Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

PLZEŇ, 2018 **ROBERT BRADA**

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne

……..……
podpis

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce doc. Ing. Pavlu Ircingovi, Ph.D. za cenné a profesionální rady, připomínky, materiály a metodické vedení práce.

Abstrakt

Cílem předkládané bakalářské práce je prozkoumat různé techniky identifikace rodného jazyka pisatelů na základě anglicky psaných esejů. Práce je rozdělena na teoretickou a praktickou část. První část představuje stručný teoretický úvod do oblasti strojového učení a zpracování přirozeného jazyka. V praktické části je popsáno použité programovací prostředí společně s praktickými ukázkami programovacího kódu. Závěr je věnován prezentaci a zhodnocení dosažených výsledků.

Klíčová slova

zpracování přirozeného jazyka, identifikace rodného jazyka, strojové učení, klasifikace

Abstract

The purpose of this study is to try and discuss different techniques that are used in native language identification based solely on essay written in English. This thesis consists of theoretical and practical part. The practical part represents introduction into machine learning and methods used in natural language processing. The practical part describes programming environment and practical examples of code are shown here. Finally, achieved results are presented and discussed in the last chapter.

Keywords

natural language processing, native language identification, machine learning, classification

Obsah

Seznam použitých zkratek

NLI ..... Úloha identifikace rodného jazyka (Native Language Identification)

NLP ..... Zpracování přirozeného jazyka (Natural Language Processing)

L1 ..... Rodný jazyk autora

L2 ..... Druhý jazyk autora (v této práci angličtina)

SVM ..... Support Vector Machines (str. 12)

BOW..... Bag Of Words (str. 15)

MTD ..... Matice termů a dokumentů (str. 15)

CBOW ..... Continuous Bag Of Words (str. 16)

PV-DM ..... Distributed Memory version of Paragraph Vector (str. 18)

PV-DBOW ..... Distributed Bag Of Words version of Paragraph Vector (str. 18)

POS..... Slovní druh (Part Of Speech)

POS tag..... Značka reprezentující slovní druh slova

rbf ..... Radiální bázová funkce (Radial basis function)

1 Úvod

Identifikace rodného jazyka spadá do oblasti zpracování přirozeného jazyka. Jedná se o úlohu, ve které se se snažíme automaticky určit rodný jazyk autora (L1) čistě na základě textu, který je autorem napsán v jeho druhém jazyce (L2). NLI úloha je pojata jako klasifikační problém. K dispozici máme sadu textů, u nichž předem známe L1 autora. Na těchto datech je program naučen.

Uplatnění výsledků NLI lze vidět například v oblasti výuky cizího jazyka. Je zřejmé, že lidé se stejným rodným jazykem budou náchylní k tomu, aby dělali podobné chyby při studiu cizího jazyka. Naučí-li se systém takové vlastnosti rozpoznávat, může být jazykovým studentům poskytnuta cílená zpětná vazba ohledně jejich chyb, což značně zefektivní výuku. Vytvořený model ideálně umí nejen klasifikovat rodné jazyky, ale dokáže zachytit například i úroveň angličtiny, kterou je text napsán. Těchto vlastností lze také využít například k identifikaci autora či rozdělení pisatelů do skupin dle jejich schopností. Rozšířenou úlohou NLI je určení rodného jazyka na základě výslovnosti. V takovém případě jsou výsledky zpravidla lepší. Tato práce je ovšem věnována identifikaci L1 pouze na základě psaného textu.

Navzdory zájmu byl vývoj v této oblasti brzděn kvůli dvěma problémům. Tím prvním je nedostatek dat. K naučení programu je třeba mít dostatečné množství dat, která by byla pro tuto úlohu vhodná. Je tedy vyžadováno mít dostatečný počet esejí napsaných v jednom jazyce různými rodilými mluvčími. Požadované množství takových dat nebylo zpočátku k dispozici. Dalším problémem byla neshoda ohledně toho, jaké rodné jazyky klasifikovat. Výsledkem byl vznik datové sady TOEFL11, která byla navržena přímo pro úlohu NLI. Tato sada obsahuje anglické texty napsané pisateli s 11 různými rodnými jazyky. Obecně lze NLI použít na jakékoli jazyky, problémem je ovšem již zmiňovaný nedostatek dat.

V současné době jsou vědecké práce na toto téma zveřejňovány především v rámci BEA workshopu (Workshop on Innovative Use of NLP for Building Educational Applications), který se zabývá aplikacemi zpracování přirozeného jazyka v oblasti výuky. Tento workshop měl v roce 2017 dvanáctý ročník. Úloha NLI byla zařazena roku 2013 a pak znovu až v roce 2017 (1).

2 Teoretický úvod

NLI se řeší metodou strojového učení. Přičemž strojové učení je oblast umělé inteligence zabývající se technikami, které umožňují počítačovému systému učit se, aniž by musel být naprogramován pro konkrétní úlohu. Učící se algoritmy lze rozdělit do dvou základních skupin (2):

- **Učení s učitelem**

  Pro vstupní data známe požadované výstupy. Po natrénování na trénovacích datech můžeme algoritmus použít ke zpracování nových dat, která nikdy neviděl.

- **Učení bez učitele**

  Pro vstupní data neznáme požadovaný výstup a chceme po algoritmu, aby je sám roztřídil do určitého počtu tříd.

NLI patří do kategorie učení s učitelem. Máme k dispozici texty, u kterých víme, jaký je rodný jazyk autora. Na těchto textech můžeme systém natrénovat a poté mu předložit texty nové.

Ve strojovém učení se obvykle řeší jedna ze tří základních úloh:

- **Klasifikace**

  Vstupní data lze rozdělit do omezeného množství tříd (např. rozpoznávání psaných číslic 1 až 9, pacient má/nemá nádor).

- **Regrese**

  Úkolem je odhadnou výstup, který spojitým způsobem závisí na vstupních datech. (např. předpovídání cen bytů, očekávaná návštěvnost webu).

- **Shlukování**

  Spadá do kategorie učení bez učitele. Úkolem je roztřídit objekty s podobnými vlastnostmi (např. rozdělit zákazníky do skupin podle jejich chování).

NLI je úloha klasifikace, jelikož na světě je omezené množství jazyků. Úkolem je určit, který z těchto jazyků je rodným jazykem autora. Aby bylo možné k textu přiřadit rodný jazyk, je třeba vytvořit a natrénovat klasifikátor, který to dokáže.

2.1 Klasifikátory

Klasifikace je ve strojovém učení a statistice druh problému, kdy máme určit, do které z kategorií dat dané pozorování patří. Algoritmus, který implementuje klasifikaci, se nazývá klasifikátor.

Základní způsoby klasifikace (2):

- **Klasifikace do jedné ze dvou tříd** (*Binary classification*)

  Vstupní data náleží pouze do jedné ze dvou tříd (pacient má/nemá nádor).

- **Klasifikace do jedné z více tříd** (*Multi-class classification*)

  Data jsou roztříděna do více než dvou tříd. Každému objektu je přiřazena pouze jedna z těchto tříd. Do této kategorie spadá NLI, kde autorův rodný jazyk je jeden z mnoha.

- **Klasifikace do více tříd zároveň** (*Multi-label classification*)

  Objekt může být zařazen do více tříd najednou. Typicky u klasifikace témat článků, kdy jeden článek může pojednávat o více tématech.

Je mnoho klasifikátorů, které lze použít. Mezi ty nejpoužívanější lze zařadit například:

- Support Vector Machines (SVM)
- Logistická regrese
- Klasifikátor podle K nejbližších sousedů
- Bayesův klasifikátor
- Rozhodovací strom
- Neuronové sítě

Je obtížné určit předem, který klasifikátor se hodí nejlépe pro danou úlohu a obecně takový postup ani neexistuje. Klasifikační algoritmy se mohu lišit v dosažených výsledcích a v tom, jak rychle se dokáží natrénovat. Je tedy třeba vybrat takový klasifikátor, který se natrénuje v rozumném čase a zároveň poskytuje dobré výsledky.

Ukázalo se, že pro NLI úlohu fungují velmi dobře SVM klasifikátory (1). Na následující straně bude tedy popsán základní princip jen SVM klasifikátoru.

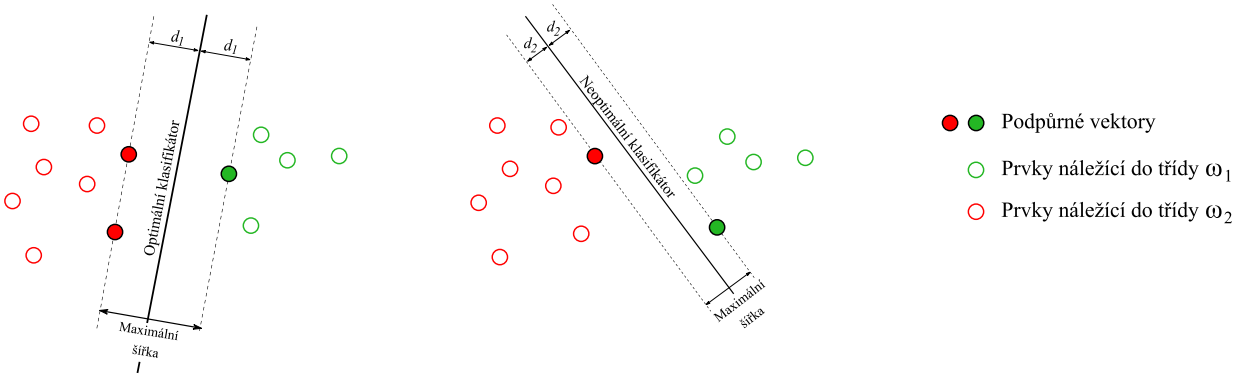
2.1.1 Lineární SVM klasifikátor

Support Vector Machines (Metoda podpůrných vektorů) je druh algoritmu učení s učitelem, který lze použít jak pro klasifikační, tak pro regresní úlohy. Základní myšlenka SVM je roztřídit data pomocí optimální nadroviny (3). Je třeba vysvětlit, v jakém smyslu je nadrovina optimální. Pro názornost uvažujme následující úlohu:

Máme k dispozici data, která jsou roztříděna do dvou tříd ω_1, ω_2 (binární klasifikace). Chceme najít přímku, která tato data rozděluje. Jak lze vidět na *Obr. 1*, takových přímek může být nekonečně mnoho (*g1, g2, g3, …*).

Obr. 1

Optimální klasifikátor je ten, jehož rozdělující nadrovina (ve 2D přímka) má maximální vzdálenost od podpůrných vektorů - podpůrné vektory (*support vectors*) jsou ty, které mají k rozdělující nadrovině nejbližší vzdálenost. Tato situace je vyobrazena na *Obr. 2*:



Obr. 2 Je zřejmé, že vzdálenost d_1 je větší než d_2. Optimální klasifikátor je takový, jehož vzdálenost odpovídající d_i je největší.

V případě, že data nejsou lineárně separovatelná, se využívá tzv. jádrové transformace. Cílem této metody je převést původně lineárně neseparovatelnou úlohu do prostoru, ve kterém již data můžeme separovat pomocí lineární nadroviny.

2.1.2 Výpočet lineárního SVM klasifikátoru

Předpokládáme, že máme množinu trénovacích vektorů $\{x_1, x_2, \dots, x_n\}$ a k ní příslušnou množinu $\{y_1, y_2, \dots, y_n\}$ která říká, že vektor x_i patří do třídy y_i. Jelikož uvažujeme binární klasifikaci, tak $y_i \in \{+1, -1\}$.

Zaveďme si rovnici klasifikátoru, která rozděluje body do dvou tříd (ve 2D případě přímka, obecně nadrovina) (3):

$$\vec{x} \cdot \vec{w} + b = 0 \qquad 1.0$$

$$\vec{w} - váhový\ vektor, \quad \vec{x} - vstupní\ vektor, \quad b - konstanta$$

Naším cílem bude určit váhový vektor $\vec{w}$. Předpokládejme, že pro všechny body platí:

$$\vec{x} \cdot \vec{w} + b \geq +1 \quad pro\ y_i = +1 \qquad 1.1$$

$$\vec{x} \cdot \vec{w} + b \leq -1 \quad pro\ y_i = -1 \qquad 1.2$$

Tedy pokud bod patří do třídy $y_i = +1$, je splněna nerovnice *1.1*. Naopak pokud bod patří do třídy $y_i = -1$, tak je splněna nerovnice *1.2*. Tyto dvě rovnice lze nepřepsat na jednu nerovnici:

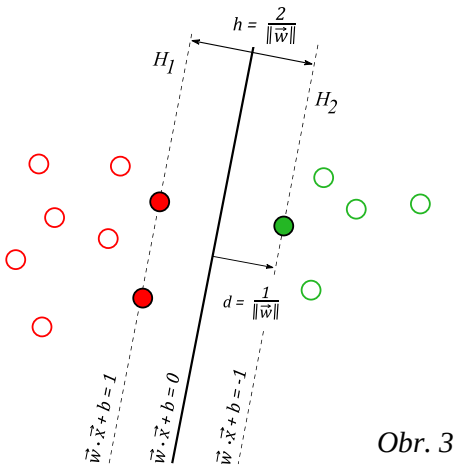
$$y_i(\vec{x} \cdot \vec{w} + b) \geq 1 \quad \forall i \qquad 1.3$$

Body, pro které platí rovnost *1.1*, leží na nadrovině H_1. Body, pro něž platí rovnost *1.2*, leží na nadrovině H_2. Rovnice těchto nadrovin jsou:

$$H_1: \quad \vec{x} \cdot \vec{w} + b = +1 \qquad 1.4$$

$$H_2: \quad \vec{x} \cdot \vec{w} + b = -1 \qquad 1.5$$

Nadroviny H_1 a H_2 jsou rovnoběžné s klasifikační přímkou, od které mají vzdálenost $d = 1/\|\vec{w}\|$. Celková šířka hraničního pásma je tedy $h = 2/\|\vec{w}\|$. Chceme-li tedy nalézt klasifikátor s největší šířkou hraničního pásma h, řešíme úlohu minimalizace $\|\vec{w}\|$ vzhledem k podmínkám *1.3* (každý bod musí ležet na správné straně hraničního pásma). Z praktických důvodů se minimalizuje člen $\|\vec{w}\|^2/2$. Jelikož je problém kvadratický, funkce, jejíž minimum hledáme, bude mít tvar paraboloidu a ten má jedno globální minimum. Celou situaci si lze představit na *Obr. 3*.



Obr. 3

Tato úloha se řeší pomocí metody Lagrangeových multiplikátorů. Jedná se o způsob hledání lokálního minima a maxima funkce vzhledem k omezení rovnosti. Omezení v našem případě představuje rovnice *1.3*.

Lagrangeova funkce pro nalezení minima vypadá následovně:

$$\min L_p = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{l} a_i y_i (\vec{x_i} \cdot \vec{w} + b) + \sum_{i=1}^{l} a_i \qquad 1.6$$

$$\text{Za podmínky } \forall i, a_i \geq 0$$

l - počet trénovacích bodů $\qquad$ a_i - Lagrangeovy multiplikátory

V praxi se kvůli výpočetní výhodnosti řeší tzv. duální problém. Místo minimalizace vzhledem k $\vec{w}$ a b můžeme provést maximalizaci vzhledem k a. Dosazením za $\vec{w}$ a b z rovnic *1.7* do rovnice *1.6* se zbavíme závislosti na těchto proměnných. Tímto postupem dostaneme rovnici *1.8* vyjadřující duální problém:

$$\vec{w} = \sum_{i=1}^{l} a_i y_i \vec{x_i} \qquad \sum_{i=1}^{l} a_i y_i = 0 \qquad 1.7$$

$$\max L_D(a_i) = \sum_{i=1}^{l} a_i - \frac{1}{2}\sum_{i=1}^{l} a_i a_j y_i y_j (x_i x_j) \qquad 1.8$$

$$\text{Za podmínky } \sum_{i=1}^{l} a_i y_i = 0\,, a_i \geq 0$$

Známe-li a_i, lze již snadno dopočítat váhový vektor $\vec{w}$ pro klasifikátor s největším hraničním pásmem:

$$\vec{w} = \sum_{i=1}^{l} a_i y_i \vec{x_i} \qquad 1.9$$

Nový bod $\vec{u}$ můžeme klasifikovat dle toho, jaké znaménko bude mít po dosazení funkce:

$$f(x) = \vec{w} \cdot \vec{u} + b = \left(\sum_{i=1}^{l} a_i y_i \vec{x_i} \cdot \vec{u}\right) + b \qquad 1.10$$

2.2 Převod textových dat do vektorové reprezentace

Vstupem do algoritmů strojového učení musí být vektor fixní délky. Text, který chceme klasifikovat, musíme tedy na takový vektor převést. Tomuto vektoru se říká příznakový vektor. Správná volba příznakového vektoru je klíčová při návrhu modelu a má zásadní vliv na dosažené výsledky. Existují dva základní modely převodu textu na příznakový vektor: Bag Of Words a doc2vec. V následujícím textu bude zmíněn i model word2vec, který neslouží pro vytvoření vektoru z dokumentu, ale ze slova.

2.2.1 Bag Of Words (BOW) model

Každý text je reprezentován svými příznaky. Těmito příznaky mohou být například n-gramy slov, n-gramy znaků, POS tagy atd. V oblasti NLP lze tyto příznaky obecně nazývat termy. Cílem tohoto modelu je vytvořit příznakový vektor, kde jednotlivé prvky tohoto vektoru souvisí s četností termů v dokumentu.

Základem vektorového modelu je matice termů a dokumentů (MTD). Jedná se o reprezentaci, ve které řádky matice odpovídají jednotlivým termům (např. slovům), zatímco sloupce odpovídají dokumentům z dané množiny (jeden dokument odpovídá jednomu sloupci této matice). Prvky $w_{i,j}$ této matice představují váhy, které přiřazujeme jednotlivým termům. Tyto váhy se obvykle odvíjí od počtu výskytů daného termu v dokumentu. V praxi se často používají dva základní způsoby, jak určit váhy matice termů a dokumentů (4):

- **Term frequency (tf)**

  Váhy $w_{i,j}$ přímo odpovídají četnosti termu v dokumentu. Čím častěji se slovo v textu vyskytuje, tím větší váha je mu přiřazena. V praxi se ovšem ukazuje, že často vyskytující se slova nenesou tak důležitou informaci o konkrétním textu. Proto se zavádí tzv. tf-idf model (viz dále).

- **Term frequency - Inverse document frequency (tf-idf)**

  Pro určení vah jednotlivých termů se nejprve určí *inverse document frequency*:

$$idf_i = log\frac{|D|}{df_i}$$

  $|D|$ - *celkovým počet dokumentů* df_i - *počet dokumentů obsahující slovo* v_i

  Konečné váhy matice termů a dokumentů se určí následovně:

$$w_{i,j} = tf_i \cdot idf_i$$

Názorná ukázka, jak převést množinu textů na matici termů a dokumentů:

Příklad:

K dispozici je datová sada obsahující 2 texty:

Text 1: Nevím, zda je to možné.

Text 2: Je možné, že je to pravda.

Prvky $w_{i,j}$ matice termů a dokumentů symbolizují četnost výskytu slova v dokumentu, váhy jsou tedy vytvořeny podle *Term frequency* modelu. Matice bude vypadat jako na *Obr. 4*:

	Text 1	Text 2
je	1	2
možné	1	1
nevím	1	0
pravda	0	1
to	1	1
zda	1	0
že	0	1

Obr. 4

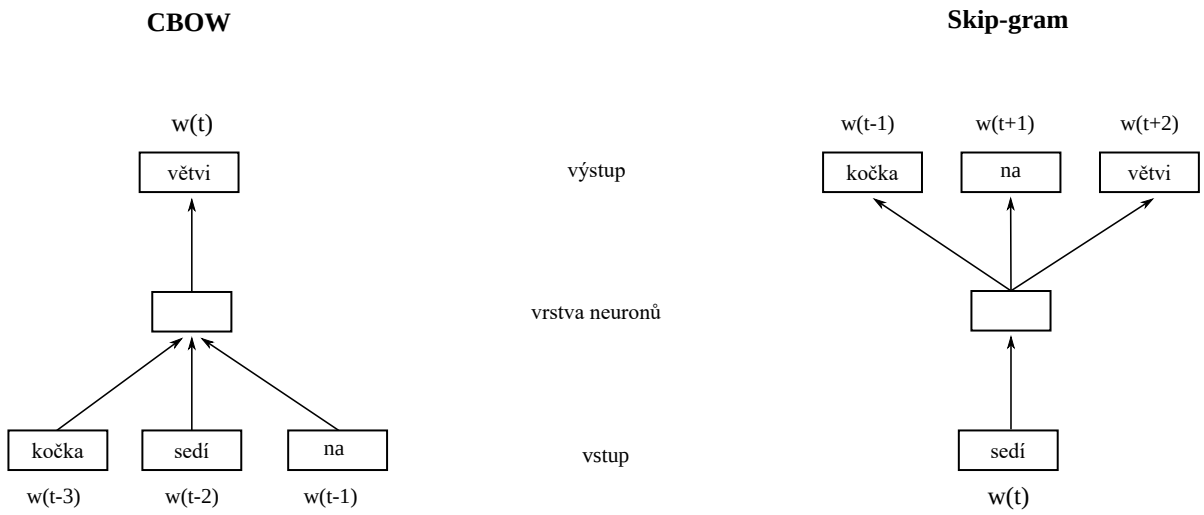
BOW model je snadný na implementaci, ale má dvě zásadní nevýhody. Ve vektoru dokumentu není zachyceno pořadí slov a navíc model neuvažuje sémantickou podobnost slov či dokumentů. Váhy termů jsou jen čísla související s četností výskytu termu v dokumentu. Tyto nedostatky napravují modely word2vec a doc2vec (5) (viz dále).

2.2.2 Word2vec model

Tento model vznikl z potřeby zachytit sémantickou podobnost slov, která jistě hraje důležitou roli v NLP úlohách. V BOW modelu není nijak zachyceno, že např. slova „dobrý“ a „nejlepší“ k sobě mají významově blíže než slova „dobrý“ a „Francie“. Cílem tohoto modelu je vytvořit pro každé slovo vektor fixní délky, který by tyto sémantické vlastnosti dokázal zachytit. V současné době se jedná o *state-of-the-art* techniku v oblasti NLP (6).

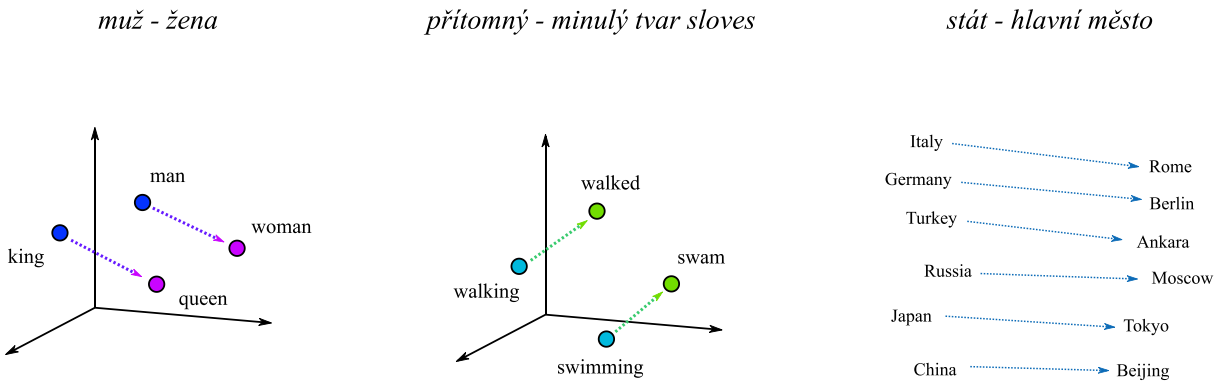
Vektory pro jednotlivá slova lze získat dvěma algoritmy - *Continuous Bag Of Words* (CBOW) a *Skip-gram*. CBOW se snaží předpovědět chybějící slovo v závislosti na okolních slovech (např. „Tráva je“ → „zelená“). Skip-gram model dělá opak - algoritmus dostane na vstup jedno slovo a hledá slova, která jsou v jeho blízkém okolí.

Oba algoritmy fungují jako neuronová síť, která se natrénuje standardním postupem. Architektura těchto sítí je zobrazena na *Obr. 5*. Hledané vektory reprezentující slova lze zjistit z natrénovaných vah neuronové sítě.



Obr. 5 - Architektura CBOW a Skip-gram modelu

Skip-gram je výrazně pomalejší algoritmus, ale ukazuje se, že poskytuje lepší výsledky pro málo frekventovaná slova. Vektory, které získáme těmito algoritmy, v sobě nesou sémantickou informaci o příslušném slovu. Se slovy (vektory) lze také dělat aritmetické operace jako např. *„king“ – „man“ + „woman“ = „queen”*. Jelikož jsou slova reprezentována vektory, můžeme je vykreslit do prostoru a pozorovat různé závislosti, jak lze vidět na *Obr. 6* (7):

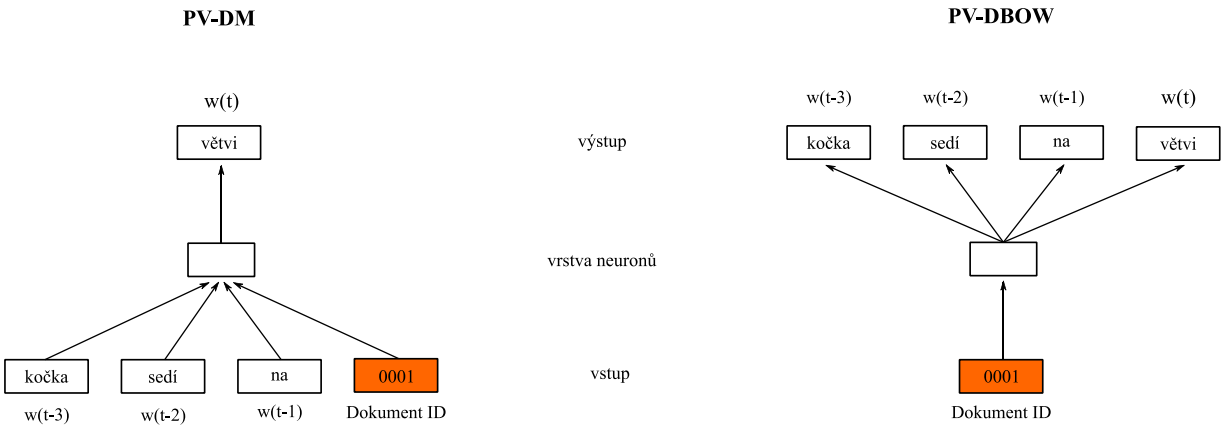


Obr. 6 - Vlastnosti vektorů slov vytvořených pomocí word2vec. (Převzato ze (7))

2.2.3 Doc2vec model

Model doc2vec funguje na podobném principu jako word2vec. Rozdílem je, že v tomto případě je cílem vytvořit vektorovou reprezentaci celého dokumentu, ne jen slova. Velkou výhodou tohoto modelu (i word2vec) je to, že vektory jsou vytvořeny z neoznačených dat (jedná se o učení bez učitele). Jakmile jsou vytvořeny vektory pro dokumenty, můžeme je použít jako vstup do standardního klasifikačního algoritmu.

Vektory lze získat, podobně jako u word2vec, dvěma algoritmy – *Distributed Bag of Words version of Paragraph Vector* (PV-DBOW) a *Distributed Memory version of Paragraph Vector* (PV-DM). Architektury těchto modelů jsou zobrazeny na *Obr. 7* a princip jejich fungování je obdobný jako u word2vec algoritmů (trénování neuronové sítě). PV-DM je identický s algoritmem CBOW pro word2vec model, jen s tím rozdílem, že ke vstupním slovům je přidáno ještě ID dokumentu, které tento dokument jednoznačně identifikuje. Během tohoto algoritmu se tak vytváří vektory jak pro jednotlivá slova, tak pro samotný dokument. Naopak algoritmus PV-DBOW vytváří pouze vektor pro samotný dokument, trénování je tedy o poznání rychlejší. Jelikož je ale ignorován kontext slov, mohou být výsledky horší (5).



Obr. 7 - Architektura PV-DM a PV-DBOW modelu

Pokud bychom si získané vektory vykreslili, bylo by možné například pozorovat, že dokumenty s podobnými tématy, by k sobě měly blíže než jiné (samozřejmě za předpokladu, že v dané množině dokumentů lze takové vlastnosti pozorovat).

2.4 Evaluace modelu

Je více způsobů, jak měřit přesnost klasifikace, mezi ty základní patří (8):

- **Accuracy score**

  Nejjednodušší způsob, jak rychle změřit přesnost klasifikace. Spočítá, kolik procent z testovacích dat jsme správně klasifikovali.

- **Confusion matrix**

  Z této matice lze zjistit nejen počet správně klasifikovaných textů, ale zároveň lze vidět, do jakých tříd byly roztříděny chybně klasifikované texty.

Správný L1 \ Předpovězený L1	ARA	FRE	...
ARA	75	3	…
FRA	1	15	…
…	…	…	...

- **Precision** (přesnost)

  je definována jako procentuální poměr relevantních výsledků analýzy ke všem výsledkům analýzou získaným. Jak lze vidět z následující rovnice, optimální hodnotou je $P = 1$

  Příklad pro konkrétní jazyk L1

$$P = \frac{Počet\ správně\ předpovězených\ L1}{Počet\ všech,\ u\ kterých\ jsme\ předpověděli\ L1}$$

- **Recall** (výtěžnost, úplnost)

  je poměr relevantních výsledků analýzy ke všem relevantním výskytům ve zkoumaném vzorku bez ohledu na to, zda byly analýzou identifikovány. Ideální hodnotou je $R = 1$.

  Příklad pro konkrétní jazyk L1

$$R = \frac{Počet\ správně\ předpovězených\ L1}{Počet\ všech,\ u\ kterých\ je\ správná\ odpověď\ L1}$$

- **F1 skóre**

  udává přesnost modelu na základě hodnot precision a recall.

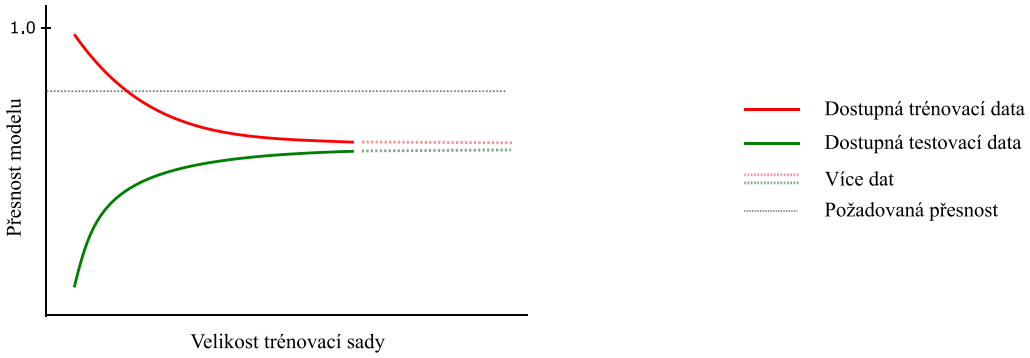
$$F1 = \frac{PR}{P + R}$$

- **Křivky učení (learning curves)**

  Tyto křivky zachycují, jak se mění přesnost modelu v závislosti na počtu trénovacích dat. Model je natrénován na určitém množství dat a poté se změří jeho přesnost na trénovací a testovací sadě. Tyto grafy zpravidla slouží k tomu, aby bylo možné rozlišit jeden z následujících dvou případů (9):

  Overfitting (přetrénování)

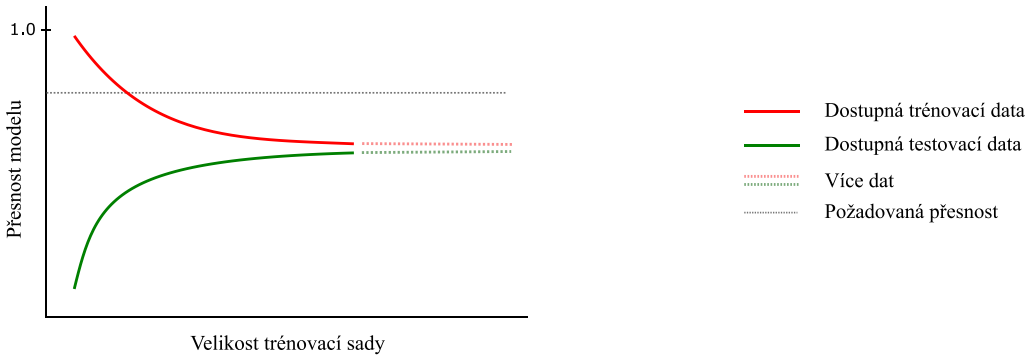
  Nastává tehdy, když je model přeučen na trénovací sadě. V praxi to znamená, že model dokáže velmi dobře klasifikovat data, na kterých byl natrénován, ale výrazně hůře klasifikuje data testovací. To je ostatně zřejmé z křivek grafu. Je také vidět, že více trénovacích dat by mohlo přispět ke zlepšení přesnosti.



Obr. 8 - Overfitting

Underfitting (nedotrénování)

Underfitting znamená, že zvolený model není dostatečně komplexní na to, aby dokázal správně klasifikovat předložená data (např. prokládání kvadratických dat přímkou). Křivky učení se v tomto případě vyznačují tím, že přesnost na trénovací a testovací sadě je po překročení určitého množství dat velmi podobná. V takovém případě nemá příliš smysl snažit se získat více dat, ale bude lepší zaměřit se na ty parametry modelu, které zajistí lepší komplexitu.



Obr. 9 - Underfitting

3 Praktická část

3.1 Použitá data

V případě učení s učitelem je nezbytné mít dostatek dat, na kterých je možné systém natrénovat a otestovat. Je obvyklé třídit data do následujících tří skupin:

- **Trénovací sada**

  Data, na kterých je natrénován model s různými volbami parametrů.

- **Validační sada**

  Data, která jsou předložena již natrénovanému modelu. Na základě výsledků je vybrán model s nejlepšími parametry.

- **Testovací sada**

  Na základě této množiny se žádné parametry nevybírají, slouží čistě jen pro kontrolu přesnosti.

V této práci je k dispozici datový korpus TOEFL11 *(Test of English as a Foreign Language),* který byl navržen přímo pro úlohu NLI. Tato data obsahují 12 100 textů, které jsou napsány v anglickém jazyce na předem vybraný seznam témat. Autory jsou lidé, jejichž rodný jazyk patří do jedné z následujících kategorií:

arabština (ARA), čínština (CHI), francouzština (FRA), němčina (GER), indština (HIN), italština (ITA), japonština (JAP), korejština (KOR), španělština (SPA), telugština (TEL), turečtina (TUR)

U každého textu víme, jaký je rodný jazyk pisatele. Dále je ke každému textu přiřazeno jedno ze 7 témat a úroveň angličtiny na stupnici od 1 do 9. Texty jsou již předem rozděleny na trénovací sadu (train) a testovací sadu (dev). Datová sada tedy není rozdělena do tří sekcí, které jsou uvedeny výše. Testovací sada v tomto případě slouží rovnou jako validační. Trénovací sada obsahuje 11 000 textů a od každého jazyka je k dispozici 1 000 textů. Testovací sada obsahuje 1 100 textů a od každého jazyka je k dispozici 100 textů.

3.2 Programovací prostředí

- **Python**

  Python je programovací jazyk, který je široce rozšířený v oblasti strojového učení. Spojuje výhody vyššího programovacího jazyka a zároveň je v něm implementována celá řada knihoven, které umožňují psát efektivní kód. Mezi nejznámější knihovny pro numerické výpočty patří NumPy. Kód vytvořený pomocí této knihovny je překládán do jazyků C/C++ a Fortran (10). Díky tomu je kód čitelný, ale i výkonný. Tato práce je vypracována v Pythonu 3.6.

- **SciPy**

  SciPy je systém opensource knihoven v Pythonu pro matematické, vědecké a inženýrské výpočty. Součástí toho balíčku jsou tři základní knihovny (11):

  NumPy – Umožňuje efektivní práci s vektory a maticemi.

  Pandas – Poskytuje nástroje pro organizaci a analýzu dat.

  Matplotlib – Slouží k vykreslování dat do 2D grafů.

- **Scikit-learn**

  Scikit-learn je populární opensource knihovna pro strojové učení naprogramovaná pro užití v Pythonu. Je postavena na základě SciPy ekosystému. Díky tomu všechny datové struktury ze SciPy a Scikit dobře komunikují. Knihovna obsahuje mnoho algoritmů a efektivních nástrojů pro klasifikační i regresní úlohy, shlukovou analýzu, práci s daty a další. Scikit-learn je postaven na knihovnách NumPy, SciPy a matplotlib (12).

- **NLTK - Natural Language Toolkit**

  NLTK je platforma pro práci s jazykovými daty. Obsahuje knihovny pro práci s daty, přes 50 datových sad, ale i již natrénované modely, které lze použít. Knihovny slouží především pro zpracování textů jako např. tokenizace, lemmatizace a další (13).

- **Gensim**

  Gensim open-source knihovna pro Python určená pro analýzu textových dat za účelem nalezení sémantické struktury. V této práci bude využita pro implementaci doc2vec a word2vec modelu (14).

3.3 Postup řešení

3.3.1 Předzpracování dat

Je možné pracovat s originálními texty, ale vhodným předzpracováním dat lze dosáhnout lepších výsledků, můžeme se totiž zbavit šumu anebo naopak získat nové a pro klasifikaci důležité informace. V této práci budou data předzpracována následujícími způsoby:

- **Tokenizace**

  Proces, který člení text složený z písmen, interpunkčních znamének a mezer na jednotlivé izolované tokeny, tj. na slovní tvary a interpunkční znaménka. Výsledkem toho je snazší práce s textem.

  Příklad:

Originální	*There are lots of cars, computers and planes in today's world.*
Tokenizovaný	*There are lots of cars , computers and planes in today ' s world .*

  Texty v tokenizované formě jsou k dispozici již v rámci použité datové sady, není tak třeba originální texty do této podoby převádět.

- **Lemmatizace**

  Proces, při kterém jsou slova převedena na svůj základní tvar (lemma).

  Jak se V NLTK provádí lemmatizace slova je předvedeno na ukázce kódu níže. Tvar výsledného lemma záleží na slovním druhu slova, který lze zavést parametrem *pos* (jak určit slovní druh je popsáno v následujícím bodě).

```
# Příklad, jak vytvořit lemma ze slova
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatizer.lemmatize("dogs", pos = "n"))
[out]: dog


lemmatizer.lemmatize("walking", pos = "v")
[out]: walk
```

  Tento proces může snížit počet příznaků, kterými text popisujeme. Zároveň se tím ale mohou ztratit informace, které jsou pro některé úlohy důležité (koncovky slov, množná čísla, atd.). Je třeba dodat, že nástrojů pro automatickou lemmatizaci je více a mohou stejná slova vyhodnotit různě.

- **POS tagging**

  Jedná se o proces, kde je každé slovo převedeno na značku reprezentující jeho slovní druh. V anglickém jazyce se běžně používá 8 slovních druhů, v českém jazyce jich je 10. Pro NLP úlohy je však výhodné tyto slovní druhy více konkretizovat, abychom zachytili více informací o textu. Použitá knihovna z NLTK převádí anglická slova na jeden z 36 slovních druhů. Těmito slovními druhy jsou například následující: (15)

NN	Podstatné jméno jednotného čísla nebo hromadné *(Noun, singular or mass)*
NNS	Podstatné jméno množného čísla *(Noun, plural)*
NNP	Vlastní jméno jednotného čísla *(Proper noun, singular)*
NNPS	Vlastní jméno množného čísla *(Proper noun, plural)*
VB	Sloveso v základní formě *(Verb, base form)*
VBD	Sloveso v minulém čase *(Verb, past tense)*
RB	Příslovce *(Adverb)*
JJ	Přídavné jméno *(Adjective)*
…	…

```
# Příklad jak vytvořit POS tagy z textu
text = word_tokenize("Now something completely different")
nltk.pos_tag(text)

# Výstupem je následující list, který každému slovu přiřadí POS tag
[out]: [('Now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

- **Slova neobsažená ve slovníku**

  Vlastnosti textu lze reprezentovat i tak, že z textu vybereme jen slova, která nejsou obsažena ve slovníku anglických slov. Je pravděpodobné, že pisatelé anglických esejí budou relativně často dělat chyby na úrovni překlepů, mohlo by se tak jednat o zajímavou vlastnost. Jako slovník použijeme korpus anglických slov z knihovny NLTK, ale lze použít i jiný seznam slov. NLTK slovník často neobsahuje množná čísla, minulé tvary sloves atp. Z toho důvodu budou použity lemmatizované tvary slov.

```
# Příklad jak zkontrolovat, zda se slovo vyskytuje ve slovníku
from nltk.corpus import words

'hello' in words.words()
[out]: True
```

3.3.2 Převod textu do vektorové podoby

Klasifikátory vyžadují, aby na jejich vstup byl přiveden vektor fixní délky. Texty můžeme na takové vektory převést pomocí BOW nebo doc2vec modelu, jak je podrobněji popsáno na str. 15.

Vektory pomocí BOW modelu

Pro vytvoření vektoru textu pomocí tohoto modelu lze v Pythonu využít knihovnu Scikit-learn. Ta obsahuje objekt *vectorizer*, který převádí text na příznakový vektor. Mezi základní vectorizery ve Scikit-learn patří:

- **CountVectorizer**

  Váha termu odpovídá počtu výskytu termu v dokumentu (tf model).

- **TfdfVectorizer**

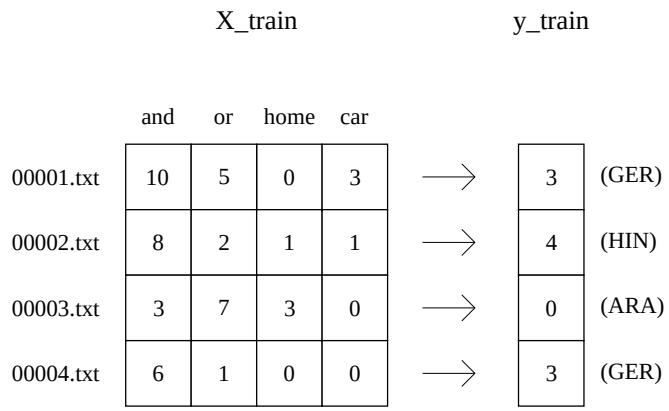
  Váha termu je počítána podle tf-idf modelu.

```
# Příklad jak vytvořit matici termů a dokumentů ve Scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer


# Vectorizer, který jako příznaky bere bigramy slov podle tf-idf modelu.
vectorizer = TfidfVectorizer(ngram_range = (1, 2))


# 1. vytvoření MTD pro trénovací data
# Matici termů a dokumentů získáme předáním množiny textů do vectorizeru.
# fit_transform(train_files) vytvoří MTD na základě všech termů, které se
# nachází v předložených trénovacích datech.
X_train = vectorizer.fit_transform(train_files)


# 2. vytvoření MTD pro testovací data
# z důvodu generalizace je dobré na testovací data použít metodu
# transform() místo fit_transform(). Je to tím, že MTD má být vytvořena
# ze slov, která jsou obsažena jen v trénovací sadě.
# V praxi totiž budou klasifikátoru předkládány texty obsahující slova,
# jejichž váhy nejsou v trénovací MTD obsaženy.
X_test = vectorizer.transform(test_files)
```

Dalším krokem je načíst požadované L1. K tomu lze využít knihovnu pandas.

```
# Načtení požadovaných L1
import pandas

y_train = pandas.read_csv('labels.train.csv').L1
y_test = pandas.read_csv('labels.test.csv').L1
# Získané vektory jsou ve tvaru ['ARA', 'JPN', 'KOR', 'ARA', ….]
# Tyto vektory je třeba před použitím v modelu převést na číselný vektor
# list všech L1:
l1_labels =
      ['ARA','CHI','FRE','GER','HIN','ITA','JPN','KOR','SPA','TEL','TUR']

# vytvoření slovníku {L1:int} podle toho, na jaké pozici je L1 v l1_labels
l1_to_int = dict((c, i) for i, c in enumerate(l1_labels))

# přemapování řetězce na integer
y_train.map(l1_to_int)
y_test.map(l1_to_int)

# výsledné vektory jsou ve tvaru [0, 6, 7, 0, ….]
```

Nyní jsou již data v podobě, ve které je můžeme předat klasifikátoru. Matici termů a dokumentů a vektor požadovaných hodnot si můžeme pro BOW model představit následovně:



Obr.10 - Názorná reprezentace struktury matic X_train a y_train

Vektory pomocí doc2vec modelu

Pro implementaci doc2vec modelu využijeme v Pythonu knihovnu Gensim. Dokument, jehož vektor chceme získat, musí být převeden na objekt *TaggedDocument*, který se skládá ze dvou částí: z listu slov dokumentu a z listu tagů pro dokument. Tagem dokumentu může být cokoli, co jednoznačně definuje text (např. jeho název).

```
# Ukázka jak vytvořit objekt TaggedDocument z množiny textů:
from gensim.models.doc2vec import TaggedDocument

class TaggedLineDocument(object):
    # Iterovatelný TaggedDocument objekt, který bude předán doc2vec modelu

    def __init__(self, doc_list, tags_list):
        # doc_list je list obsahující text dokumentu
        # tags_list obsahuje unikátní tagy pro příslušné dokumenty
        # (v tomto případě název dokumentu)
        self.tags_list = tags_list
        self.doc_list = doc_list

    def __iter__(self):
        for idx, doc in enumerate(self.doc_list):
            # vrací Iterovatelný objekt
            yield TaggedDocument(words=doc.split(),
                  tags=[self.tags_list[idx]])

documents = ['toto je prvni dokument', 'text druheho dokumentu']
document_names = ['0001.txt', '0002.txt']

# vytvoření iterovatelného TaggedLineDocument objektu
iterator = TaggedLineDocument(documents, document_names)

# Dokumenty převedené na objekt TaggedDocument vypadají následovně:
print(list(iterator))
[out]: [TaggedDocument(words=['toto', 'je', 'prvni', 'dokument'],
tags=['0001.txt']), TaggedDocument(words=['text', 'druheho', 'dokumentu'],
tags=['0002.txt'])]
```

Iterovatelný objekt *iterator* lze již použít přímo jako vstup pro trénování modelu. Tento model je ale nejprve třeba vytvořit, což je popsáno na následující straně.

Vytvoření a trénování doc2vec modelu

Model má několik základních parametrů:

dm (int {0, 1}) – definuje použitý trénovací algoritmus. Výchozí hodnota *dm = 1* znamená, že je použit model PV-DM. Pokud *dm = 0*, tak je použit model PV-DBOW (viz str. 18).

size (int) – dimenze příznakového vektoru, který bude reprezentovat dokument

window (int) – maximální vzdálenost mezi aktuálním a předpovídaným slovem v dokumentu.

alpha (float) – počáteční konstanta učení

min_alpha (float) – na tuto hodnotu lineárně klesá počáteční konstanta učení po čas trénování

```
# Vytvoření a natrénování doc2vec modelu
import gensim

# Vytvoření modelu
model = gensim.models.Doc2Vec(size=50,window=8,alpha=0.025,min_alpha=0.001)
# Před samotným trénováním je třeba vytvořit vocabulary
model.build_vocab(iterator)
# Trénování modelu
model.train(iterator, total_examples=model.corpus_count, epochs=20)
```

Výsledkem takového trénování je model, který obsahuje jak vektory pro dokumenty, tak vektory pro slova, která se v dokumentech vyskytují (jelikož byl použit PV-DM model). K těmto vektorům lze přistupovat následovně:

```
# Vektor pro dokument s tagem 00012.txt
model.docvecs['00012.txt']
# Vektor pro slovo "good"
model['good']
```

Pokud máme k dispozici trénovací a testovací sadu (což máme), je dobré spojit tyto sady dohromady a do modelu vložit všechna data naráz. Získané vektory pak můžeme znovu rozdělit na trénovací a testovací množinu (*X_train*, *y_train*). Pokud bychom vytvořili zvláštní model pro trénovací a zvláštní pro testovací sadu, tak by výsledky byly špatné. Je to tím, že sémantické informace o textech by každý model do vektoru zaznamenal jinak.

Příklad: Model pro trénovací data vytvoří vektor $[1, -1, 0, 1]$. Ten samý vektor vytvořený modelem pro testovací data by v sobě nesl úplně jinou informaci o textu.

Pokud tento postup neaplikujeme, tak by klasifikátor natrénovaný na trénovacích datech nedokázal klasifikovat testovací data.

3.3.3 Vytvoření a natrénování klasifikátoru

Modelů pro klasifikaci je mnoho, výhodou Scikit-learn je, že se se všemi pracuje podobně. Všechny klasifikátory sdílejí metody pro natrénování a predikci, což výrazně zjednodušuje práci.

```
# Vytvoření modelu ve Sciki-learn (např. Logistická regrese)
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()


# Pokud bychom chtěli použít LinearSVC klasifikátor, stačí napsat:
from sklearn.svm import LinearSVC
clf = LinearSVC()
```

Vytvořený model je třeba natrénovat:

```
# Natrénování modelu
# Model natrénujeme předáním matice termů a dokumentů a požadovaných výstupů
clf.fit(X_train, y_train)
```

3.3.4 Evaluace modelu

Abychom mohli změřit úspěšnost modelu, je třeba získat předpovědi L1 pro testovací texty.

```
# předpovídání L1 u testovacích textů
y_pred = clf.predict(X_test)
```

Nyní můžeme změřit přesnost modelu podle metrik uvedených na str. 29

```
import sklearn.metrics


# Accuracy
metrics.accuracy_score(y_test, y_pred)
# Confusion matrix
metrics.confusion_matrix(y_test, y_pred)
# Precision, recall, f1-score
metrics.classification_report(y_test, y_pred)
```

3.3.5 Úprava parametrů modelu

Jednotlivé klasifikátory mají různé parametry, které je možné ladit. Cílem je najít takové nastavení parametrů, které poskytne nejlepší výsledky. Ve Scikit-learn existuje pro tento účel objekt *GridSearchCV*, který hledání nejlepších parametrů zjednodušuje. Stačí napsat všechny parametry, které chceme vyzkoušet, do formy slovníku a *GridSearchCV* najde nejlepší konstelaci parametrů.

```
# Ukázka jak najít nejlepší parametry klasifikátoru
from sklearn.model_selection import GridSearchCV

# vytvoření klasifikátoru, který chceme otestovat
clf = LinearSVC()

# parametry, které chceme zkusit, napíšeme do slovníku
parameter_grid = {'loss': ['hinge', 'squared_hinge'],
                  'C': [0.1, 0.3, 0.5, 0.8, 1]}

# Parametry předáme do objektu GridSearchCV a model natrénujeme
grid_search = GridSearchCV(clf, parameter_grid, scoring='accuracy')
grid_search.fit(X_train, y_train)

# zobrazení parametrů pro model s nejlepšími výsledky
parameters = grid_search.best_params_
```

4 Prezentace výsledků

Tato kapitola je rozdělena do dvou částí: První část je věnována výsledkům dosažených pomocí BOW modelu. Ve druhé části jsou prezentovány výsledky dosažené pomocí doc2vec modelu.

4.1 Výsledky dosažené BOW modelem

4.1.1 Porovnání klasifikátorů

Před tím, než začneme ladit příznaky textů a parametry klasifikátorů, je dobré udělat si rychlý náhled na to, které klasifikátory budou pro tento problém fungovat nejlépe. Jednotlivé klasifikátory se mohou lišit v přesnosti, ale i ve výpočetní náročnosti, což může být u velkých dat klíčový parametr. Nevýhodou některých algoritmů může být i to, že fungují jako tzv. černá skříňka (Black box). Návrhář tak nemá možnost podívat se, které příznaky jsou pro klasifikátor důležité nebo s jakými pravděpodobnostmi jsou data roztříděna do jednotlivých tříd. Při výběru klasifikátoru je třeba přihlížet ke všem těmto kritériím.

Termy: unigramy slov
Vectorizer: TfidfVectorizer

Klasifikátor	Precision	Recall	F1 score	Accuracy
LinearSVC	**0,783**	**0,777**	**0,776**	**0,777**
SVM (rbf kernel)	0,399	0,335	0,314	0,334
LogisticRegression	0,683	0,664	0,662	0,663
KNN (n = 4)	0,290	0,251	0,244	0,251
Decision Tree	0,208	0,109	0,071	0,109
Perceptron	0,680	0,672	0,664	0,671
Random Forest	0,465	0,472	0,462	0,471

Z tabulky je zřejmé, že LinearSVC poskytuje znatelně nejlepší výsledky. Tento klasifikátor ani nemá výrazný problém s dobou trénování, což nelze říci např. o SVM s rbf jádrovou funkcí. Dalšími kroky je tedy najít optimální parametry pro tento klasifikátor a zkusit volbu různých příznakových vektorů.

4.1.2 Výběr nejlepších parametrů

Jako nejlepší klasifikátor se ukázal být LinearSVC. Dále je třeba určit, které příznaky budou v kombinaci s tímto klasifikátorem fungovat nejlépe. Způsob zpracování příznaků lze ovlivnit různými parametry. V této práci budeme upravovat následujících parametry:

- *ngram_range (min_n, max_n)* **–** Rozsah n-gramů, které mají být vytvořeny**.**
- *max_df (float)* – Ignorovat v modelu termy, které se vyskytují častěji než *max_df*.
- *min_df (float)* – Ignorovat v modelu termy, které se vyskytují méně často než *min_df*.
- *use_idf (boolean)* – Nastavení vah podle tf-idf modelu.
- *sublinear_tf (boolean)* – Aplikace *sublinear tf scaling*, tzn. nahradit *tf* za *1 + log(tf)*.
- *C (float)* – regularizační parametr klasifikátoru. Vysoká hodnota může způsobit, že klasifikátor bude přetrénován.

Výběr výsledků testování pro LinearSVC klasifikátor

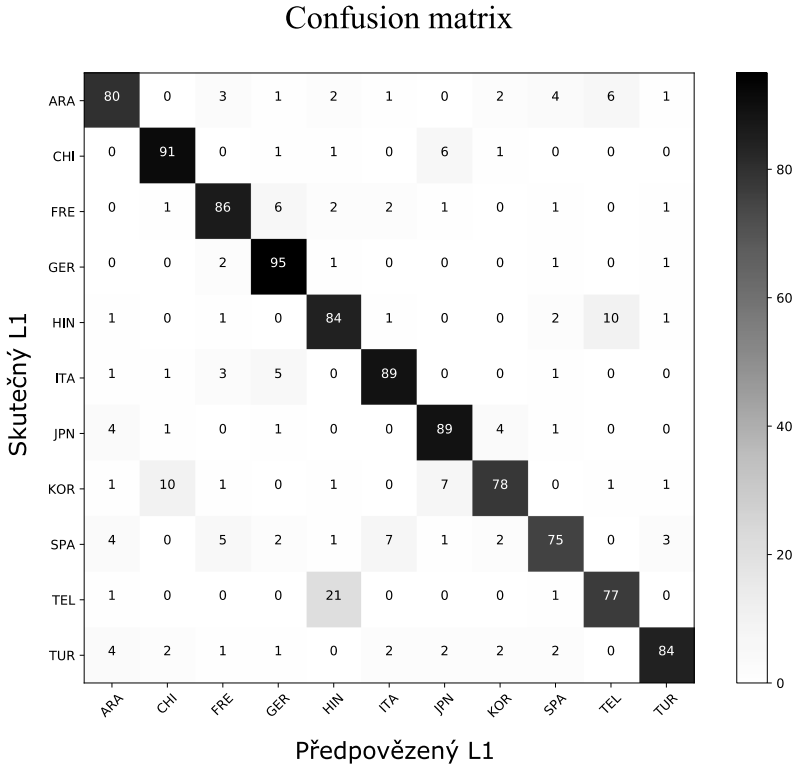
Příznaky	min_df	max_df	sublinear_tf	tf-idf	C	F1-score
lemmatizované trigramy slov	0,0001	0,9999	True	True	1.0	82,4%
tokenizované trigramy slov	0,0001	0,9999	True	True	1.0	82,5%
n-gramy znaků (1-11)	0,001	0,999	True	True	0,8	84,1%
n-gramy znaků (1-11) + trigramy slov	0,001 0,0001	0,999 0,9999	**True**	**True**	**1,0**	**84,3 %**
n-gramy POS (1-7)	0,001	0,999	True	True	0,8	51,0%
unigramy neanglických slov	0.0	1	True	True	0,35	40,0 %

Z výsledků je vidět, že nejlepší přesnosti dosahují modely využívající jako příznaky n-gramy slov a znaků. Také se ukázalo, že lemmatizace slov neměla zásadní vliv na přesnost klasifikace. Nejhorších výsledky dosahují modely využívající POS tagy a neanglická slova.

Ve všech případech bylo dobré použít *tf-idf* model a *sublinear tf scaling*. Parametry *min_df* a *max_df* zlepšily jednak přesnost modelu, tak i výpočetní čas (zredukoval se počet příznaků). Výchozí nastavení regularizačního parametru $C = 1{,}0$ je zřejmě rozumná výchozí hodnota. Malými změnami se v některých případech podařilo mírně zlepšit přesnost modelu.

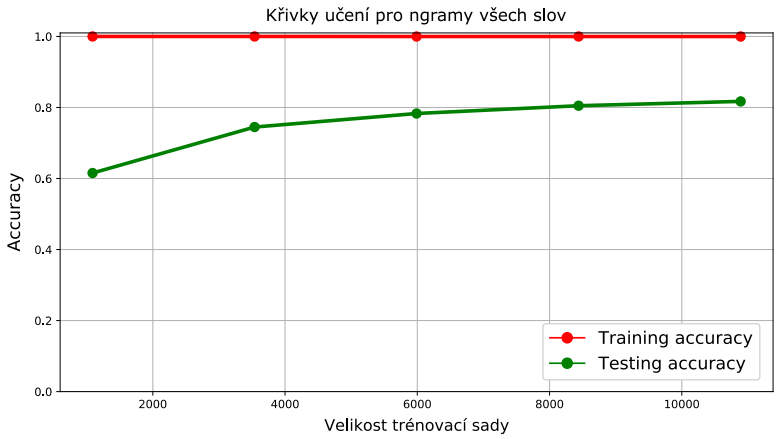
4.1.3 Rozbor nejlepšího modelu

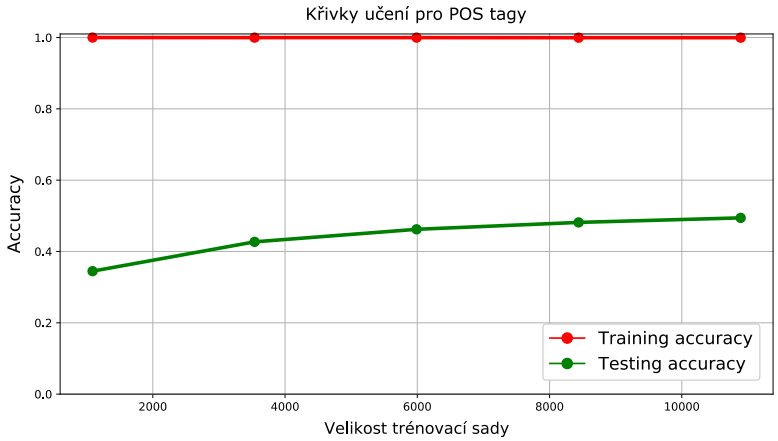
L1	Precision	Recall	F1 score
ARA	0,833	0,800	0,816
CHI	0,858	0,910	0,883
FRE	0,843	0,860	0,851
GER	0,848	0,950	0,896
HIN	0,743	0,840	0,789
ITA	0,873	0,890	0,881
JPN	0,840	0,890	0,864
KOR	0,876	0,780	0,825
SPA	0,852	0,750	0,798
TEL	0,819	0,770	0,794
TUR	0,913	0,840	0,875
Průměr	**0,845**	**0,844**	**0,843**

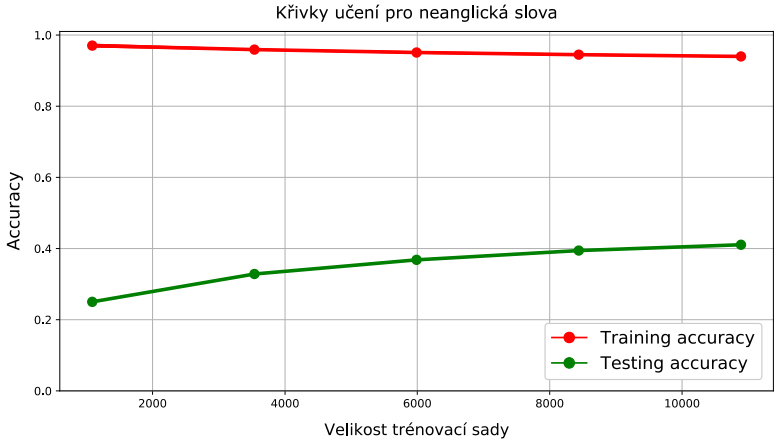


Nejlepší výsledky jsou dosahovány při klasifikaci textů, kde rodný jazyk autora je němčina (GER). Naopak největší chyba nastává v případě klasifikace hindštiny (HIN) a telugštiny (TEL). Špatné výsledky jsou dosahovány též při klasifikaci španělštiny (SPA), kde bylo dosaženo dobré hodnoty precison (0,852), ale znatelně horší hodnoty recall (0,750). Výraznější chyby nastaly také při klasifikaci mezi jazyky čínština - korejština, čínština - japonština a japonština - korejština.

4.1.4 Křivky učení







Z grafů těchto křivek lze usoudit, že ve všech případech došlo k přetrénování. Modely využívající POS tagy a n-gramy slov mají dokonce 100 % přesnost i na trénovací sadě čítající 11 000 textů, což jen tvrzení o přetrénování potvrzuje. Přetrénování lze zmírnit změnou regularizačního parametru C, přesnost se ovšem může výrazně zhoršit. Lze očekávat, že s větším množstvím dat by došlo k velmi malému zlepšení přesnosti ve všech případech.

4.1.5 Vliv příznaků na klasifikaci

Pro lepší porozumění klasifikátoru je dobré vědět, podle čeho se klasifikátor rozhoduje. Pro tyto účely existuje v Pythonu knihovna *lime*. Knihovna spolupracuje pouze s těmi klasifikátory, které ukládají proměnnou *predict_proba*, ve které jsou uloženy pravděpodobnosti pro jednotlivé třídy.

Nejlepší výsledky byly dosaženy LinearSVC klasifikátorem, který tuto vlastnost ovšem nemá. Proto jsou prezentované výsledky předvedeny na klasifikátoru SVC s lineární jádrovou funkcí. Tento klasifikátor je velmi podobný LinearSVC, rozdíl je ve vztahu pro ztrátovou funkci. Jako příznaky jsou zvoleny n-gramy slov a neanglická slova. Natrénování SVC klasifikátoru pro nejlepší příznaky (n-gramy slov a znaků) bylo výpočetně příliš složité.

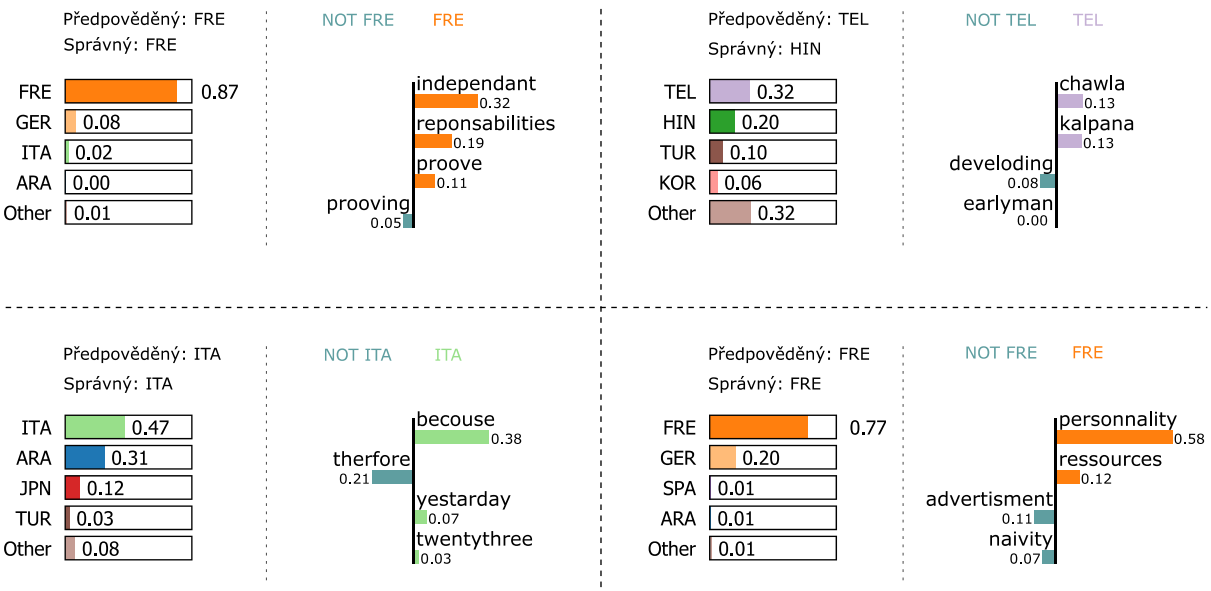
Pokud zvolíme jako příznaky n-gramy slov, dosahuje tento klasifikátor F1-skóre 79,8%. V případě neanglických slov je F1- skóre 36,8 %.

```
# Kód pro vizualizaci příznaků (již předpokládá vytvořený vectorizer
# a matice X_train, y_train)

from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from lime.lime_text import LimeTextExplainer


clf = SVC(kernel = 'linear', probability=True)
clf.fit(X_train, y_train)


# použití knihovny lime pro vykreslení příznaků pro daný text
c = make_pipeline(vectorizer, clf)
explainer = LimeTextExplainer()
exp = explainer.explain_instance(text_to_explain, c.predict_proba,
      num_features=4)
```

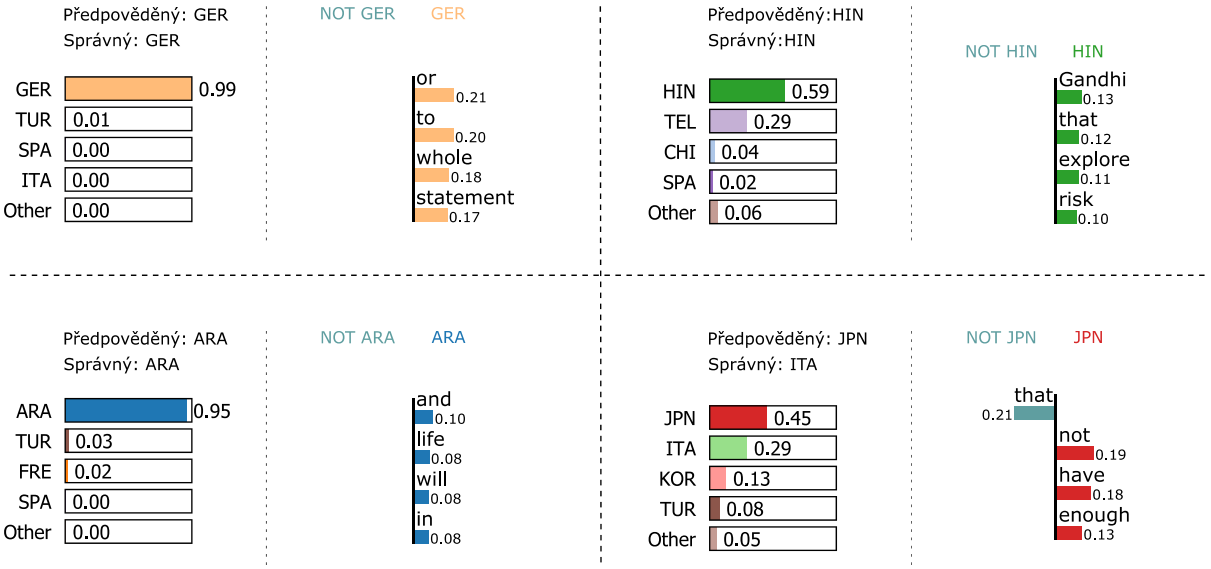
Vizualizace výsledků

Na obrázku na další straně jsou ukázka toho, jaké příznaky hrály důležitou roli při klasifikaci. U každého z textů můžeme vidět tabulku s pravděpodobnostmi, které klasifikátor přiřadil jednotlivým L1. K nejpravděpodobnějšímu L1 jsou přiřazena 4 nejdůležitější slova, podle kterých se klasifikátor rozhodl.

Klasifikátor využívající neangliká slova (F1score 36,8 %)



Klasifikátor využívající všechna slova (F1score 79,8 %)



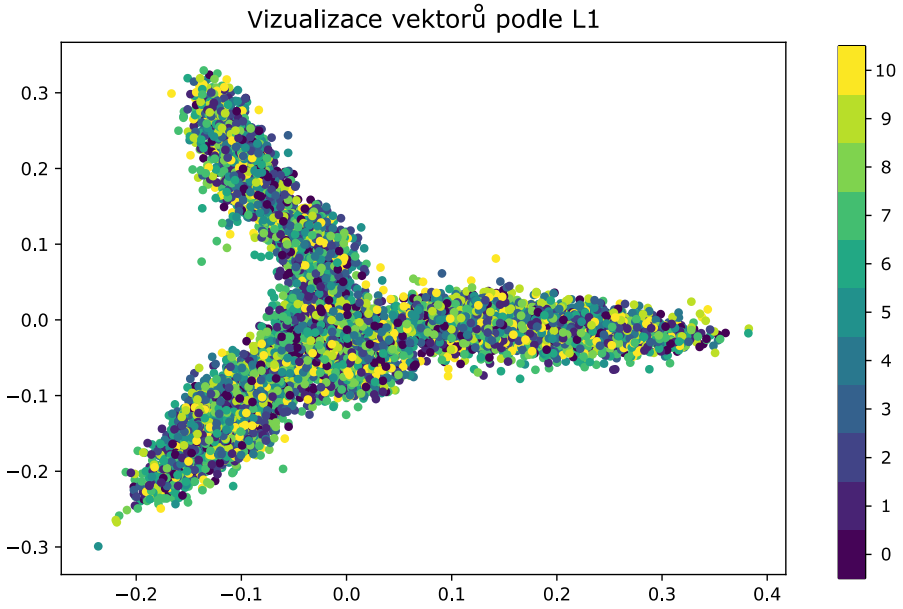
Smyslem tohoto obrázku je názorně ukázat, že i horší klasifikátor využívající neanglická slova může poskytnout užitečné informace. Lze si všimnout, že text, jehož L1 je francouzština, byl klasifikován na základě slova *independant*, což je francouzský překlad slova *independent*. Můžeme se tedy například domnívat, že Francouzi budou mít se správným hláskováním tohoto slova větší problémy, než jiné národnosti. Podobných příkladů lze najít více.

V případě klasifikace pomocí všech slov vidíme, že klasifikátor si je mnohem jistější v předpovědích. Pohledem na důležité příznaky se ovšem nedozvíme téměř nic o tom, jakým

stylem píše člověk určité národnosti, jaké dělá chyby atp. Většina slov, podle kterých se klasifikátor rozhoduje, s rodným jazykem nijak nesouvisí. Potvrzuje se tak předpoklad, že pravděpodobně došlo k přetrénování modelu. Zajímavý poznatek lze ovšem pozorovat u hindštiny, kde se klasifikátor rozhodl na základě slova *Gandhi*, což je logické. Klasifikátor se navíc rozhodoval mezi jazyky HIN a TEL. Jelikož oba jazyky pochází z podobné geografické oblasti, lze očekávat, že slovo *Gandhi* se bude vykytovat u obou jazyků často. Z toho nejspíše pramení velká chybovost klasifikace mezi jazyky HIN-TEL. Příznaky podobného typu (názvy měst, geografických oblastí, atd.) je možné pozorovat i u jiných jazyků. Z toho důvodu nejspíše dochází i častěji k chybné klasifikaci JPN-KOR, jelikož obě země si mají geograficky a kulturně blíže, než k jiným zemím.

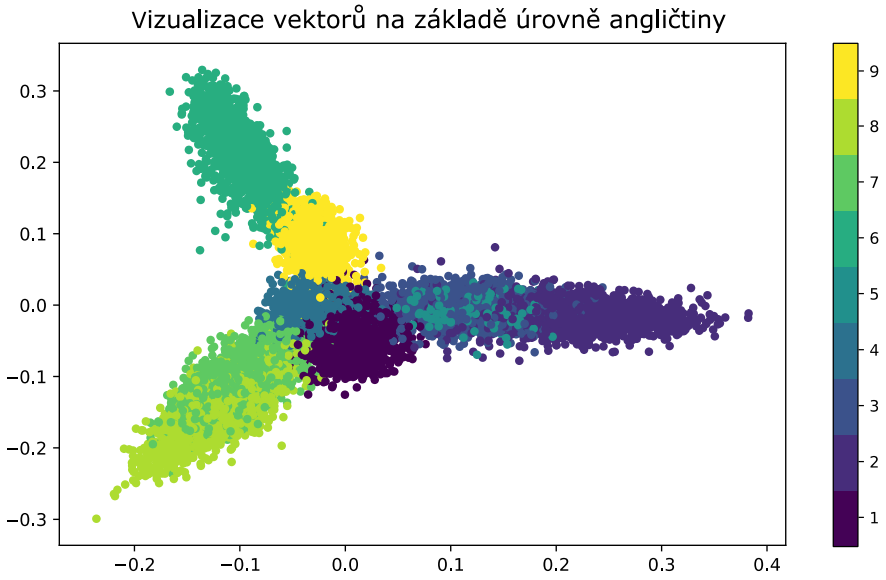
4.1.6 Vizualizace příznakových vektorů

Příznakové vektory lze pomocí *Principal Component Analysis* (PCA) promítnout do 2D prostoru. Rozmístění těchto vektorů v prostoru si můžeme zobrazit v grafu. Ke každému textu je přiřazen nejen rodný jazyk, ale i úroveň angličtiny pisatele, která je hodnocena na stupnici od 1 do 9. Vizualizací příznakových vektorů si lze vytvořit rychlou představu o tom, zda mezi vektory existují nějaké závislosti.



Obr.11

Z *Obr. 11* lze vidět, že texty se stejným rodným jazykem žádné významné shluky netvoří, ale i tak se podařilo dosáhnout přesnosti klasifikace okolo 83%. Projekcí do 2D se samozřejmě ztratila část informace o rozmístění vektorů v původním n-dimenzionálním prostoru. Dále zkusíme vizualizovat dokumenty podle společné úrovně angličtiny (viz. *Obr. 12*).



Obr.12

Je vidět, že texty se stejnou úrovní angličtiny tvoří jasně ohraničené shluky. Lze tak očekávat, že v příznakových vektorech je informace o úrovni angličtiny zachycena velmi dobře. Tuto vlastnost lze v NLI využít.

4.2 Výsledky dosažené doc2vec modelem

4.2.1 Porovnání klasifikátorů

Nejprve zvolíme nějaké počáteční parametry modelu a zkusíme, který klasifikátor bude s tímto modelem pracovat nejlépe.

Výsledky pro model s parametry
size=100, window=8, alpha=0.025, min_alpha=0.0001, epochs=20

Klasifikátor	Precision	Recall	F1 score	Accuracy
LinearSVC	0,460	0,464	0,458	0,463
SVM (rbf kernel)	**0,495**	**0,493**	**0,491**	**0,492**
LogisticRegression	0,468	0,469	0,468	0,469
KNN (n = 8)	0,264	0,200	0,190	0,200
GaussianNB	0,411	0,401	0,398	0,401
Perceptron	0,355	0,327	0,304	0,327
RandomForest	0,216	0,209	0,208	0,209

Nejlepší výsledky jsou dosaženy klasifikátory SVM (rbf kernel), LinearSVC, LogisticRegression. Rozdíl v přesnosti mezi těmito klasifikátory není velký, při hledání nejlepšího modelu tedy budeme uvažovat všechny tři.

Pozn.: Porovnáním s výsledky u BOW modelu si lze názorně ukázat, že různé klasifikátory poskytují různé výsledky pro jiný model. Potvrzuje se tak, že nelze předem říci, co bude fungovat nejlépe. V případě BOW modelu byl LinearSVC jednoznačně nejlepší, zde tomu tak není.

4.2.2 Výběr nejlepších parametrů modelu

Během testování byly vyzkoušeny různé dimenze příznakového vektoru. V tabulce jsou zachyceny nejlepší parametry pro každou z dimenzí.

size	window	alpha	min_alpha	epochs	F1-score		
					Lin. SVC	SVM-rbf	Log. Reg.
100	8	0,025	0,0001	60	55,1 %	57,2 %	55,1 %
200	10	0,025	0,0001	100	62,3 %	59,0 %	64,2 %
300	10	0,025	0,0001	100	66,6 %	65,7 %	66,6 %
400	10	0,025	0,0001	100	66,3 %	68,4 %	65,7 %
500	**10**	**0,025**	**0,0001**	**100**	**67,1 %**	**69,6 %**	**67,4 %**
600	10	0,025	0,0001	60	66,8 %	65,5 %	66,52 %

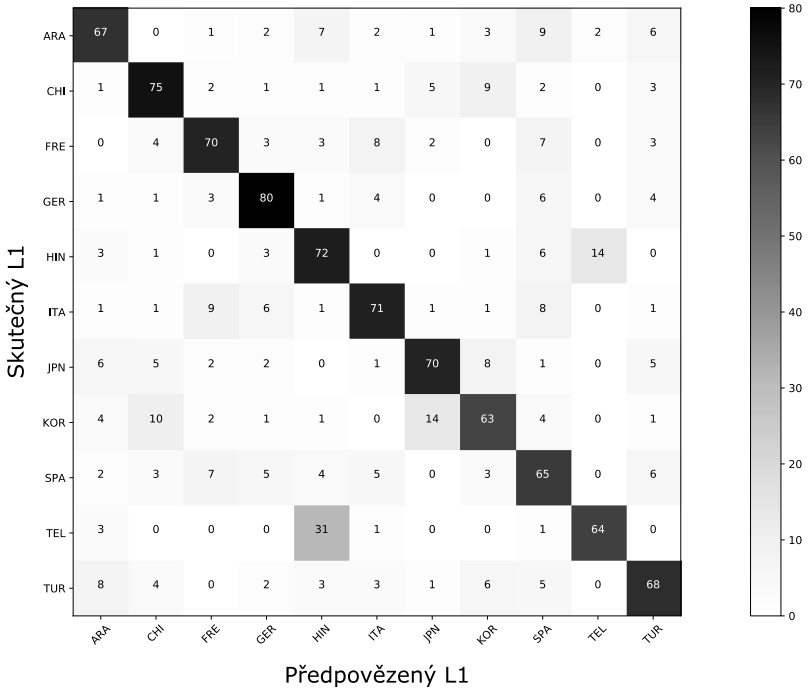
Jako nejlepší doc2vec model se ukázal být ten, který vytváří příznakový vektor o 500 složkách. Nejlepší výsledek byl dosažen SVM klasifikátorem s rbf jádrovou funkcí. Doba trénování SVM byla delší než u LogisticRegression nebo LinearSVC, ale na dostupné datové sadě proběhlo natrénování v rozumném čase. V případě většího množství dat by to už mohl být problém.

Je nutno poznamenat, že i když je dosažené F1 skóre o 14,7 % horší než v případě BOW modelu, tak se jedná o velmi dobrý výsledek. Ke klasifikaci totiž bylo využito jen 500 příznaků, zatímco v případě nejlepšího BOW modelu měl příznakový vektor přibližně 1 300 000 hodnot.

4.2.3 Rozbor nejlepšího modelu

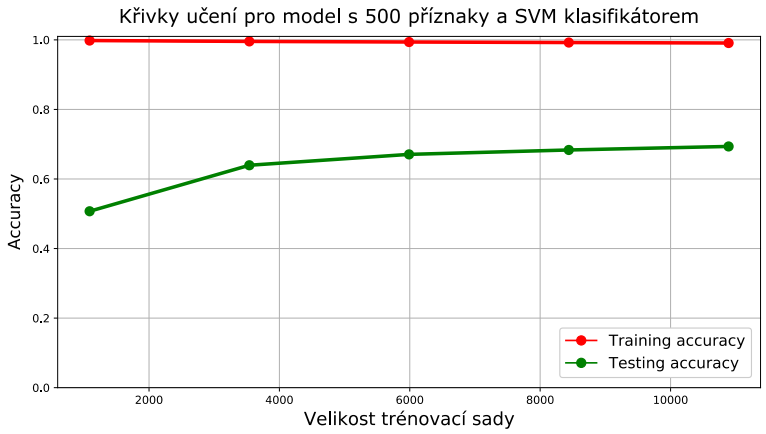
L1	Precision	Recall	F1 score
ARA	0,698	0,670	0,684
CHI	0,721	0,750	0,735
FRE	0,729	0,700	0,714
GER	0,762	0,800	0,780
HIN	0,581	0,720	0,643
ITA	0,740	0,710	0,724
JPN	0,745	0,700	0,722
KOR	0,670	0,630	0,649
SPA	0,570	0,650	0,607
TEL	0,800	0,640	0,711
TUR	0,701	0,680	0,690
Průměr	**0,701**	**0,695**	**0,696**

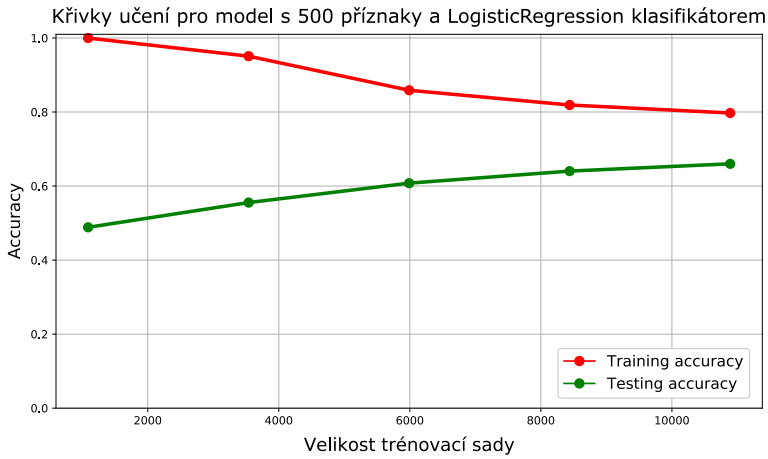
Confusion matrix

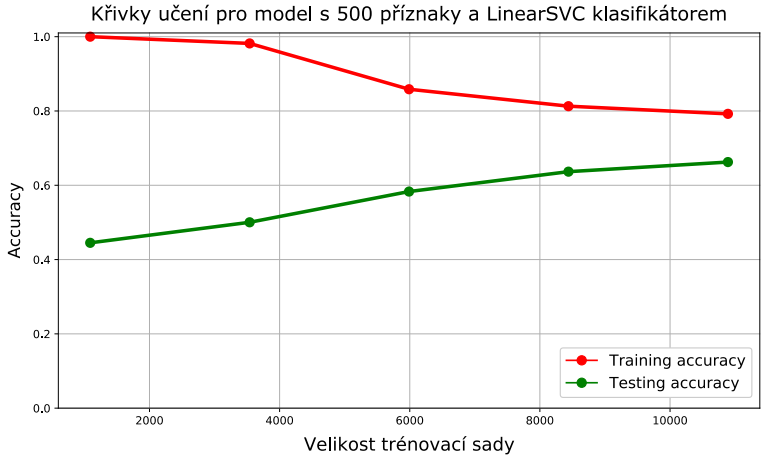


Porovnáme-li dosažené výsledky s těmi, kterých jsme dosáhli pomocí BOW modelu, tak je zřejmé, že největší chyby klasifikace se vyskytly u stejných jazyků. Největší problém opět tvoří rozlišení mezi jazyky hindština - telugština. Další výraznější chybovost můžeme sledovat mezi jazyky japonština - korejština. Naopak nejlepšího skóre jsme opět dosáhli při klasifikaci němčiny. Je tak zajímavé pozorovat, že i když byly příznakové vektory vytvořeny na základě jiného principu, tak má klasifikátor problém se stejnými jazyky.

4.2.4 Křivky učení







Z grafů lze vidět, že model klasifikující pomocí SVM měl sice nejlepší přesnost, ale došlo k přetrénování. To neplatí u modelů, které využívají LinearRegression a LinearSVC, ty přetrénované nejsou. V praxi tedy bude pravděpodobně lepší použít tyto nepřetrénované modely. S větším množstvím trénovacích dat nelze očekávat výrazně lepší přesnost.

4.2.5 Vizualizace příznakových vektorů

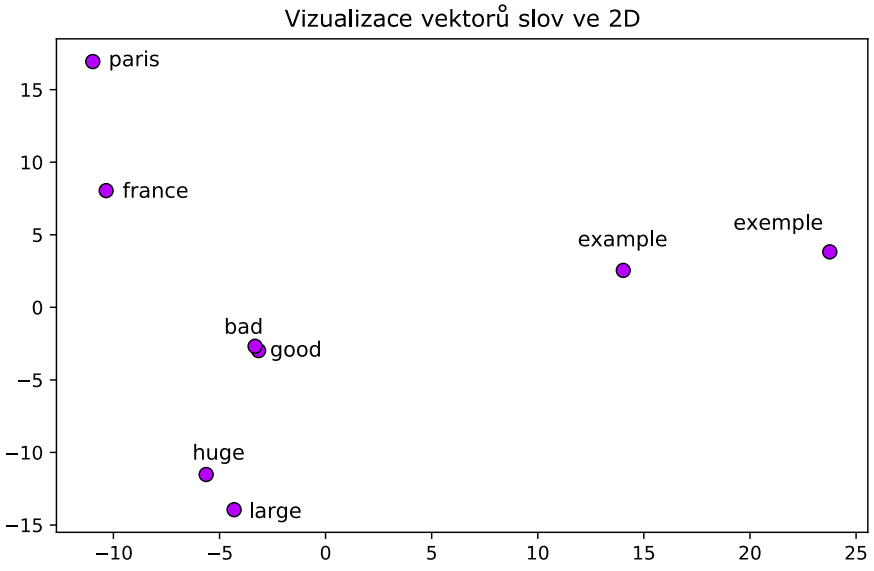
Jelikož jsme k vytvoření vektorů dokumentů použili model PV-DM, tak jsme zároveň vytvořili vektory i pro všechna slova, která se v datové sadě vyskytla. Toho můžeme využít například k nalezení slov, která mají podobné sémantické vlastnosti. Podobnost mezi dvěma slovy (vektory) se měří pomocí kosinové podobnosti (*cosine similarity*), která je v následující tabulce uvedena v závorce. Identická slova mají podobnost 1.

Podobnost slov

Example	Speak	Good	Gandhi	Korea
Instance (0,771)	Talk (0,274)	Better (0,338)	Mahatma (0,474)	Japan (0,239)
Exemple (0,614)	Communicate (0,241)	Bad (0,301)	Nelson (0,306)	Korean (0,238)
Exaple (0,427)	Interact (0,218)	Best (0,289)	Lincoln (0,256)	Taiwan (0,224)
Ex (0,414)	Write (0,204)	Easy (0,273)	Mandela (0,256)	Turkey (0,207)
Exmple (0,413)	Discuss (0,201)	Different (0,271)	Theresa (0,253)	Germany (0,194)

Ke každému ze slov je přiřazeno 5 nejpodobnějších slov. Je vidět, že mezi slovy je jasná souvislost. Lze i vypozorovat, ke kterým překlepům docházelo v textu často (např. *example* - *exemple*). Tuto skutečnost bylo možné vyčíst i z BOW modelu, kdy byly jako příznaky použita neanglická slova. Můžeme si též všimnout, že nejpodobnější ke slovu *Korea* je slovo *Japan*. Z toho také může plynout častá chyba klasifikace mezi korejštinou a japonštinou.

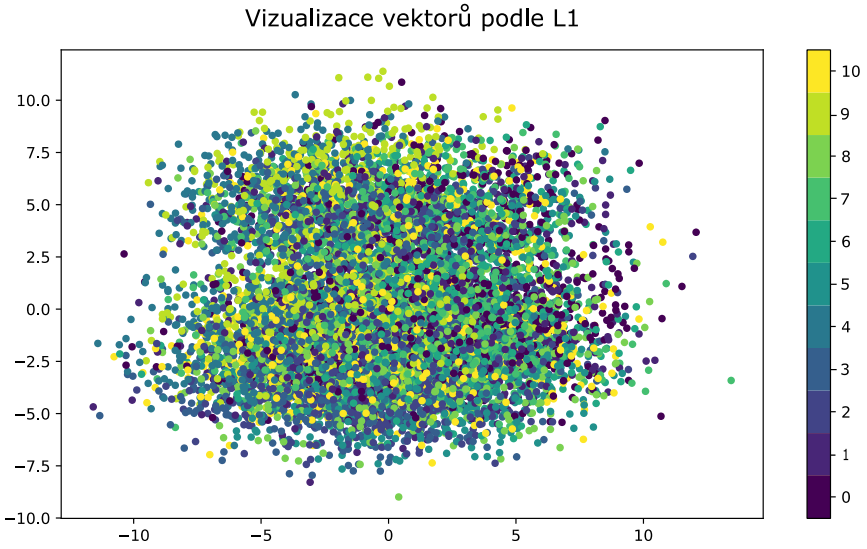
Slova jsou v tomto případě reprezentována vektorem o 500 složkách. Tyto vektory lze promítnout do 2D prostoru. Vektory o dvou složkách již můžeme vynést do grafu (*Obr.13)* :



Obr.13

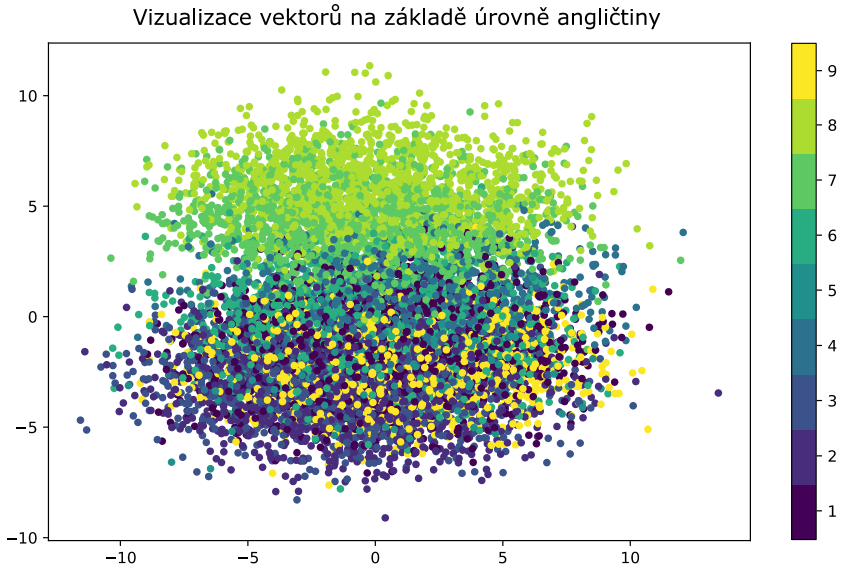
Podobné dokumenty

Stejně jako jsme vizualizovali vektory slov, tak můžeme udělat to samé s vektory dokumentů, viz *Obr. 14*.



Obr.14

Je vidět, že informace o rodném jazyce není ve vektorech zachycena příliš dobře. V grafu sice jsou světlejší a tmavší oblasti, ale výrazné shluky vektorů se společným L1 pozorovat nelze. I tak to ovšem stačilo na úspěšnost klasifikace kolo 67 %. Nyní vykreslíme vektory na základě úrovně angličtiny:



Obr.15

Z *Obr. 15* je zřejmé, že texty se stejnou úrovní angličtiny tvoří shluky. Ty se sice překrývají, ale jistá závislost je zřejmá. Výsledky jsou horší, než při použití BOW modelu (viz str. 38).

5 Závěr

Cílem této práce bylo vyzkoušet různé přístupy pro řešení NLI úlohy. F1-skóre výchozího kódu, který byl dodán organizátory BEA workshopu, je 71,04%. Byly vyzkoušeny dva modely převodu textů na příznakové vektory - BOW a doc2vec. Z hlediska přesnosti klasifikace si lépe vedl BOW model. Nejlepší přesnost byla dosažena v případě, že jsme jako příznaky použili n-gramy slov a znaků a ke klasifikací byl použit LinearSVC klasifikátor. F1 skóre takového modelu je 84,3 %. Podařilo se tak překonat výchozí F1-skóre o 13.26%. V případě doc2vec modelu mělo nejlepší F1-skóre hodnotu 69,6 %. Toto skóre bylo dosaženo SVM klasifikátorem s rbf jádrovou funkcí. Jedná se sice o výrazně horší hodnotu, ale výsledky klasifikátoru nelze hodnotit pouze na základě přesnosti klasifikace. Například vykreslením křivek učení jsme zjistili, že BOW model byl přetrénován, je tedy pravděpodobné, že výsledky na jiných datech by byly výrazně horší. V tomto ohledu si vedl lépe doc2vec model. Vykreslením nejdůležitějších slov pro klasifikaci šlo ukázat, že nejlepší BOW model se rozhoduje podle slov, která s rodným jazykem příliš nesouvisí, výjimku tvořily názvy geografických oblastí či jména osobností (např. Gandhi). Naopak analýzou vektorů vytvořených pomocí doc2vec modelu jsme mohli objevit například časté gramatické chyby nebo překlepy.

I přes to, že doc2vec model měl horší přesnost klasifikace rodného jazyka, může být jeho použití v praxi v mnoha ohledech vhodnější. Je to hlavně tím, že nedošlo k velkému přetrénování a z vektorů slov lze zjistit cenné informace o chybách, které pisatelé často dělají. Ačkoli to nebylo primárním cílem práce, vykreslením příznakových vektorů bylo zjištěno, že v těchto vektorech je dobře zanesena informace o úrovni angličtiny. Toho lze v kombinaci s poznatky z NLI úlohy využít pro lepší personalizovanou výuku cizího jazyka.

Reference

1. Tetreault, J., Blanchard, D. a Cahill, A.*: A Report on the First Native Language Identification Shared Task.* Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications, 2013, pp. 48-57

2. Simeone, O**.:** *A Brief Introduction to Machine Learning for Engineers.* King's College London, Department of Informatics, 2017

3. Berwick, R.: *An Idiot's guide to Support vector machines (SVMs)*. Prezentace ke kurzu *6.034 Artificial Intelligence* na MIT, 2011. Dostupné online na: http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

4. Ircing, P.: *Vektorová sémantika a vyhledávání informací.* Slajdy z přednášek předmětu *Vyhledávání informací (IR)* na ZČU

5. Le, Q. a Mikolov, T.: *Distributed Representations of Sentences and Documents.* Proceedings of the 31st International Conference on International Conference on Machine Learning. Beijing, 2014

6. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. a Dean, J.: *Distributed Representations of Words and Phrases and their Compositionality.* 2013

7. TensorFlow, *Vector Representations of Words*. Odstavec *The Skip-gram model.* [Online, Citováno 12. 2 2018.] https://www.tensorflow.org/tutorials/word2vec

8. Fawcett, T.*: An introduction to ROC analysis*. Pattern Recognition Letters - Svazek 27, vydání 8. Palo Alto, 2005, pp. 861-874

9. Ng, Andrew**.** *Learning Curves*. Výukové video na portále *Coursera.* [Online, Citováno 10 2 2018.] https://www.coursera.org/learn/machine-learning/lecture/Kont7/learning-curves

10. Oliphant, T.: *A guide to NumPy.* USA: Trelgol Publishing, 2006, http://www.numpy.org/

11. Jones E., Oliphant E., Peterson P. a další: *SciPy: Open Source Scientific Tools for Python*, 2001, http://www.scipy.org/

12. Pegregosa, F. a další: *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research 12, 2011, http://scikit-learn.org

13. Bird, S., Loper, E. a Klein, E.: *Natural Language Processing with Python.* O'Reilly Media Inc., 2009, http://www.nltk.org/

14. Rehurek, R. a Sojka, P.: *Software Framework for Topic Modelling with Large Corpora,* Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 2010, pp, 45-50, https://radimrehurek.com/gensim/

15. Santorini, B.: *Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision).* University of Pennsylvania, Department of Computer & Information Science, 1990, pp. 1-6