoritmus podpůrných vektorů je pro práci s textovými daty velice vhodný (Joachims, 1998). Vektory reprezentující

textová data jsou velice řídké a skládájí se z velkého množství atributů. SVM má totiž tu vlastnost, že schopnost nalezení hranice může být nezávislá na dimenzionalitě vektorového prostoru. Pokud tedy v datech existuje nějaké hraniční pásmo, je možné generalizovat i vysoce dimenzionální data, což je přesně případ vektorů reprezentující textová data.

Metoda nejbližšího souseda

Metoda nejbližšího souseda, potažmo k nejbližších sousedů, je založena na porovnávání dokumentů a výpočtu jejich vzájemné podobnosti. Místo procesu učení, který z trénovacích generalizuje nějakou znalost, jsou trénovací příklady spolu s informací o příslušnosti do některé ze tříd ukládány do databáze a jednotlivé atributy slouží jako souřadnice ve více rozměrném prostoru, čímž hodnoty těchto atributů určují polohu dokumentu v daném prostoru (Manning, Raghavan, Schütze, 2008). Neznámý případ je následně porovnán se všemi záznamy v databázi a jeho třída se určí podle k nejbližších (nejpodobnějších) případů. Počet uložených dokumentů je důležité pečlivě vybírat, protože při malém počtu nemusí být k dispozici dostatek informací pro správné klasifikování neznámých případů. Na druhou stranu velké množství bude prodlužovat proces hledání nejbližších sousedů. V případě textových dat jsou dokumenty reprezentovány příznakovými vektory. Předpokládá se, že dokumenty ze stejné třídy budou do určité míry používat shodná slova, a tak budou i ve vícerozměrném prostoru blízko sebe.

Pro měření podobnosti jednotlivých dokumentů existuje celá řada metrik. Nejzákladnější je *Eukleidova vzdálenost.*

$$\delta(d_t, d_q) = \|d_t - d_q\| = \sqrt{\sum_{i=1}^{n}(w_{t,i} - w_{q,i})^2}$$

Ta měří rozdíl mezi dotazovaným dokumentech d_q a trénovacím dokumentem d_t jako druhou odmocninu ze součtu čtverců. Použití této vzdálenosti bývá u většiny problémů řešených pomocí metody nejbližšího souseda nejčastější. Dalším používaným typem je *Manhattanská vzdálenost.*

$$\delta(d_t, d_q) = |d_t - d_q| = \sum_{i=1}^{n} |w_{t,i} - w_{q,i}|$$

Podobnost dokumentů je počítána jako součet absolutních rozdílu mezi jednotlivými příznaky vektorů reprezentující dané dokumenty. Posledním zde uvedeným způsobem výpočtu podobnosti dokumentů je *kosinová vzdálenost,* kdy hodnota podobnosti dvou dokumentů je dána hodnotou kosinu úhlu, který spolu příznakové vektory těchto dokumentů svírají.

$$\delta(d_t, d_q) = \frac{d_t \cdot d_q}{\|d_t\|\|d_q\|} = \frac{\sum_{i=1}^{n} d_{t,i} d_{q,i}}{\sqrt{\sum_{i=1}^{n} w_{t,i}^2}\sqrt{\sum_{i=1}^{n} w_{q,i}^2}}$$

Kosinová vzdálenost je pro hodnocení podobnosti textových dat velice vhodná, protože nezohledňuje[5] rozsah hodnot jednotlivých příznaků obou vektorů. Pokud dokumenty obsahují stejná slova, ale v různých počtech, může při použití Eukleidovské vzdálenosti vyjít jako více podobný dokument, který obsahuje menší počet stejných slov, ale v podobném množství. Tuto situaci je možné vidět na výše uvedeném obrázku 3. Z hlediska kosinové vzdálenosti je však situace přesně opačná, protože určující je svíraný úhel, nikoliv délka vektorů.

2.4 Hodnocení úspěšností

Po naučení klasifikátoru je nutné nějakým způsobem změřit kvalitu jeho klasifikačních schopností. Proces učení je totiž často založen na minimalizaci chyby, které klasifikátor dosáhne na trénovacích dat. Zde však hrozí problém tzv. přeučení (overfitting), kdy výsledný klasifikátor dosahuje vynikajících výsledků na trénovacích datech, nicméně na testovacích datech bude jeho chybovost mnohem vyšší v důsledku nenaučení znalosti zobecnitelné i na data nepoužitá v procesu učení. Opačným problémem je naopak nedoučení (underfitting), kdy je učení klasifikátoru zastaveno příliš brzy a tudíž získá pouze malé množství znalosti. Zhodnocení kvality klasifikátoru se tedy provádí měřením jeho výkonnosti na testovacích datech. Pro tyto účely existují různé metriky, které umožňují vyjádřit kvalitu dosažených výsledků v číselné podobě a je tedy možné mezi sebou jednotlivé klasifikátory porovnávat. Pomyslným odrazovým můstkem pro výpočet některých metrik je kontingenční tabulka, v případě binární klasifikace o rozměrech 2×2, označovaná také jako *matice záměn.* Na jedné z os jsou zaznamenávány dokumenty podle jejich skutečné

Tabulka 2: Matice záměn pro binární klasifikátor pozitivních recenzí

	positive	negative
positive	TP	FP
negative	FN	TN

třídy a na druhé ose třídy, které byly dokumentům přiřazeny klasifikátorem (viz tabulka 2). Celkem tak mohou nastat čtyři způsoby klasifikace dokumentu. Pozitivní dokument může být označen správně jako pozitivní, v takovém případě jde o *True Positive* klasifikaci, nebo nesprávně jako negativní, kdy jde o *False Negative* klasifikaci. Obdobně pro negativní recenze ozančené správně jako negativní *True Negative* a nesprávně jako pozitivní *False Positive.* Počet správně klasifikovaných dokumentů se tak nachází na diagonále matice.

$$acc = \frac{TP + TN}{TP + FP + FN + TN}$$

[5] ve skutečnosti hodnoty převádí do jednotkového rozsahu, čímž dojde k eliminaci vlivu různých rozsahů

Základní metrikou pro ohodnocení celkové úspěšnosti klasifikace je *accuracy,* což je podíl správně klasifikovaných případů ze všech testovaných. Tento způsob měření výsledné přesnosti dokáže poskytnout základní představu o získaném klasifikátoru, avšak v některých případech může být tento způsob zhodnocení velice nepřesný. Pokud jsou jednotlivé třídy v testovací množině zastoupeny rovnoměrně, je hodnocení pomocí *acc* v pořádku. V případě značné převahy jedné ze tříd nad druhou však může nastat následující situace. Množina o tísici dokumentech obsahuje pouze deset pozitivních případů, které chceme pomocí klasifikátoru rozeznat (viz tabulka 3. Pokud bude klasifikátor zcela ignorovat existenci pozitivních případů a pro všechny případy určí jako výslednou třídu negativní, bude dosahovat velice vysoké 99% přesnosti, případně obrácením vnímání hodnoty pouze 1% chybovosti. Účelem však je správně rozeznat oněch 10 pozitvních dokumentů. Aby bylo možné podobné případy správně zhodnotit, je nutné použít i jiné metriky. Těmito metri-

Tabulka 3: Matice záměn klasifikace nevyvážené testovací množiny

	positive	negative
positive	0	0
negative	10	990

kami, které dokáží kvantifikovat úspěšnost klasifikátoru při klasifikaci pozitivních případů, jsou *precision* a *recall. Precision* udává poměr správně klasifikovaných pozitivních případů na všech případech klasifikovaných jako pozitivní. *Recall* naproti tomu vyjadřuje poměr správně klasifikovaných pozitivních případů na všech skutečně pozitivních případech.

$$prec = \frac{TP}{TP + FP} \qquad rec = \frac{TP}{TP + FN}$$

Použitím těchto metrik na zhodnocení výše popsaného klasifikátoru se tedy vyruší vliv drtivé většiny správně negativních dokumentů, protože v jejich výpočtu žádným způsobem nefigurují. Výsledné hodnoty jsou v obou případech nulové, na čemž je jasně vidět neschopnost klasifikátoru rozeznat pozitivní dokumenty. Zlepšení se dosáhne tím, že některé z dokumentů budou klasifikovány jako pozitivní. V případě nutnosti vyjádřit přesnost klasifikátoru pomocí jediné číselné hodnoty je možné použít metriku *F-measure*, která kombinuje dohromady precision a recall, a to jako jejich harmonický průměr.

$$F = \frac{2 \cdot prec \cdot rec}{prec + rec}$$

Pokud se klasifikuje do více než dvou tříd, je vhodné podobným způsobem vyjádřit kvalitu získané znalosti pomocí jedné hodnoty (Manning, Raghavan, Schütze, 2008). Toho se dosahuje tzv. *makroprůměrováním* nebo *mikroprůměrováním.* V případě makroprůměrování jsou pro každou třídu zvlášť spočítány jednotlivé metriky, které

jsou následně zprůměrovány. U mikroprůměrování jsou matice záměn pro jednotlivé třídy sečteny do jedné společné, ze které jsou následně spočítány hodnotící metriky. Pokud je některá ze tříd zastoupena ve větším množství, bude přenost dosahovaná nad touto třídou při mikroprůměrování převažovat nad přesnostmi klasifikace ostatních, méně zastoupených tříd. V takovém případě je vhodnější použít variantu makroprůměrování, protože do konečného výsledku přispívají úspěšnosti klasifikace jednotlivých tříd stejným dílem.

2.5 Paralelní zpracování

Paralelní zpracování v sobě skrývá možnost snížit celkovou časovou náročnost klasifikace na základě rozsáhlých kolekcí textových dat. Rozdělením celého problému na několik menších by mělo dojít k rapidnímu snížení času, potřebného pro natrénování klasifikátoru, protože výpočetní náročnost zpravidla neroste u těchto typu problémů lineárně. Paralení přístup již byl pro tyto účely použit, např. pro detekci webových útoků byl použit klasifikátor založený na podobnosti dokumentů, přičemž klasický sekvenční přístup nebyl schopen zpracovat příchozí data v reálném čase. Řešením problému bylo zparalelizování tohoto procesu (Ulmer et al., 2011). Dalším možným přístupem bylo vytvoření skupiny klasifikátorů na základě dílčích dimenzí rozsáhlé kolekce textových dat (Lertnattee, Theeramunkong, 2004). Na PEF byl realizován výzkum zaměřený na způsob rozdělení datové množiny na jednotlivé podmnožiny tak, aby byla co možná nejvíce zachována kvalita výsledné komise klasifikátorů vůči klasifikátoru naučeném na celém datovém souboru (Žižka, Dařena, 2012). Tento konkrétní výzkum byl jedním ze základních podkladů pro vypracování této práce.

3 Metodika zpracování

Výstupem diplomové práce je zrealizování několika klasifikačních experimentů nad větší kolekcí textových dat, které budou sloužit k ověření hypotézy o možnosti aplikování paralelizovaného přístupu při učení klasifikátorů nad rozsáhlými kolekcemi textových dokumentů. Pro získání potřebných výsledků bude nutné:

- sebrat dostatečné množství ohodnocených textových dat,
- zvolit testované algoritmy strojového učení,
- vybrat a rozdělit množinu dokumentů pro paralelní zpracování,
- předzpracovat vhodným způsobem textové dokumenty,
- naučit a otestovat klasifikátor na celé vybrané množině a na rozdělených podmnožinách.

3.1 Data

Základem pro uskutečnění potřebných experimentů je dostatečné množství textových dokumentů, vyjadřující určitý postoj autora textu nebo zabývající se nějakým tématem. Takových dat je na internetu dostupných poměrně velké množství, ale získání jejich dostatečně velkého počtu není zcela triviální. Data totiž musí být vhodně ohodnocena, aby mohla být použita k vytváření jednotlivých klasifikátorů pomocí tzv. *učení s učitelem.* Pro každý dokument je nutné znát jeho příslušnost do klasifikační třídy, a to i pro testovací data, jelikož výsledné klasifikátory je třeba otestovat na datech nepoužitých v procesu učení a porovnat třídu určenou klasifikátorem se skutečnou třídou dokumentu. Způsoby vytvoření potřebné kolekce dokumentů jsou celkem dva. První je využití některé z již existujících kolekcí textových dokumentů a druhý je sestavení vlastní nové kolekce.

3.1.1 Standardní zdroje textových kolekcí

V celé řadě vědeckých pracích zabývajících se problematikou dolování z textových dat, konkrétně kategorizací dokumentů, bývají velice často používány obecně známé (angl. well-known) a zároveň volně dostupné kolekce ohodnocených dokumentů (Cardoso-Cachopo, 2007).

20 Newsgroups

Tato kolekce dokumentů obsahuje zhruba dvacet tisíc příspěvků, které jsou téměř rovnomněrně rozděleny do dvaceti tříd představující různá témata, kterých se příspěvky týkají (baseball, hockey, electronics, atheism), přičemž příslušnost dokumentu do třídy je exkluzivní (patří vždy pouze do jedné třídy). Kromě samotného

textu obsahují dokumenty řadu meta informací, jako adresu a jméno odesílatele či příjemce, název vlákna nebo předmět zprávy (Rennie, 2008).

Reuters-21578

Další často používaná kolekce Reuters-21578 obsahuje celkem 21578 dokumentů, z nichž zhruba polovina je ohodnocena alespoň jednou třídou. Z celé kolekce je možné použít dvě menší kolekce *R10* a *R90* obsahující dané dokumenty rozdělené do odpovádajícího počtu tříd (např. grain, livestock, coffe, cpu). Dokumenty jsou uložené v SGML formátu, takže obsahují navíc strukturní značky (Lewis, 2004).

WebKB

Kolekce WebKB obsahuje internetové stránky sebrané z různých univerzit. Celkem je k dispozici přes 8 tisíc stránek, které byly ručně klasifikovány do následujících kategorií: student, faculty, staff, department, course, project, a other. Stránky jsou uloženy přímo v HTML formátu, takže podobně jako u Reuters kolekce obsahují strukturní značky. Navíc je na začátku každého dokumentu uvedena MIME hlavička. Některé z uložených stránek tak mohou obsahovat pouze informace pro prohlížeč o přesměrování na jinou adresu (Cardoso-Cachopo, 2007).

Výhoda použití některé z uvedených kolekcí spočívá ve snadné srovnatelnosti získaných výsledků s ostatními výzkumy, které tyto datové kolekce při svých experimentech použily. Některé jsou navíc dostupné i v určitém prezpracovaném stavu, kupříkladu jsou již odstraněna stop slova nebo byl aplikován stemming, což může usnadnit proces přípravy dat. Pro potřeby této práce jsou však tyto jinak velice vhodné kolekce nepoužitelné, a to kvůli příliš malému množství dokumentů. Pro vytvoření dostatečně rozsáhlé kolekce dat bude tedy nutné zvolit jiný způsob.

3.1.2 Vytvoření nové kolekce

Pokud z nějakého důvodu není možné použít pro experimenty standardně používané kolekce dokumentů, musí být použita některá prozatím obecně nerozšířená kolekce nebo vytvořena zcela nová. Zdrojem textových dat s ohodnocením mohou být nejrůznější internetové stránky, na kterých mohou návštěvníci psát do diskuzí, komentářů či hodnocení produktů a zároveň přidat ještě jiný způsob určení významu napsaného textu, např. plusové a minusové body nebo počty hvězdiček. Celosvětově známé internetové stránky jako internetová filmová databáze IMBb nebo třeba elektronický obchod Amazon mají v současné době obrovskou základnu uživatelů, kteří píší recenze na filmy či produkty a poskytují tak nemalé množtsví textových dat ohodnocených počtem hvězdiček, přiřazením k produktu či filmu určité kategorie, apod. Mohou tak sloužit jako dobrý zdroj pro získání dostatečného množství ohodnocených dat psaných v přirozeném jazyce.

Většinou však není časově nebo i finančně možné ručním způsobem procházet podobné stránky a ukládat jednotlivé recenze do nové kolekce, která by měla obsaho-

vat řádově několik desítek, ne-li stovek tisíc dokumentů. Proces je možné zautomatizovat vytvořením programu, který bude na základě url stránky postupně stahovat, extrahovat z nich potřebné údaje a ukládat je ve vhodném formátu pro následné použití při experimentech.

Tabulka 4: Charakteristika dat hotelových recenzí

Nejkratší recenze	1 slovo
Nejdelší recenze	400 slov
Průměrná délka recenze	24 slov

V této diplomové práci by byl normálně zvolen druhý způsob, protože standardní textové kolekce bohužel neobsahují postačující počet dokumentů, který je pro realizaci zamýšlených experimentů bezpodmínečně nutný. Vedoucím práce však byla dána k dispozici již zkompletovaná kolekce dokumentů (Žižka, Dařena, 2010), která má požadované vlastnosti, tedy ohodnocení textových dat a dostatečné množství dokumentů. Jedná se o recenze hotelů psaných v anglickém jazyce a dle jejich obsahu jsou rozděleny do dvou tříd, pozitivní a negativní recenze, přičemž jejich příslušnost do daných tříd určovali přímo autoři recenzí. Nejzákladnější charektristiky těchto dat jsou uvedeny v tabulce 4.

3.2 Algoritmy strojového učení

Na konci předchozí kapitoly byly krátce představeny algoritmy, které se pro klasifikaci textových dat používají nejčastěji. Experimenty budou provedeny pouze s několika vybranými metodami, a to:

- neuronové sítě,
- podpůrné vektory,
- nejbližší soused.

Důvodem jejich výběru je především časová náročnost, kterou by mělo být možné díky paralelnímu přístupu výrazně snížit. Důležitým faktorem však je, jaký bude vliv na celkovou úspěšnost získaných klasifikátorů a který ze způsobů rozdělení celkové množiny dat na podmnožiny bude poskytovat nejlepší výsledky.

3.3 Výběr množiny dat a její rozdělění

Základem pro srovnání vlivu paralelizace dolování z textových dat je rozsáhlá množina dokumentů, na kterou budou aplikovány algoritmy strojového učení, a to jednak na celou množinu a poté na jednotlivé podmnožiny, na které bude vybraná množina rozdělena.

Výsledné podmnožiny by měly zachovat poměr, v jakém se nachází zastoupení jednotlivých tříd v celé množině. V ideálním případě by už při výběru hlavní množiny mělo být zajištěno rovnocenné zastoupení všech tříd, aby byly klasifikační algorimty natrénovány na stejném množství pozitivních i negativních případů a dokázaly se tak naučit co nejlépe mezi oběma případy rozlišovat. To však v praxi často nemusí být možné a mělo by tedy před rozdělením nejdříve dojít ke zjištění poměru, v jakém jsou třídy zastoupeny a rozdělení na podmnožiny provést obdobným způsobem. Testování získaných klasifikátorů musí následně probíhat na shodné množině testovacích dokumentů, která bude z hlediska její velikosti vytvářena v určitém předem daném poměru vůči hlavní trénovací množině.

3.4 Technologie pro implementaci

Celý proces průběhu experimentů je třeba rozdělit na dvě hlavní části:

- příprava dat,
- učení a testování klasifikátoru.

Vzhledem k obrovskému rozsahu zpracovávaných dat není možné ručně provádět jejich výběr, či dokonce převod dokumentů do vektorové podoby a přepočítání vah jednotlivých příznaků vektoru. Pro tyto účely bude nutné vytvořit jednoduchou aplikaci, která se postará o výběr dat, jejich následné rozdělení na podmnožiny a převedení textů do vektorové reprezentace. Použitý programovací jazyk by tedy měl umožňovat pokud možno co nejjednodušší práci s textovými daty a soubory, a to pokud možno bez omezení na velikost zpracovávaných dat. Pro tyto účely je velice vhodný programovací jazyk Perl, který vznikl právě jako náhrada prostředků pro zpracování textu, které v době jeho vzniku již nebyly pro tyto účely dostačující (Dařena, 2005). Předzpracování textových dokumentů bude tedy realizováno pomocí několika skriptů.

Naučení klasifikátorů a otestování jejich znalosti vyžaduje obdobný přístup. Je třeba vytvořir aplikaci, která zpracuje předpřipravená data pomocí zvoleného učícího algoritmu a otestuje výsledný klasifikátor. Veškerý průběh běhu programu je nutné zaznámenávat do souboru (pro pozdější kontrolu dosažených výsledků). Stejně tak testovací část by měla předpovědi klasifikátoru zaznamenat do souboru, aby bylo možné po provedení všech potřebných experimentů výsledky porovnat a vyhodnotit. Požadavky na použitý programovací jazyk tedy budou poněkud odlišné:

- existence knihoven pro algoritmy strojového učení – i když je možné provést vlastní implementaci vybraných algoritmů, není podobný postup nejlepším možným řešením, protože odladění korektního fungování by bylo časově náročné a některé možné chyby, které by mohly mít negativní vliv na získané výsledky, by se nemuselo podařit ani odhalit, je tedy nutné vybrat takový programovací jazyk, pro který jsou již tyto algoritmy implementovány v podobě rozšířených a uživateli ověřených knihoven,

- objektově orientovaný přístup programování,
- rychlost – výstupem vybraného jazyk by měla být aplikace běžící co nejrychleji, ideálně by tedy mělo jít o jazyk kompilovaný, protože během procesu učení klasifikátorů dochází díky rozsáhlosti zpracovávaného souboru dat (očekává se zhruba sto tisíc dokumentů reprezetovaných vektory s více jak deseti tisíci atributy) k značnému prodlužování doby učení a také i doby potřebné pro odladění celé aplikace a získání konečných výsledků,
- možnost práce s pamětí – stejně jako má velikost zpracovávaných dat vliv na dobu učení, roste i náročnost na paměť, takže vybraný jazyk by měl v případě nutnosti umožňovat správu používané paměti,
- typově striktní – neočekávané chyby, které by mohly také negativně ovlivnit získané výsledky, je eliminovat díky striktní typové kontrole,
- spustitelnost na libovolném operačním systému,
- autorova znalost používání jazyka.

Dle takto stanovených požadavků byl pro implementaci zvolen programovací jazyk C++. Ten je velice rozšířený a umožňuje tak výběr z řady již používaných a ověřených knihoven, které je možné použít pro implementaci klasifikátorů.

3.4.1 Výběr knihoven

Pokud by měla být provedena implementace všech potřebných programových prostředků, které jsou nutné pro realizaci experimentů, bylo by nutné věnovat velké množství času na odladění různých částí aplikace a hledání možných chyb, které by mohly ovlivnit získané výsledky. Z tohto důvodu budou pro některé kritické části aplikace použity knihovny, jejichž použití by mělo zaručit

- oddladěné a ověřené části zdrojového kódu,
- implementaci od odborníků z dané oblasti,
- časovou úsporu nutnou pro implementaci,
- možnost řešení problémů v rámci komunity používající knihovnu,
- zpracovanou dokumentaci.

Použití knihoven přináší řadu výhod a jediným problémem může být absence určité požadované funkcionality, kterou je třeba vlastními silami doprogramovat.

Příprava dat

V první části, určené pro přípravu a předzpracování textových dokumentů, bude možné využít programový modul *TextMining,* jehož autorem je vedoucí této diplomové práce (Žižka, Dařena, 2010). Tento modul poskytuje jednoduché rozhraní pro

převod textových dokumentů do vektorové reprezentace. Při transformaci textových dat je možné volit z celé řady vážení hodnot ve výsledných příznakových vektorech, a to jak pro lokální váhy, tak i pro globální váhy a normalizaci hodnot. Je možné odfiltrovat i slova s nízkým nebo naopak příliš vysokým počtem výskytů, opět na více úrovních – jednak v rámci jednotlivých dokumentů, v rámci celého datového souboru, nebo podle počtu dokumentů, ve kterých se jednotlivá slova vyskytují. Důležitou součástí je vytváření slovníků nad zpracovávanou množinou dokumentů, které je pak možné při přípravě jiné množiny textových dat načíst, čímž dojde ke sjednocení reprezentace těchto dvou množin pomocí vektorů se stejnými atributy. Jediný požadavek je kladený na formát souboru vstupních dat, které musí mít následující tvar.

```
CLASS   \t  DOCUMENT
-------------------------------------------------------------
NEGATIVE    This is very dissapointing movie, don't waste your
            time with it.
POSITIVE    Absolutely brilliant piece, perfect cast, gorgeous
            visuals and very catching story.
```

Na každém řádku vstupního souboru se nachází dvojice hodnot třída a samotný dokument vzájemně od sebe oddělené znakem tabulátoru. Transformovaná data je možné ukládát do několika různých formátů, např. pro algoritmus *C5* generující rozhodovací strom, aplikaci *Weka* pro dolování z dat vyvinuté na univerzitě Waikato, nebo programový balík pro shlukovací úlohy *CLUTO.*

Využití tohoto modulu je možné dvěma způsoby. Jednak jej lze zahrnout do vlastních skriptů a využít poskytnuté rozhraní pro nastavení parametrů převodního algoritmu a spuštění transformace dat. Druhým způsobem je použití i grafické nadstavby nad tímto modulem, implementované pomocí modulu pro grafické uživatelské rozhraní *Perl/Tk.* Tato aplikace umožňuje uživatelsky přívětivé zadávání parametrů a také validaci zadaných hodnot (Novák, Dařena, 2012).

Realizace experimentů

Druhou část implementace bude tvořit aplikace pro realizaci samotných experimentů. Zde budou kritickou část tvořit jednotlivé algoritmy strojového učení, které musí být pro získání nezávadných výsledků řádně odladěny a zbaveny všech chyb, které by mohly způsobit zkreslení výsledných hodnot. Je tedy přirozené sáhnout po již existujících knihovnách, které poskytnou kvalitní a ověřenou implementaci těchto algoritmů.

Prvním použitým algoritmem jsou umělé neuronové sítě. Na internetové stránce *sourceforge.net* sloužící jako úložistě pro open source projekty je dle jednoduchého vyhledávacího dotazu dostupných zhruba dvacet projektů, které implementují neuronové sítě pro programovací jazyk C++. Na výběr je tedy z poměrně velkého množství a popis všech těchto možností by byl zbytečně dlouhý. Autor práce má již

zkušenosti s používáním knihovny *Fast Artificial Neural Network* (zkráceně FANN), která je implementována v programovacím jazyku C. Dostupná je ale celá řada bindings pro ostatní programovací jazyky, včetně používaného C++. Dle domovské stránky[6] projektu poskytuje tato knihovna celou řadu funkcí:

- vícevrstvá architektura sítě,
- čtyři různé učící algoritmy,
- plné či částečné propojení neuronů,
- podpora dynamicky se upravující topologie,
- snadná použitelnost,
- vysoká rychlost (údajně až 150 násobek rychlosti jiných knihoven),
- rychlé ukládání a načítání natrénovaných sítí,
- nadstavby s GUI,
- již zmíněné bindings do velkého množství programovacích jazyků.

Tato knihovna tedy slibuje opravdu kvalitní implementaci s různými možnostmi nastavení topologie neuronové sítě, způsobu propojení neuronů i různých učících algoritmů. Pokud by tedy měla výsledná aplikace sloužit jako podklad pro další výzkum a experimenty, může být použita řada těchto rozličných možností poskytované knihovnou FANN. Největším kladem je však slibovaná rychlost knihovny. Největší nevýhodou při používání neuronových sítí při zpracování textových dat je totiž právě velká časová náročnost. Jakýkoliv způsob urychlení celého procesu je tedy vítaný a pro implementaci klasifikátoru neuronové sítě bude použita knihovna FANN.

Dále bude testován vliv paralelizace zpracování dat na algoritmus podpůrných vektorů. Na internetovém portálu[7] *support vector machines* je k dispozici velice rozsáhlý seznam knihoven a softwaru implementující podpůrné vektory, a to nejen pro programovací jazyk C a C++. Uveden je zde třeba již dříve zmíněný balík algoritmů pro dolovaní z dat Weka, Java applety, toolbox pro Matlab, jazyk R či Python. Nechybí zde ani poměrně známá knihovna *libsvm*[8] opět poskytující rozhraní do celé řady jiných programovacích jazyků. Na domovské stránce knihovny je také uveden výčet některých význačných vlastností implementace:

- efektivní klasfikace do více tříd,
- krosvalidace,
- různé jádrové (kernel) funkce,

[6] `http://leenissen.dk/fann/wp/`
[7] `http://www.support-vector-machines.org/`
[8] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

- demonstrativní GUI aplikace,
- automatické určení parametrů.

Na prvním místě v seznamu SVM softwaru je však uvedena knihovna *SVM light*[9] jejíž autorem Thorsten Joachims, který se zasadil rozšíření použitelnosti podpůrných vektorů pro klasifikiaci textových dat. Tato knihovna se řadí mezi nejpoužívanější. Poskytuje vysoce optimalizovanou implementaci, která umožňuje její použití pro velice rozsáhlé datové soubory. K dispozici jsou také různé jádrové funkce:

- lineární,
- polynomiální,
- radiálně bázová,
- sigmoid tanh,
- uživatelem definovaná funkce.

Vzhledem k tomu, že Joachims používal tuto knihovnu pro řadu vědeckých článků zabývajících se klasifikací textových dat, bude pro implementaci tohoto algoritmu použita právě knihovna SVM light, která byla po konzultaci doporučena i vedoucím práce.

Posledním algoritmem strojového učení je metoda nejbližšího souseda. Problém při klasifikaci textových dat pomocí tohoto přístupu spočívá obdobně jako u neuronových sítí ve velké časové náročnosti, která roste s množstvím zpracovávaných dokumentů a současně s velikostí slovníku. Implementace samotného algoritmu nemusí být nikterak složitá, protože dochází pouze k postupnému srovnávání dotazovaného vektoru se všemi vektory uloženými v databázi. Tento základní přístup však trpí zmíněnou časovou náročností. Proto existují modifikace, které umožní zmenšení počtu záznamů, se kterými je dotazovaný vektor porovnáván. Pro výběr těchto vektorů jsou použity K-D stromy, které rozdělí prostor na tzv. hyperkvádry a klasifikovaný vektor je porovnáván pouze s případy ze stejného hyperkvádru. Knihovna *ANN: A Library for Approximate Nearest Neighbor Searching*[10] implementuje tento typ nejbližších sousedů. Bude použita pouze jako předloha pro vlastní implementaci tohoto typu klasifikátoru.

[9] `http://svmlight.joachims.org/`
[10] `http://www.cs.umd.edu/~mount/ANN/`

4 Výsledky

Výsledkem této diplomové práce by měly být závěry odvozené z výsledků získaných provedením několika experimentů, které jsou zaměřeny na srovnání dolování znalostí z rozsáhlé textové kolekce a několika menších dílčích kolekcí vzniknuvších rozdělením původní rozsáhlé kolekce dokumentů. Experimenty budou realizovány skrze aplikaci, která je pro tyto účely implementována, stejně jako skripty vytvořené pro usnadnění přípravy textových dat. Výstupy práce jsou tedy:

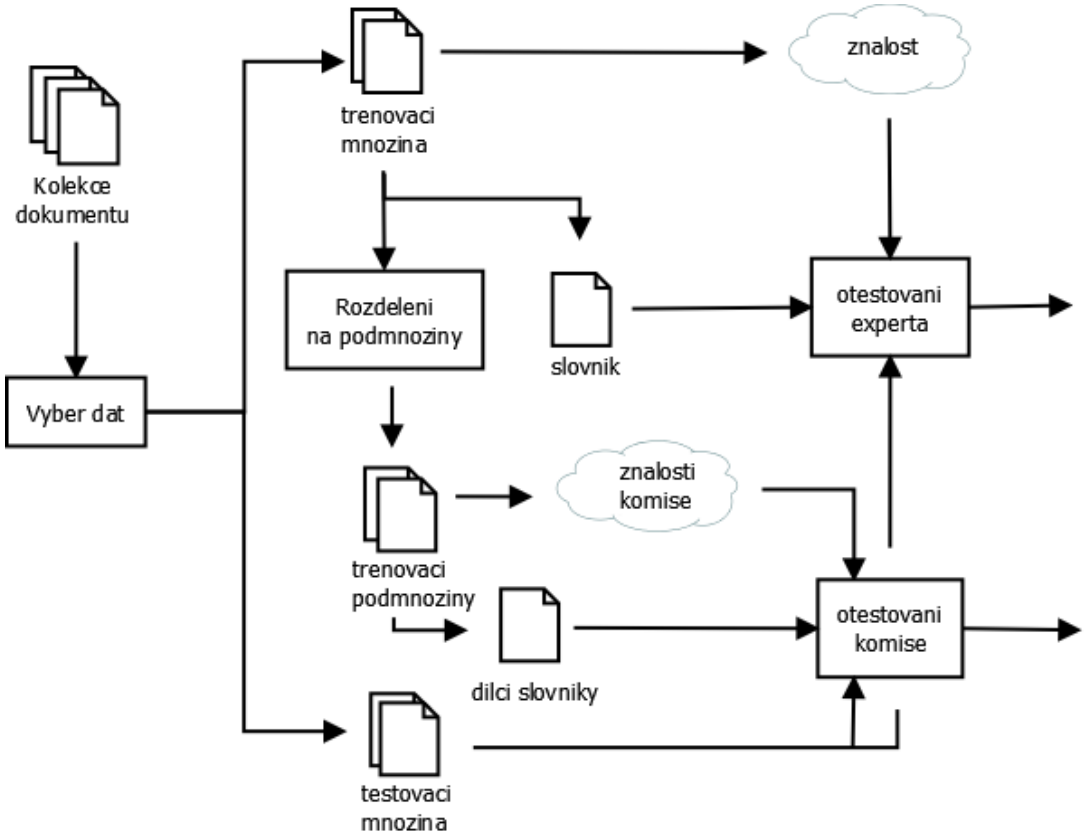
- výsledky experimentů,
- skripty pro přípravu dat,
- aplikace pro realizaci experimentů.

4.1 Návrh experimentů

Cílem experimentů je prověřit vliv paralelizovaného přístupu při dolování znalostí z rozsáhlých kolekcí textových dat. Především pak jaký bude mít vliv na výslednou klasifikaci různé rozdělení trénovacích dokumentů na menší množiny a jaké budou rozdíly mezi jednotlivými algoritmy strojového učení, použité pro tvorbu klasifikátoru. Důležitý by také mohl být vliv různých nastavení pro jednotlivé typy klasifikátorů, např. jiná architektura neuronové sítě nebo jiný typ jádrové funkce u podpůrných vektorů. Na obrázku 9 je znázorněno schéma průběhu experimentu. Z dostupné kolekce dokumentů je nutné nejdříve vybrat vhodné množství dat, které je poté rozděleno na trénovací a testovací množinu. Určující je požadovaný počet dokumentů v trénovací množině, takže testovací dokumenty jsou vybírány poměrně vůči počtu trénovacích dokumentů, a to v poměru 1 : 3, takže na jeden testovací dokument připadají tři trénovací dokumenty. Trénovací množina je následně celá použita pro natrénování jednoho klasifikátoru, který představuje klasický přístup ke klasifikaci textových dokumentů. Zároveň je rozdělena na vzájemné disjunktní podmnožiny, nad kterými jsou opět naučeny jednotlivé klasifikátory, které vytvoří komisi. Z každé trénovací množiny je vytvořen slovník, pomocí kterého jsou transformována testovací data a příslušný klasifikátor je na těchto datech otestován. V případě komise je konečná třída neznámého dokumentu určena dle rozhodnutí většiny.

4.2 Návrh aplikace pro realizaci experimentů

Aplikace pro realizaci experimentů musí sloužit jako nástroj, který umožní jeho uživateli snadno definovat, jaký druh klasifikátoru bude pro klasifikaci použit a jaké hodnoty parametrů, kterými je možné ovlivnit průběh trénování a testování daného klasifikátoru, budou nastaveny. Také musí být snadno nastavitelné, na jakých datech bude klasifikátor trénován a na kterých datech bude také testován. Veškerý průběh celého experimentu je nutné nějakým způsobem zaznamenávat, takže je také nutné



Obrázek 9: Schématické znazornění hlavních kroků experimentu. Expertem je jeden klasifikátor naučený na celé trénovací množině a komisi tvoří několik klasifikátorů naučených na trénovacích podmnožinách.

umožnit nastavení parametrů, jako název souboru pro uchování predikcí naučeného klasifikátoru nebo název souboru pro uložení získaného klasifikátoru. Aplikace by měla všechna tato nastavení načíst, vytvořit odpovídající typ klasifikátoru, nastavit zadané parametry klasifikátoru, spustit učení nad zadanými trénovacími daty a po ukončení trénovací fáze otestovat naučený klasifikátor na souboru s testovací množinou dokumentů. Veškeré výstupy na standardní a chybový výstup by měly být zaznamenány do logovacího souboru pro pozdější zkontrolování průběhu celého experimentu. Zvlášť by pak měly být uchovávány výsledky z testování získaného klasifikátoru, a to v podobě souboru obsahujícího informace o třídě, do které testovací dokument opravdu patří, a třídě predikované daným klasifikátorem, aby mohly být po skončení experimentu vypočítány hodnotící metriky úspěšnosti.

4.3 Implementace

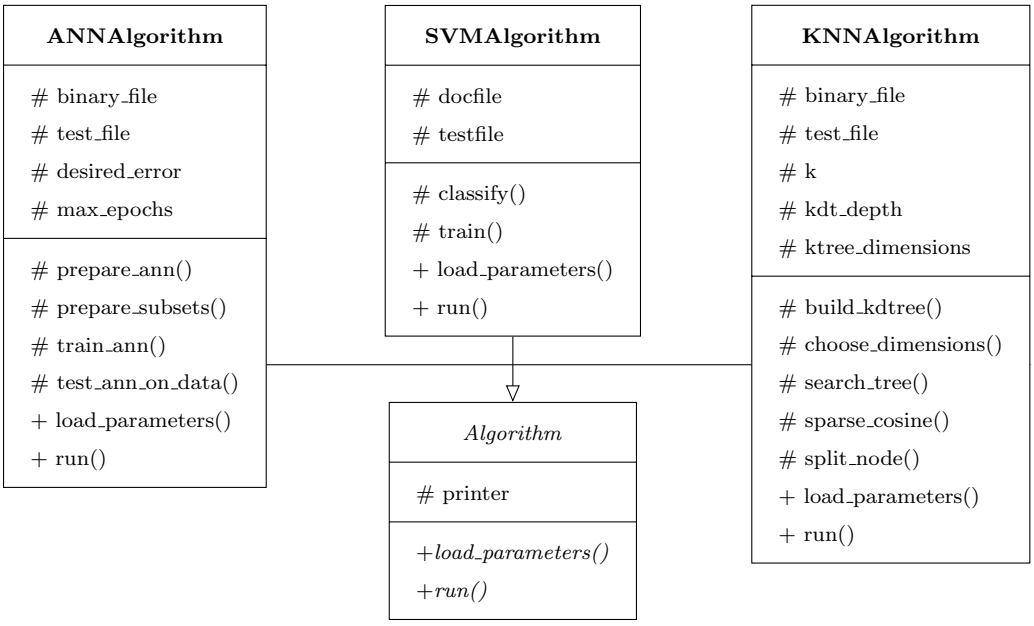
K implementaci byl vybrán programovací jazyk C++ a jedním z důvodů tohoto výběru byla podpora objektového přístupu daným jazykem. Implementace aplikace tedy bude krátce popsána z hlediska jednotlivých objektů, které celou aplikaci tvoří.

Experiment

Základní třída sloužící pro spuštění celého experimentu. Stará se o vytvoření objektů pro tisk zpráv a zpracování konfiguračního souboru, ve kterém jsou uložena veškerá nastavení konkrétního experimentu. Z tohoto souboru přečte pouze typ používaného klasifikátoru a dle zadaného nastavení vytvoří objekt pro daný algoritmus strojového učení. Po jeho vytvoření je algoritmus spuštěn.

Algorithm

Algorithm je abstraktní třídou, pomocí které jsou u tříd pro konkrétní algoritmy strojového učení vynucovány implementace dvou základních metod, volaných skrze metody třídy Experiment. První metoda slouží pro načtení parametrů z konfiguračního souboru, jejichž část bývá zpravidla specifická pro daný algoritmus. Druhou metodou je implementováno spuštění samotného algoritmu. Všechny následující třídy reprezentující konkrétní algoritmy jsou potomky této třídy (viz obrázek 10).



Obrázek 10: Část třídního diagramu zobrazující třídy reprezentující jednotlivé algoritmy. Kvůli rozsahu jsou uvedeny pouze vybrané metody a atributy bez datových typů a parametrů.

ANNAlgorithm

Třída reprezentující algoritmus neuronových sítí umožňuje natrénování a otestování modelu sítě oproti datovému souboru. Pro samotný klasifikátor je používána knihovna FANN, rozšířená o některé funkcionality, které knihovna sama o sobě neposkytuje.

SVMAlgorithm

Tato třída slouží pro reprezentaci klasifikátoru podpůrných vektorů. V implementaci je použita knihovna SVM light. Archiv, který je dostupný na domovské stránce této knihovny, obsahuje zdrojové kódy pro knihovnu samotnou, ale také pro programy na naučení a otestování podpůrných vektorů. Těla metod `train` a `classify` jsou převzata ze souborů se zdrojovým kódem těchto programů a jsou pouze vhodně upravena do objektové podoby.

KNNAlgorithm

Třída pro metodu nejbližšího souseda obstarává porovnání testovacích dokumentů s trénovacími, výpočet vzdálenosti mezi dvěma vektory reprezentující porovnávané dokumenty, výběr k nejbližších sousedů a určení predikované třídy na základě příslušnosti do tříd u nejméně vzdálených dokumentů.

Printer

Velice jednoduchá abstraktní třída sloužící jako předloha pro třídy implementující jednotný způsob výpisu zpráv z průběhu experimentu. Obě následující třídy jsou potomky této třídy, která vynucuje implementaci metod info, debug, warn a error.

TerminalPrinter

Tento potomek třídy Printer implementuje výpis zpráv na standardní výstup a chybových zpráv na standardní chybový výstup. Každému ze čtyř typů zprávy je možné nastavit hlavičku, která bude vždy předcházet vypisovanému textu zprávy.

Log4cPrinter

Sofistikovanější způsob výpisu zpráv, který veškerý výstup zároveň ukládá do souboru, implementuje tato třída, a to pomocí knihovny *Log4cplus*[11] určené pro logování výstupu programu. Nastavení celého logování se provádí pomocí konfiguračního souboru. Ten je do programu předán jako jeden z obecných parametrů v konfiguračním souboru celého experimentu.

Skripty pro přípravu dat

Skripty pro přípravu dat slouží pro hromadné zpracování a přípravu kolekce textových dokumentů. Skripty pro přípravu `prepare_fann_data` a `prepare_svm_data` zajistí vytvoření potřebných datových souborů pro realizaci experimentu na základě velikosti trénovací množiny a počtu dílčích podmnožin. Při svém běhu volají skripty `split`, sloužící pro rozdělení dat na podmnožiny, a `vectorize_fann_data` nebo

[11] `http://log4cplus.sourceforge.net/`

`vectorize_svm_data`, které používají rozšířený modul TextMining pro převod textových dat do vektorové podoby.

4.3.1 Adresářová struktura

Aplikace je do jisté míry implementována na pevně danou adresářovou strukturu, ale názvy jednotlivých složek a jejich umístění je možné snadno změnit, a to skrze hodnoty v konfiguračním souboru experimentu. Výchozí struktura je znázorněna následující:

```
+-/
    +-CPP/
    |   +-config/
    |   +-logs/
    |   +-results/
    |   +-src/
    |       +-include/
    |   +-convert
    |   +-a.out
    +-output/
    +-data
        +-_texts.text
        +-split/
    +-experiment.sh
    +-generate_config_files.pl
    +-prepare_fann_data.sh
    +-prepare_svm_data.sh
    +-results.pl
    +-split.pl
    +-vectorize_fann_data.pl
    +-vectorize_svm_data.pl
```

Několik perlovských a shellovských skriptů slouží pouze pro přípravu dat a spouštění sady experimentů. Při jejich implementaci již byla tato adresářová struktura brána jako závazná a tudíž jsou potřebné datové soubory očekávány ve složkách `data` a `data/split` nebo názvy a umístění programů `convert` a `a.out` ve složce `CPP`.

4.3.2 Rozšíření modulu TextMining

Transformace textových dokumentů do vektorové podoby spolu s přepočítáním vah jednotlivých termů a následným uložením do souboru s učitým formátem obstarává programový modul *TextMining*. Bohužel neobsahuje podporu pro formáty, které jsou vyžadovány knihovnami použitými pro jednotlivé klasifikátory. Aby tedy bylo možné modul použít, je nutné rozšířit jeho podporované výstupní formáty.

Formát FANN

Knihovna FANN používá pro uložení datových souborů vlastní formát textového souboru, kde se na prvním řádku nachází mezerou oddělené základní informace o datovém souboru, a to celkový počet příkladů, počet atributů vstupního vektoru a počet atributů výstupního vektoru. Dále již jsou v souboru uvedeny jednotlivé příklady, vždy jako dvojice řádků, kde na prvním je uveden vstupní vektor a na druhém výstupní vektor. Celkový počet řádků souboru je tedy $2n + 1$, kde n je počet příkladů. Veškeré hodnoty vektorů jsou od sebe odděleny jednou mezerou. Zde je uveden obecný zápis a jednoduchý příklad datového souboru pro funci XOR:

```
data_length num_input num_output            4 2 1
input-vector_1                              1 1
output-vector_1                             1
input-vector_2                              0 1
output-vector_2                             0
...                                         ...
input-vector_n                              0 0
output-vector_n                             1
```

Formát FANN-SPARSE

Vzhledem k rozsáhlosti jednotlivých vektorů, které mohou mít více jak deset tisíc atributů z nichž drtivá většina nese nulovou hodnotu, je mnohem efektivnější ukládat datové soubory v tzv. řídkém formátu. Ukládány jsou pouze nenulové hodnoty, a to jako dvojice číslo atributu a hodnota. Formát souboru tedy zůstává totožný s původním formátem, pouze jednotlivé vektory jsou uloženy v řídkém formátu, přičemž všechny číselné hodnot jsou od sebe odděleny mezerou.

Formát SVMlight

Knihovna SVM light vyžaduje datové soubory uloženy také ve svém vlastním formátu. Jelikož však byla implementována i za účelem klasifikace textových dokumentů pomocí podpůrných vektorů, je tomuto faktu uzpůsoben i samotný formát datového souboru. Stejně jako v případě knihovny FANN jde o textový soubor, který na každém řádku uchovává jeden vektor. Vektory jsou však rovnou vyžadovány v řídkém formátu, což eliminuje prostorovou náročnost ukládání textových dokumentů ve vektorové reprezentaci. Zde je uveden obecný formát zápisu jednoho příkladu a konkrétní příklad funkce XOR:

```
class index:value index:value index:value ...
+1 0:1 1:1
-1 0:0 1:1
-1 0:1 1:0
+1 0:0 0:0
```

Na začátku každého řádku je tedy nutné uvést třídu, do které příklad patří, v případě binárního klasifikátoru jsou možnými třídami pouze hodnoty $+1$ a -1. Dále jsou již zapsány pouze dvojtečkou oddělená čísla index atributu a hodnota, přičemž tyto jednotlivé dvojice jsou od sebe navzájem odděleny mezerou.

Slovník s parametry pro vážení

Pokud je klasifikátor naučen na konkrétní množině trénovacích dokumentů, které byly při transformaci do vektorové reprezentace určitým způsobem váženy, je nutné stejným způsobem předzpracovat i testovací množinu. Jednoduchým přístupem by bylo vytvořit jednu velkou množinu dat sloučení trénovací i testovací, případně i validační množiny, a tu zpracovat najednou. Tím by však došlo ke stanovení parametrů pro vážení hodnot termů i na základě dokumentů v testovací množině, což není žádané, jelikož určující je pouze trénovací množina. Pokud by například testovací množina obsahovala slovo, které by se ve trénovací množině nevyskytovalo, došlo by v případě zpracování jedné velké množiny dokumentů k zahrnutí tohoto slova do příznakového vektoru, ale nikdy by tento příznak nebyl při trénování použit, protože se v trénovacích datech nevyskytuje žádný dokument, který by mohl danému atributu přidělit nějakou hodnotu. Pokud však budou trénovací data upravena na základě parametrů pouze z trénovací množiny, dojde k vyřazení tohoto pro klasifikátor neznámého slova. Obdobná logika platí i pro parametry pro vážení jednotlivých termů.

Použitý modul umožňuje zpracovávat textové dokumenty na základě zadaného slovníku, kdy při převodu do vektorové reprezentace dojde k vyřazení těch slov, která se ve slovníku nenachází. Bohužel však nejsou přenášeny vážící parametry. Modul byl tedy rozšířen o možnost generování slovníku, který pro jednotlivá slova uchovává také potřebné vážící parametry, jako celkový výskyt slova v kolekci, minimální a maximální hodnota výskytu slova apod. Při zadání tohoto upraveného slovníku nejsou tedy tyto hodnoty zjišťovány ze zadaného vstupního datové souboru, ale ze slovníku, čímž je zajištěna shodná reprezentace hodnot trénovací i testovací množiny.

4.3.3 Rozšíření knihovny FANN

Knihovna FANN poskytuje pro práci s neuronovými sítěmi pouze dvě třídy, které reprezentují:

- neuronovou síť,
- použitá data.

Aby nebylo nutné výrazně zasahovat do kódu knihovny, jsou od těchto dvou tříd odvozeni potomci, kteří implementují potřebná funkční rozšíření.

Rozdělení dat a normalizace

Data pro neuronové sítě je třeba rozdělit na celkem tři disjunktní množiny, a to trénovací, testovací a validační data. Knihovna FANN umožňuje načtení jednoho datového souboru pro proces učení, pro testování je poté možné použít zcela jiný soubor. V zájmu zachování původní podoby funcí této knihovny, budou tyto tři množiny zahrnuty v jednom jediném souboru. Data bude tedy nutné interně vhodně rozdělit. Pro tyto účely bylo implementováno několik funkcí, které jednak vypočítají hranice jednotlivých množin, umožní výběr aktuálně používané množiny a práci s určitou podmnožinou jako s jednolitým datovým souborem.

Při používání neuronových sítí je nutné, stejně jako u ostatních typů klasifikátorů, řešit normalizaci dat. Pokud by data zůstala ve svém původním tvaru, pro síť by bylo mnohem náročnější vhodně upravovat váhy vazeb mezi neurony tak, aby chyba zůstala co nejmenší. Normalizaci dat je možné použít již při převádění dokumentů do vektorové reprezentace. Jelikož by však kvůli rozdělování dat není při předzpracování dat předem jasné, které příklady budou patřit do jaké množiny, je i normalizace prováděna přímo při běhu aplikace.

Implementace tedy rozšířila možnosti knihovny FANN o následující funkcionalitu. Normalizaci dat je možné zapnout zavoláním metody, která vypočítá a nastaví normalizační parametry pro všechny jednotlivé atributy vstupních dat, a to na základě aktuálně vybrané množiny. Pokud nejsou data rozdělena, jsou parametry vypočítány z celého datového souboru. Těmito hodnotami jsou:

- minimální hodnota,
- maximální hodnota,
- suma hodnot,
- rozsah hodnot,
- průměrná hodnota.

Normalizovaná hodnota je vypočítána jako podíl mezi rozdílem původní hodnoty a průměrné hodnoty a rozsahu hodnot (rozdíl maximální a minimální hodnoty) pro daný atribut.

$$x'_{i,j} = \frac{x_{i,j} - avg_j}{max_j - min_j}$$

Data jsou do normalizované hodnoty převáděna vždy při jejich výběru z datového souboru. Kombinací výběru trénovací množiny a následným zavoláním normalizační metody je tedy dosaženo aplikování normalizačních parametrů pouze z trénovací množiny na data v ostatních množinách. Pokud je nutné normalizovat data ze zcela jiného datového souboru, je možné mezi nimi předat momentálně používané normalizační parametry.

Validace během trénování

Při učení neuronových sítí může dojít k celkem dvěma negativním výsledkům. Prvním je nedoučení, kdy dojde k tomu, že naučená síť nedokázala během učení dostatečně zobecnit znalost skrytou v trénovacích datech, protože byl tento proces ukončen příliš brzy, nejčastěji v důsledku příliš vysoké hodnoty požadovanové chyby. Druhým případem je přesný opak. Síť se učí na trénovacích datech tak dlouho, až začne ztrácet schopnost generalizace skryté znalosti a ze sítě se stává expert pouze na trénovací data. Na testovacích datech tak bude v obou případech síť dosahovat zpravidla nízké přesnosti. Zatímco nedoučení je možné relativně snadno vyřešit změnou parametrů, zamezení přeučení sítě je o něco složitější. Učení je třeba zastavit ve chvíli, kdy se začne zhoršovat schopnost sítě generalizace, tzn. dojde ke zvýšení chybovosti na tzv. validační množině dat.

Možnost validace během trénování neuronové sítě musela být doprogramována, aby mohlo být zabráněno přeučení klasifikátorů. Knihovna FANN umožňuje implementaci callback funkce, která je volána během každé trénovací epochy a její návrátová hodnota je určující pro pokračování trénování nebo jeho přerušení. Jde tedy o ideální způsob, jak rozšířit funcionalitu knihovny o validaci. Zastavovací pravidlo má následující podobu. Pokud dojde několikrát bezprostředně po sobě ke zvýšení chyby na validační množině dat, učení sítě bude předčasně zastaveno. Počet zvýšení validační chyby je pak jedním z parametrů učení sítě.

V callback funkci dojde ke změně trénovacích dat na validační množinu, uloží se současná hodnota chyby (vypočítána při trénování), otestuje se současné naučení sítě na validačních datech a zkontroluje se nová chybovost s jejím historickým vývojem. Pokud došlo k příliš velkému počtu zvýšení, funkce vrátí negativní hodnotu, čímž dojde k zastavení trénování. V opačném případě se chyba uloží, obnoví se dříve uložená trénovací chyba a změní se množina dat na trénovací. Funkce v tomto případě vrátí pozitivní hodnotu.

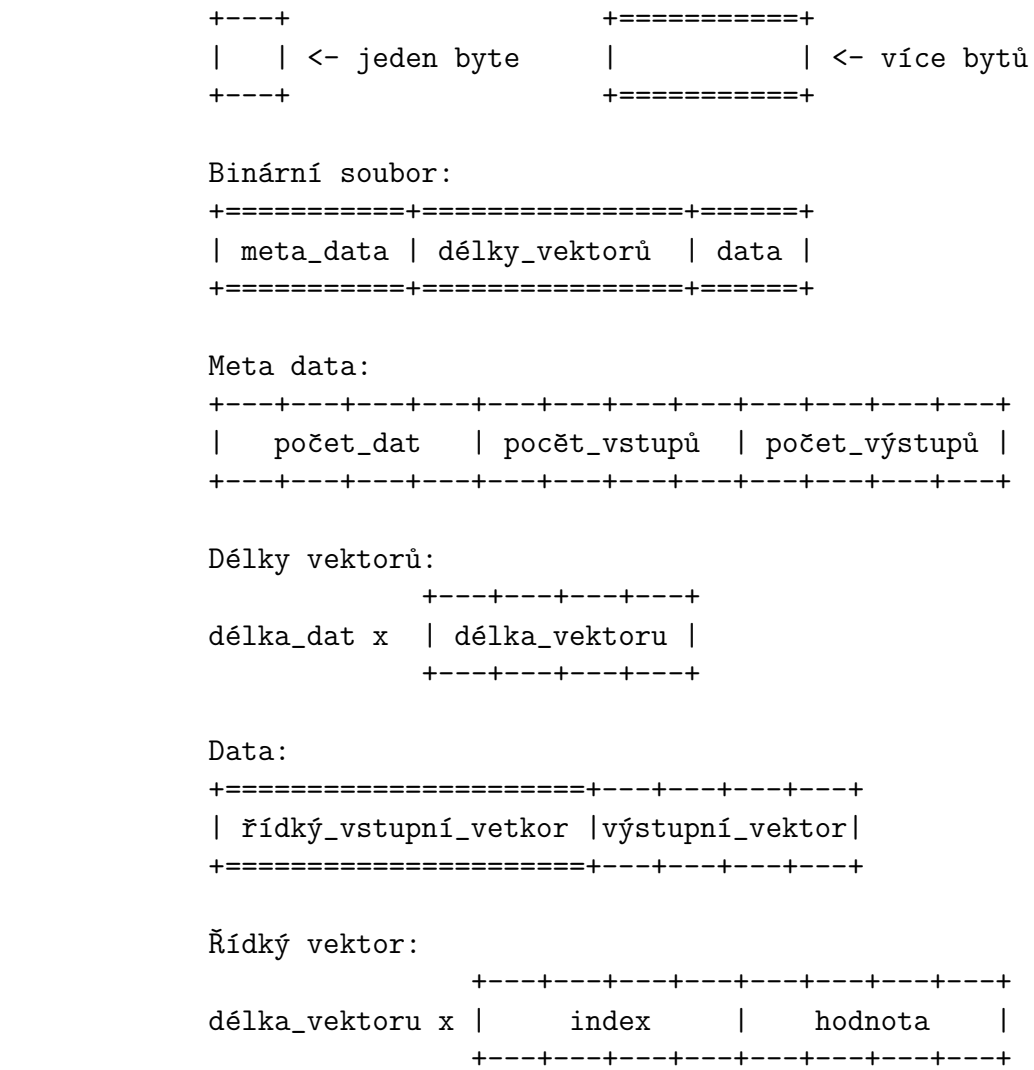
Řídký formát dat

Přepracován byl rovněž způsob reprezentace vstupních a výstupních vektorů z hustého tvaru na řídký. Tím dojde ke markantnímu snížení prostorové náročnosti celého algoritmu. Z implementačního hlediska šlo o poměrně jednoduchou úpravu, kdy jsou z datovéhou souboru přečteny a ve formě atributů objektu uchovány údaje o požadované velikosti vstupních a výstupních vektorů, které jsou následně použity pro sestavení hustých vektorů z řídké reprezentace načtené z datového souboru. Data jsou tedy uchovávána a načítána efektivně a k jejich rozšíření do hustého tvaru dochází pouze ve chvíli výpočtu při učení neuronové sítě.

4.3.4 Formát pro binární uchování dat

Data, se kterými neuronové sítě pracují, jsou vyjádřena číselnými hodnotami. Uložení číselných hodnot v textovém souboru je však neefektivní a obecně je práce

s textovým souborem náročnější, pokud jde o přístup k různým částem souboru. Výhodnější variantou je ukládání těchto dat v podobě binárního souboru, který umožňuje dynamicky přistupovat k hodnotám na libovolném místě v souboru, bez komplikovaného čtení a práci s textovými daty. Na obrázku 11 je znázorněn formátu souboru, který byl implementován pro účely uchování vektorové reprezentace textových dokumentů a dynamický přístup k jednotlivým vektorům v průběhu trénování nebo testování sítě. Použití tohoto souboru také usnadní práci s rozdělením datového souboru na trénovací, testovací a validační podmnožiny. Rozhraní mezi



Obrázek 11: Schéma uspořádání dat v binárním souboru

binárním souborem a třídou pro reprezentaci dat používaných neuronovou sítí zajišťuje třída *tie*.

4.3.5 Nejbližší soused

Poslední implementovanou částí je algoritmus nejbližšího souseda, který klasifikuje na základě podobnosti či vzdálenosti jednotlivých dokumentů. K vlastní implementaci bylo přistoupeno z důvodu nemožnosti využití některé z dostupných knihoven, které načítaly celá vstupní data do paměti. V případě dat zpracovávaných v této práci jde o bezmála 10 GB, což není pro běžné počítače prostorově únosné. Ve vlastní implementaci budou tedy využity již dostupné prostředky v podobě binárního souboru a řídké reprezentace dat. Pro další snížení náročnosti je množství prohledávaných dokumentů zredukováno pomocí tzv. k-dimenzionálního stromu, který rozděluje prostor vstupních dat dle jednotlivých dimenzí na menší podprostory. Tímto způsobem jsou všechny trénovací dokumenty rozděleny do podmnožin a při klasifikaci neznámého dokumentu se nejdříve zjistí, do které podmnožiny by dle svých hodnot atributů spadal, a následně je porovnán pouze s dokumenty z této podmnožiny.

4.4 Experimenty

Cílem experimentů je prozkoumat možnost paralelního zpracování rozsáhlé kolekce dokumentů a zjistit, jaký může mít podobný přístup vliv na přesnost výsledné klasifikace těchto textových dat. Experimenty jsou realizovány následujícím způsobem: jednak je nutné vyzkoušet různá rozdělení trénovací množiny a také různé velikosti této množiny. Na základě již realizovaných experimentů (Žižka, Dařena, 2012) byl zvolen podobný přístup, aby získané výsledky bylo možné porovnat. Hlavní experiment bude realizován nad vybranou trénovací množinou o 200 000 dokumentech a tato kolekce bude rozdělena na 10, 5, 4, 3 a 2 disjunktní podmnožiny. Stejným způsobem pak bude zpracována i množina o 100 000 dokumentech, na které bude také otestován vliv změny parametrů klasifikátorů na výslednou klasifikaci.

Pro realizaci experimentů byl použit domácí počítač s poměrně starým hardwarovým vybavením. Použitý stroj byl osazen tříjádrovým procesorem od firmy AMD a k dispozici bylo celkem 4 GB operační paměti. Tento počítač posloužil jako *host* pro vytvoření dvou virtuálních strojů, na kterých byly exeprimenty spouštěny. Oběma virtuální strojům bylo přiděleno jedno jádro fyzického procesoru a 1 GB z dostupné operační paměti. Jako operační systém byla použita linuxová distribuce Ubuntu 12.10 v tzv. *minimální* verzi.

Pro všechny experimenty jsou použity totožné dokumenty, které jsou pouze reprezentovány různým způsobem, a to podle použitého typu klasifikátoru. Z vybraných dokumentů byla vyřazena slova, která se na globální úrovni – tzn. v rámci celé vybrané kolekce – vyskytují méně než třikrát. Jiný způsob vyřazování slov z vektorové reprezentace dokumentů nebyl použit. Data jsou také vybírána tak, aby poměr mezi oběma třídami byl vyrovnaný. Díky vyváženosti obou tříd v použitých datech je možné provést shodnocení kvality získané znalosti pomocí metriky *Acc,* stejně tak jsou ale uvedeny i ostatní výše popsané indikátory přesnosti klasi-

fikátoru. Jelikož byly vytvořeny komise klasifikátorů se sudým počtem, je dalším ze sledovaných parametrů počet testovacích případů, o kterých komise nedokázala (v důsledku shodného počtu hlasů) rozhodnout. Důležitým faktorem je také čas, který je potřebný pro natrénování.

4.4.1 Neuronové sítě

Narozdíl od ostatních použitých klasifikátorů vyžadují neuronové sítě pro svůj běh a získání kvalitních výsledků vyžadují použití validační množiny. Ta je vybrána stejným způsobem jako testovací množina, tzn. vzhledem k velikosti trénovací množiny, a to v poměru 3 : 1. Model neuronové sítě bylo možné parametrizovat řadou různých nastavení. Při zpracování dokumentů bylo použito následující nastavení:

- reprezentace dat: term-frequency,
- počet skrytých vrstev: 1,
- velikost skrytých vrstev: 50 neuronů,
- učící algoritmus: backpropagation,
- přechodové funkce: sigmoidální,
- velikost chyby: 0.005,
- validační zastavení: více jak 5 zvýšení validační chyby.

S tímto nastavením jsou nad jednotlivými trénovacími množinami vytvořeny potřebné klasifikátory a následně všechny otestovány na testovací množině. Dosažené výsledky jsou zobrazeny v tabulce 5. Časové hodnoty jsou zde uvedeny jako počet minut, přičemž jde o součet času potřebného pro naučení všech klasifikátorů v dané komisi. Ve sloupcích jsou uvedeny hodnoty: počet klasifikátorů, čas učení, metriky accuracy, recall, precision, f-measure, počet dokumentů bez klasifikace a skóre komise (výpočet uveden dále). Na výsledcích je poměrně hezky vidět vliv nerozhodnosti klasifikátorů, které tvořily komisi se sudým počtem členů, čímž mohlo docházet ke shodnému počtu hlasů pro obě třídy, a proto nemohlo být rozhodnuto o konečné klasifikaci testovaného dokumentu. Nejde však o velké množství případů, u nejhorší varianty, kterou je dvoučlenná komise, bylo takto nezařazeno zhruba 1 % testovacích případů. S vyšším počtem členů komise se množství neklasifikovaných dokumentů zmenšuje, a to přibližně ve stejném poměru pro obě velikosti trénovacích dat. Z hlediska přesnosti klasifikace jsou dosažené výsledky přibližně stejné. Výjimku opět tvoří dvoučlenná komise, u které došlo k poklesu přesnosti o zhruba pět procent, což je převážně důsledek většího množství neklasifikovaných dokumentů. V ostatních metrikách jsou již výsledky mezi jednotlivými komisemi velice vyrovnané, rozdílnost se pohybuje maximálně do dvou procent. Přesnosti je tedy za pomocí komise možné dosáhnout téměř totožné, jako v případě jednoho klasifikátoru, dokonce je zpozo-

Tabulka 5: Výsledky dosažené s klasifikátorem neuronové sítě

100k							
N	Čas [min]	Acc	Rec	Prec	F	Score_k	Bez klasifikace
1	875	0.9195	0.9218	0.9173	0.9195	—	—
2	360	0.8671	0.9625	0.8973	0.9288	0.7658	2941
3	222	0.9256	0.9253	0.9259	0.9256	0.8765	—
4	168	0.9055	0.9520	0.9160	0.9337	0.8964	1454
5	130	0.9334	0.9503	0.9147	0.9322	0.9333	—
10	80	0.9228	0.9451	0.9258	0.9354	0.9561	658
200k							
1	10560	0.9132	0.9007	0.9289	0.9146	—	—
2	1740	0.8564	0.9613	0.8990	0.9291	0.8865	6456
3	1080	0.9334	0.9399	0.9261	0.9330	0.9599	—
4	720	0.9061	0.9520	0.9252	0.9384	0.9620	3169
5	600	0.9373	0.9511	0.9220	0.9363	0.9848	—
10	240	0.9320	0.9554	0.9228	0.9388	0.9989	932

rováno i nepatrné zvýšení. Největší rozdíly jsou však vidět v čase potřebném pro sestavení komise. V případě menší varianty trénovací množiny trvalo naučení jednoho klasifikátoru zhruba 14 hodin. Tento čas je možné zkrátit více jak o polovinu již pouhým rozdělením na dvě podmnožiny. Úspora času dále roste s velikostí komise. U varianty s větší trénovací množinou je časová úspora ještě výraznější. Samostatný klasifikátor byl trénován zhruba sedm a půl dne, zatímco dvoučlenná komise umožnila snížit potřebný čas na 29 hodin. S vyšším počtem klasifikátorů a větším množstvím zpracovávaných tedy výrazně roste časová úspora.

Pokud by bylo třeba vyjádřit jednou číselnou hodnotu kvalitu konkrétní komise, zohledňující čas potřebný na naučení, je nutné navrhnout výpočet této hodnoty. Číselné vyjádření by tedy mělo brát v potaz ušetřené množství času a změnu přesnosti komise oproti jednomu klasifikátoru. Vzorec, navrhnovaný pro výpočet této hodnoty je následující:

$$score_k = 0.5 \cdot \left(\left(1 - \frac{time_1}{time_k} \right) + \frac{Acc_1}{Acc_k} \right)$$

Jde o pouhou průměrnou hodnotu z poměrů mezi časem a přesností komise a jednoho klasifikátoru, přičemž čas je přepočítán na doplněk do jedné, aby větší úspora času

měla pozitivní vliv na výslednou hodnotu skóre. Použitím tohoto způsobu vyjádření kvality je možné jednotlivé varianty mezi sebou snadno porovnat. Jako výhodnější se tedy jeví komise s vyšším počtem členů, zjeména kvůli výrazným časovým úsporám. Vzorec by dále bylo možné upravit na vážený průměr, pomocí kterého by mohl být kladen větší důraz na změnu přesnosti.

4.4.2 Podpůrné vektory

Algoritmus podpůrných vektorů je sám o sobě velice vhodným pro zpracování textových dokumentů. Jednak je jeho proces učení mnohem rychlejší než u neuronových sítí, ale také může dosáhnout lepší přesnosti. Aplikování paralelního přístupu by tedy teoreticky mohlo také ještě zvýšit použitelnost tohoto typu klasifikátoru při zpracování rozsáhlých kolekcí.

Joachims publikoval řadu článků, ve kterých se věnoval použití podpůrných vektorů při klasifikaci textových dokumentů. Parametrizace klasifikátoru tedy vychází z doporučení uvedených v těchto článcích (Joachims, 1998).

- reprezentace dat: tf-idf,
- normalizace dat: kosinová,
- jádrová funkce: polynomiální,
- stupeň polynomu: 3.

Průběh je totožný jako u neuronových sítí – každý klasifikátor je s tímto nastavením natrénován nad konkrétní trénovací množinou a následně otestován. Výsledky shrnuje tabulka 6, přičemž čas je tentokrát vyjádřen jako počet vteřin. Dosažené výsledky jsou velice podobné, jako v případě neuronových sítí. Došlo však ke snížení počtu neklasifikovaných dokumentů a obecně je dosahováno lepších hodnocení. Přesnost mezi jednotlivými komisemi je vyrovnanější, ale vůči jednomu klasifikátoru nedosáhla (narozdíl od sítí) žádná komise lepší přesnosti. Časová úspora, ačkoliv je stále vzhledem k jednomu klasifikátoru poměrně velká (z necelých 4 hodin na půl hodiny), není tak výrazná jako u neuronových sítí. Zde se tedy nabízí otázka, zda tato časová úspora stojí jednoduše řečeno za námahu s přípravou dat pro dílčí klasifikátory. Rozdíl v přesnosti mezi komisí 10 klasifikátorů a samostatného klasifikátoru činí 1 %, takže získaná znalost je teoreticky téměř totožná a mnohem jednodušší je vytvořit a používat klasifikátor jeden. Uplatněním *Occamovi břitvy* by tedy mohl být jako nejlepší řešení zvolen i navzdory časové úspoře samostatný klasifikátor. Na základě skóre pro ohodnocení komise jsou stejně jako v předchozím případě výhodnější skupiny s vyšším počtem klasifikátorů.

4.4.3 Nejbližší soused

Posledním typem klasifikace dokumentů je metoda nejbližšího souseda. Jelikož nebyl tento přístup implementován pomocí knihovny, vychází veškeré možnosti paramet-

Tabulka 6: Výsledky dosažené s klasifikátorem podpůrných vektorů

100k							
N	Čas [sec]	Acc	Rec	Prec	F	Score_k	Bez klasifikace
1	3945	0.9471	0.9657	0.9268	0.9459	—	—
2	2400	0.9299	0.9728	0.9065	0.9385	0.6867	819
3	1800	0.9422	0.9642	0.9185	0.9408	0.7693	—
4	1200	0.9337	0.9691	0.9074	0.9372	0.8408	492
5	1200	0.9401	0.9647	0.9137	0.9385	0.8442	—
10	500	0.9346	0.9679	0.9047	0.9352	0.9300	234
200k							
1	14064	0.9506	0.9648	0.9352	0.9498	—	—
2	10800	0.9374	0.9731	0.9190	0.9453	0.6091	1396
3	7200	0.9480	0.9658	0.9290	0.9470	0.7427	—
4	5280	0.9409	0.9705	0.9201	0.9446	0.8072	796
5	3840	0.9458	0.9652	0.9250	0.9447	0.8610	—
10	2100	0.9393	0.9661	0.9150	0.9399	0.9194	402

rizace běhu z poměrně přímočaře pojaté implementace.

- reprezentace dat: term-frequency,
- měření podobnosti: kosinová podobnost,
- počet sousedů: 5,
- počet dělících dimenzí: 10,
- maximální hloubka K-D stromu: 8.

Dosažené výsledky získané porovnáním dokumentů z testovací množiny oproti jednotlivým trénovacím množinám jsou zobrazeny v tabulce 7, kde čas je stejně jako u neuronových sítí vyjádřen počtem minut. Oproti předhozím klasifikátorům výrazně vzrostl počet neklasifikovaných dokumentů, což má také nezanedbatelný dopad na výslednou přesnost komisí se sudým počtem členů. U dvoučlenné komise nebylo možné z celkového počtu testovaných dokumentů pro více jak čtvrtinu určit výslednou třídu, což má za následek více jak 10 % propad v přesnosti, ale zároveň došlo k stejnému nárůstu hodnoty metriky recall. Z hlediska časové náročnosti došlo k malé úspoře v případě menší trénovací množiny u dvoučlenné komise, v ostatních

Tabulka 7: Výsledky dosažené s klasifikátorem nejbližšího souseda

N	Čas [min]	Acc	Rec	Prec	F	Score_k	Bez klasifikace
			100k				
1	57	0.7728	0.7444	0.8331	0.7863	—	—
2	48	0.6344	0.8508	0.7064	0.7719	0.4894	8951
3	53	0.7999	0.7688	0.8579	0.8109	0.5526	—
4	52	0.7047	0.7904	0.8888	0.8367	0.4998	5247
5	61	0.8212	0.7729	0.9096	0.8357	0.4962	—
10	60	0.7948	0.8161	0.8979	0.8550	0.4879	2403
			200k				
1	370	0.7945	0.7524	0.8784	0.8105	—	—
2	250	0.6337	0.8507	0.7286	0.7850	0.5610	18132
3	232	0.8239	0.7847	0.8926	0.8352	0.7050	—
4	235	0.7448	0.8405	0.8606	0.8505	0.6512	9746
5	211	0.8334	0.7872	0.9137	0.8458	0.7393	—
10	270	0.8063	0.8209	0.9079	0.8622	0.6426	4326

případech byly časy velice podobné. U větší trénovací množiny již došlo k určitým časovým úsporám, avšak oproti předchozím variantám nikterak velkým. Ohodnocení jednotlivých komisí pomocí dříve navrženého vzorce jasně ukazuje vliv neklasifikovaných dat, kdy se rozdíl ve výsledné hodnotě mezi komisí se sudým a lichým počtem hlasů pohybuje mezi pěti a sedmnácti procenty, přičemž lepších výsledků dosahovali početnější komise. Použitím komise s lichým počtem klasifikátorů je tedy možné dosáhnout určité úspory času a také zlepšení přesnosti klasifikace. Důvodem je porovnávání neznámého dokumentu s větším počtem sousedů, než je tomu u samostatného klasifikátoru.

4.4.4 Shrnutí výsledků

Na základě získaných výsledků je možné odvodit určité závěry o možnosti paralelizace zpracování rozsáhlých kolekcí textových dokumentů. Rozdělením zpracovávané kolekce dokumentů na několik menších disjunktních množin je možné dosáhnout výrazného snížení časové náročnosti, a to s velice malým snížením schopnosti klasifikátoru správně klasifikovat neznámé dokumenty. Výjimku tvoří metoda nejbližšího souseda, u které bylo vypozorováno i mírné zvýšení přesnosti, avšak stanovení

konkrétního trendu časové úspory není jednoznačné. Z experimentu s rozsáhlejší trénovací množinou je vidět rostoucí úspora se zvyšujícím se počtem členů komise, avšak tento závěr narušuje komise s deseti členy, kdy došlo naopak ke zvýšení času. U metody nejbližšího souseda byl také nejvíce patrný negativní dopad sudého počtu klasifikátorů, který může mít za následek zhoršení přesnosti kvůli neklasifikovaným dokumentům. Nejvýraznější časové úspory je možné dosáhnout s neuronovými sítěmi, u kterých má díky velké časové náročnosti roustoucí s počtem zpracovávaných dat paralelní zpracování největší význam. Pro zjištění vlivu různých parametrů na výsledky získané při paralelním zpracování textových dat byly také realizovány experimenty s rozdílným nastavením. Z časového hlediska byla použita pouze trénovací množina o 100 000 dokumentech. U neuronových sítí byla zjednodušena architektura a snížena požadovaná chyba, u podpůrných vektorů změněna jádrová funkce na radiální a u nejbližšího souseda byl zvýšen počet sousedů. Dosažené výsledky jsou velice podobné a obecný trend ve snížení časové náročnosti se zanedbatelným snížením přesnosti platí pro neuronové sítě i podpůrné vektory. U nejbližšího souseda bylo dosaženo také podobných výsledků v podobě zvýšení přesnosti při použití lichého počtu klasifikátorů, ale nevýrazná časová úspora byla spíše náhodná.

5 Diskuze

V této diplomové práci bylo snahou prozkoumat možnosti paralelizovaného přístupu k dolování znalostí z rozsáhlých kolekcí textových dokumentů, které jsou pro své typické vlastnosti velice časově i prostorově náročné na zpracování. Získané výsledky ukázaly, že rozdělením problému na několik menších dílčích problémů je možné dosáhnout snížení časové náročnosti a neutrpět výrazné snížení přesnosti klasifikace. Výjimkou je algoritmus nejbližšího souseda, u kterého ke snížení času docházelo spíše náhodně, ale pokud byl počet klasifikátorů lichý, bylo možné zvýšit celkovou přesnost. Rozhodujícím faktorem je velikost komise klasifikátorů, která s vyšším počtem členů umožňuje vyšší snížení časové náročnosti, přičemž rozdíly v přesnosti byly většinou velice malé. Výhodnější je tedy použití vyššího počtu klasifikátorů, pokud možno lichého počtu, aby nedocházelo k nerozhodnosti hlasů komise při klasifikaci neznámého dokumentu.

5.1 Nedostatky implementace

Největším nedostatkem je současná podoba objektového návrhu aplikace, který používá menší množství tříd obsahující funkcionality, které by bylo možné vyčlenit a získat tím čistší návrh. Pokud by tedy aplikace měla sloužit jako podklad pro další výzkum nebo i praktické využití, bylo by velice vhodné provést refaktoring. Určitým nedostatkem může být také vlastní implementace algoritmu nejbližšího souseda a K-D stromů, které nebyly ověřeny větším množstvím uživatelů, jako tomu bývá u volně dostupných knihoven.

5.2 Možnosti navázání na práci

Práce se dané problematice věnuje ještě na poměrně obecné úrovni a zkoumán byl především vliv paralelizace na různé algoritmy. Možností, jak dále přistoupit k tomuto problému, je několik. Jednak by bylo vhodné použít jiný typ textových dat, který by měl rozdílné charakteristiky, a porovnat získané výsledky. Vliv parametrizace jednotlivých algoritmů byl v této práci také zahrnut, avšak z časových důvodů bylo provedeno pouze malé množství experimentů s různým nastavením a vyvozené závěry tedy nemusí mít dostatečnou podporu v empirických datech.

5.3 Možnosti využití v praxi

Díky rozsáhlým úsporám v čase potřebném pro naučení klasifikátoru by mělo být možné podobný přístup využít pro zpracování velice rozsáhlých kolekcí textových dokumentů, a to i na běžném domácím počítači. V akademické a firemní sféře je takto možné ušetřit výrazným způsobem výpočetní prostředky při náročných výpočtech spojených se zpracováním velkého množství textových dat.

6 Závěr

Cílem této diplomové práce bylo prozkoumat možnost aplikování paralelního přístupu při dolování znalostí z textových dat. Práce se zaměřila především na rozdílnost ve výsledcích pro tři různé klasifikační algoritmy, a to neuronovými sítěmi, podpůrnými vektory a metodou nejbližšího souseda. Při zpracování práce byla nastudována problematika předzpracování textových dokumentů a jejich transformace do vektorové reprezentace, způsob fungování klasifikátorů používaných při práci s textovými daty a již dosažné výsledky v oblasti paralelizace tohoto typu problému.

V práci samotné je popsána značná část teorie spojené se zpracováním textových dokumentů a jejich přípravou pro algoritmy strojového učení, stejně tak i princip těchto algoritmů a krátké shrnutí dosavadních poznatků paralelizování zpracování textových dat.

Jako jeden z výstupů, nikoliv však hlavní, byla vytvořena aplikace pro realizaci experimentů. Jedním z hlavních problémů při implementaci byla prostorová náročnost zpracování textových dat. Řešením byla úprava vybraných knihoven tak, aby byly schopné pracovat s řídkými vektory, díky kterým bylo dosaženo enormního snížení požadvaku na paměťové vybavení počítače, a experimenty tak mohly být realizovány i na běžném domácím počítači. Provedené úpravy by tedy bylo možné dále využít při jakékoliv úloze týkající se klasifikace velkého množství textových dat.

Hlavním výstupem jsou výsledky samotných experimentů, které měly za úkol odhalit vliv paralelizace na úšpěšnost a náročnost tvorby jednotlivých klasifikátorů. Snahou bylo rozšířit poznatky a nastínit tak další možnosti směrování výzkumného snažení v této oblasti.

A Příloha 1

Zdrojové kódy aplikace spolu s výsledky experimentů jsou k dispozici na přiloženém DVD.

B Literatura

AI Basics *John McCarthy* Artificial intelligence: Manufactured Minds [online]. [cit. 2013-04-13]. Dostupné z WWW: http://library.thinkquest.org/05aug/01158/mccarthy.html.

Banko, M., Brill, E. *Scaling to Very Very Large Corpora for Natural Language Disambiguation* Proceedings of the 39th Annual Meeting on Association for Computational Linguistics – ACL '01 [online]. 2001, s. 26–33 [cit. 2013-05-03]. Dostupné z: doi:10.3115/1073012.1073017 .

Cardoso-Cachopo, A. *Datasets for single-label text categorization* [online]. 2007 [cit. 2013-05-11]. Dostupné z WWW: http://web.ist.utl.pt/acardoso/datasets/.

Dařena, F. *Myslíme v jazyku Perl* Vyd. 1. Praha: Grada, 2005, 700 s. ISBN 80-247-1147-8.

Dařena, F. *Vybrané přístupy k dolování znalostí ze sociálních médií* 2011. Habilitační práce. Mendelova univerzita v Brně.

Fung, P., Kan, M., Horita, Y. *Extracting Japanese Domain and Technical Terms is Relatively Easy* Computer Science Deparment, Columbia University, New York, 1996. Dostupné z: doi:10.1.1.151.278.

Joachims, T. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features* Proceeding of the European Conference on Machine Learning, Springer, 1998.

Kernighan, M., Church, K., Gale, W. *A spelling correction program based on a noisy channel model* In Proc. ACL, 1990, s. 205–210.

Konchady, M. *Text Mining Application Programming* Boston: Charles River Media, 2006, xix, 412 s. ISBN 978-1-58450-460-3.

Konečný, V., Trenz O. *Umělá inteligence* Brno: 2010. 120 s.

Koza, J. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems* Computer Science Department, Standford University [online]. 1990 [cit. 2013-04-13]. Dostupné z WWW: http://www.genetic-programming.com/jkpdf/tr1314.pdf.

Lertnattee, V., Theeramunkong, T. *Parallel text categorization for multi-dimensional data* Lecture Notes in Computer Science, Vol. 3320, 2004, s. 38–41.

Lewis, D. *Reuters-21578 Text Categorization Test Collection* [online]. 2007 [cit. 2013-05-11]. Dostupné z WWW: http://daviddlewis.com/resources/testcollections/reuters21578.

Manning, C., Raghavan, P., Schütze, H. *Introduction to Information Retrieval* Cambridge University Press, 2008.

Neto, J., Santos, A., Kaestner, C., Freitas, A. *Document Clustering and Text Summarization* Pontificia Universidade Catolica do Parana, 2000. Dostupné z: doi:10.1.1.43.4634.

Ng, A. *Machine Learning* Coursera [online]. Stanford University, [cit. 2013-05-03]. Dostupné z WWW: https://www.coursera.org/course/ml.

Novák, Z., Dařena, F. *Aplikace pro přípravu textových dat* [CD-ROM]. In PEFnet 2012. ISBN 978-80-7375-669-7.

Porter, M. *An algorithm for suffix stripping* Program, 14(3), s. 130–137, 1980.

Rennie, J. *20 Newsgroups* [online]. 2008 [cit. 2013-05-11]. Dostupné z WWW: http://qwone.com/~jason/20Newsgroups/.

Rosenblatt, F. *The Perceptron–a perceiving and recognizing automation* Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

Russel, S., Norvig, P. *Artificial intelligence: A modern approach* Englewood Cliffs, N.J.: Prentice Hall, 1995, xxviii, 932 s. ISBN 01-310-3805-2.

Salton, G., Wong, A., Yang, C. *A Vector Space Model for Automatic Indexing* Communications of the ACM 18, 11 (November 1975), s. 613–620. Dostupné z: doi:10.1145/361219.361220.

Sebastiani, F. *Text categorization* Text Mining and its Applications to Intelligence, CRM and Knowldge, WIT Press. 2005, s. 109–129.

Singhal, A. *Term weighting revisited* [online]. 1997 [cit. 2013-05-10]. Disertační práce. Cornell University. Dostupné z WWW: http://ecommons.cornell.edu/bitstream/1813/7281/1/97-1626.pdf.

Šíma, J., Neruda, R. *Teoretické otázky neuronových sítí* Praha: Matfyzpress, 1996, 390 s. ISBN 80-858-6318-9.

Text Mining Science *Text Mining for Science* [online]. 2012 [cit. 2013-04-16]. Dostupné z WWW: http://textminingscience.com/pages/text-mining-science.

Turing, A. *Computing machinery and intelligence* [online]. [cit. 2013-04-13]. Dostupné z WWW: http://www.loebner.net/Prizef/TuringArticle.html.

Turney, P., Pantel, P. *From Frequency to Meaning: Vector Space Models of Semantics* Journal of Artificial Intelligence Research 37, 2010, s. 141–188.

Ulmer, C., Gokhale, M., Gallagher, B., Top, P., Eliassi-Rad, T. *Massively parallel acceleration of a document-similarity classifier to detect web attacks* Journal of Parallel and Distrubuted Computing, Vol. 71, No. 2, 2011, s. 225–235.

Vapnik, V. *The Nature of Statistical Learning Theory* Springer, 1995.

Wang, Y., Zhang, J., Vassileva, J. *Towards Effective Recommendation of Social Data across Social Networking Sites* In: Dicheva, D., Dochev, D. (eds.) Artificial Intelligence: Methodology, Systems, and Applications. Springer, 2010, s. 61–70.

Weiss, S., Indurkhya, N., Zhang, T., Damerau, F. *Text Mining: Predictive Methods for Analyzing Unstructured Information* New York, Springer, 2010. ISBN 978-1-44192996-9.

Weizenbaum, J. *ELIZA—a computer program for the study of natural language communication between man and machine* Communications of the ACM, v.9 n.1, p.36–45. Dostupné z: doi:10.1145/365153.365168.

Whitby, B. *Reflections on artificial intelligence* Exeter: Intellect, 1996. ISBN 18-715-1668-4.

Witten, I., Eibe, F. *Data Mining: Practical Machine Learning Tools and Techniques* 2nd ed. Amsterdam: Morgan Kaufman, 2005. ISBN 978-008-0477-022.

Zhang, Y., Jin, R., Zhou, Z. *Understanding bag-of-words model: a statictical framework* 2010. s. 43–52. Dostupné z: doi:10.1007/s13042-010-0001-0.

Žižka, J., Dařena F. *Automatic Sentiment Analysis Using the Textual Pattern Content Similarity in Natural Language* Lecture Notes in Artificial Intelligence, 2010, 6231, 1: 224–231. ISSN 0302-9743.

Žižka J., Dařena F. *Parallel Processing of Very Many Textual Customers' Reviews Freely Written Down in Natural Languages* In IMMM 2012: The Second International Conference on Advances in Information Mining and Management. 2012, s. 147–153. ISBN 978-1-61208-227-1.