Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Identifikace parafráze

Plzeň 2016 Duc Vuong Tran

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. června 2016

Duc Vuong Tran

Abstract

The aim of this work is to find and work with algorithms used for Machine Translation. Describe measures and work with them. Learn about different methods for representing word meanings and phrases and also methods for paraphrase identification. Try implement described methods for paraphrase identification and propose appropriate solutions. Methods will be tested with corpus *MRPC (Microsoft Research Paraphrase Corpus)*.

Abstrakt

Cílem této práce je seznámit se s algoritmy používané v oblasti strojového překladu. Dále pak popisovat jednotlivé míry a práce s nimi. Seznámíme se s metodami pro reprezentaci významu slov a slovních spojení. Dále pak s metodami pro identifikaci parafráze. Vybrané metody pro identifikaci parafráze se pokusím implementovat a navrhnout vhodná řešení. Použiji vybrané metody na testování korpusů *MRPC (Microsoft Research Paraphrase Corpus)*.

Poděkování

Za cenné rady a odborné vedení při zpracování bakalářské práce děkuji vedoucímu bakalářské práce Ing. Tomášovi Brychcínovi, Ph.D.

Obsah

1 Úvod

Při převyprávění určitého příběhu, který jsme slyšeli od nějakého vypravěče, je důležité, aby autorovy myšlenky převážily nad ostatními myšlenkami, které v původním příběhu nebyly. Protože totiž chceme, aby k posluchači došel náš příběh takový, který má k originálnímu příběhu nejblíž. Mluvíme o parafrázi.
S parafrází se setkáváme velice často v reálném životě. Lze říct, že parafrázujeme při každé komunikaci. Lidský mozek je ovšem tak inteligentní, že tyto parafráze dokáže identifikovat, popřípadě klasifikovat s vysokou přesností za velmi krátkou dobu. Jak jsou ale na tom stroje?
V tomto projektu se budu zabývat různými algoritmy pro práci s identifikací parafráze.

Zadání

1. Seznamte se s metodami pro reprezentaci významu slov a slovních spojení.
2. Seznamte se s metodami pro identifikaci parafráze.
3. Vybrané metody pro identifikaci parafráze implementujte a pokuste se navrhnout vhodná vylepšení.
4. Důkladně otestujte vybrané metody na korpusu MRPC (Microsoft Research Paraphrase Corpus).

Cílem je seznámit se s různými algoritmy a metodami pro reprezentaci významu slov a práce s identifikaci parafráze. Dále se student naučí pracovat s trénovacími a testovacími daty, tzv. *Korpusy* MRPC (Microsoft Research Paraphrase Corpus).

Parafráze

Parafráze je přeformulování myšlenek a zjištění jiného autora, jejich přepis vlastními slovy a vlastním stylem.

Je nutné změnit jak slova, tak strukturu vět původního díla. Je důležité si uvědomit, že cílem použití parafráze není šetřit slova, protože parafráze

je obvykle stejně dlouhá jako originál. Cílem parafráze je vyjádřit převzaté informace stylem, který je autorovi vlastní a čtenáři již známý. Jiným cílem je také ukázat čtenáři, že jsme plně pochopili myšlenku autora textu, protože jsme schopni ji vyjádřit vlastními slovy nebo-li parafrázovat.
Příkladem parafráze:

1. „Rozdílná řeč ženského a mužského těla: Ženy – stojí k sobě při komunikaci blíž.“
2. „Existuje několik rozdílů v řeči mužského a ženského těla – při komunikování mají mezi sebou ženy menší vzdálenost.“

nebo

1. „Šel ulicí, kde bylo popadané listí.“
2. „Procházel se ulicí pokrytou listím.“

Klasifikace

Klasifikace [11] je v oblasti Machine-learning druh problému, kdy máme určit, do které z předem určených kategorií dané informace patří. Lze říct, že se jedná o třídění nebo zařazování do různých tříd.
Ke klasifikaci jsou obvykle vázány dva procesy - učení a testování. V našem projektu tomu nebude jinak. Je tedy patrné, že kromě testovací množiny, budeme potřebovat také trénovací množinu.
Existují různé druhy klasifikací, například:

- **Binární klasifikace** klasifikuje do dvou tříd. Je to základní a nejjednodušší úloha.
- **Diskrétní klasifikace** zařazuje informace do více než dvou tříd.
- **Jednotřídní klasifikace** dostává informace pouze z jedné třídy a má určit „outliery“ (odlehlé hodnoty) neboli anomálie. Používá se pro detekci anomálií a detekci novinek.
- **Vícelabelová klasifikace** přiřazuje ke každé informaci obecně víc tříd.
- **Fuzzy klasifikace** určuje pravděpodobnost příslušnosti k jednotlivým třídám.

V tomto projektu budeme pracovat spíše s binární klasifikací, kde dvě zmíněné třídy jsou *„Je parafráze“* a *„Není parafráze“*.

2 Použité knihovny a algoritmy

V této kapitole budou uvedeny různé knihovny a algoritmy, které byly použity v tomto projektu. Nepoužité algoritmy a míry, které lze v některých z těchto knihovnách najít, nebudou proto popisovány.

Knihovny

V projektu byly použity tři knihovny. Jedna knihovna pro klasifikátor s názvem *Brainy* a dvě ostatní knihovny obsahují různé algoritmy a míry, které lze použít pro identifikace parafráze.

Brainy

Brainy [5] je knihovna napsaná v jazyce Java. Její autorem je Ing. Michal Konkol, Ph.D., který se zabývá Nature Language Processing na Západočeské univerzitě v Plzni.

Brainy se skládá ze tří hlavních komponentů. Hlavní komponent je komponent pro strojové učení - Machine learning component. Machine learning component poskytuje rozhraní pro základní procesy strojového učení jako je například klasifikace a shlukování. Tento komponent používá datové struktury (matice, vektory) z jedné části z matematického komponentu - Math component. Math component má dvě části, z nichž jedna je efektivní implementace lineárních algebraických struktur a algoritmů, druhá zase obsahuje optimalizační algoritmy. Posledním komponentem je komponent pro zpracování příznaků - Feature processing component. Tento komponent spravuje tvorbu matic a vektorů z uživatelských objektů.
Prvním krokem je extrakce příznaku - Feature extraction. Je to transformace uživatelských objektů do matic a vektorů. Pro tento účel má *Brainy* rozhraní pro uživatelsky definované příznaky. Lze nadefinovat běžné příznaky pro různé úkoly. Další krok je vytváření instance trénovací třídy pro požadovaný algoritmus a proces učení. Trénovací třídy potom zpracují data a vytvoří klasifikační objekt, který je připravený pro další použití.
Pro shlukování má *Brainy* jen jedno rozhraní se dvěma využitími. Všechny algoritmy pro shlukování mohou zpracovávat objekty. Některé z nich doká-

žou vytvořit nový objekt se stejným rozhraním, které je trénováno a může zpracovávat další objekty.

DKPro

DKPro [1] je projekt, který pochází z laboratoře *Ubiquitous Knowledge Processing Lab* (UKP) v Německu od profesora Iryna Gurevych.
Je to velký obalovací projekt, který obsahuje rostoucí počet dalších podprojektů v oblasti zpracování přirozeného jazyka. Tedy různé procesy jako například lingvistické předzpracování, strojové učení, lexikální databáze, atd. Podprojekty v *DKPro* mají na sebe vzájemné vztahy a často na sebe navazují.
Do naší bakalářské práce byl tento projekt naimplementován jako samostatný modul, který obsahuje různé algoritmy pro vyhodnocování parafráze. Konkrétně se jedná o tyto algoritmy:

- Greedy String Tilling
- Jaccard index
- TF-IDF
- N-Gram Containment

Popisy těchto algoritmů jsou uvedeny na dalších stránkách této kapitoly. Dále byla využita jedna metoda pro výpočet vzdálenost vektoru - *Cosine distance*.

Phrasal

Stanford Phrasal [4] je statistický phrasebased (založený na fráze) systém pro strojový překlad napsaný v jazyce Java. Poskytuje snadno použitelné API pro zavádění nových dekódovacích modelů příznaků. Má schopnost překládat pomocí fráze, které obsahují mezery a podmíněné extrakce tabulek frází. Pro účely bakalářské práce *Phrasal* obsahuje několik algoritmů pro určování metrik mezi zdrojem a překladem. V našem případě to budou dvě věty a skóre metriky bude výsledek daného algoritmu. Z této API byly použity tyto algoritmy:

- TERp - Rozšířená verze algorimy *TER*, který je definován jako počet úprav potřebný k transformaci reference (původní dokumenty) do překladu (porovnávací dokumenty). *TERp* nabízí další operace na základě znalosti parafráze a synonym.

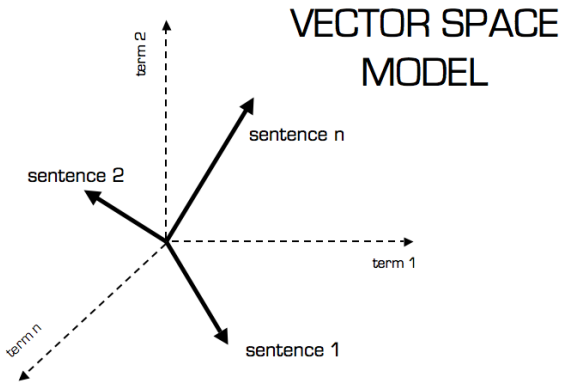
- SLTER Gain - Další rozšířená verze algoritmy *TER*.
- BLEU - Velmi používaná míra v oblasti strojového překladu. Počítá počet shod n-gramů. Tato míra spoléhá jen na shody slov a nemá žádný přehled o synonymech nebo parafrázích.
- BLEU Gain - Rozšířená verze algoritmy *BLEU*.

Algoritmy

V této části budou popisovány jednotlivé algoritmy, které byly naimplementovány a použity. Některé algoritmy byly naprogramovány, existují i algoritmy, které byly staženy a použity z knihovnách.

TF-IDF

TF-IDF[7] je metodika hodnocení relevance používaná pro získávání informací v textu. Toto hodnocení popisuje důležitost slova pro daný dokument v celé kolekci dokumentů nebo v korpusu. Hodnota důležitosti roste úměrně v závislosti na počtu výskytu daného slova v dokumentu, který je dále kompenzován s frekvencí slova v celém korpusu. Tato metoda se používá v *VSM (Vector space model)* jako způsob ohodnocení vektorů (Obrázek 2.1).



Obrázek 2.1: Cílem je získat ohodnocený vektor pro každou větu [zdroj: *christianperone.com*].

Metoda TF-IDF je vhodná při filtrování stop-words v různých oborech, včetně sumarizace a klasifikace textu.

Pro výpočet hodnoty TF-IDF používáme tento vzorec:

$$tfidf(w,d) = tf(w,d) \cdot idf(t) \tag{2.1}$$

TF hodnota

TF je zkratka pro *Term frequency*. Již z názvu je patrné, že se jedná o četnosti určitého slova vzhledem k danému dokumentu.
Lze říct, že se jedná o důležitost slova pro daný dokument. Pokud se nějaké slovo v jednom dokumentu vyskytuje vícekrát, znamená to, že je pro daný dokument důležitý a tím i jeho TF hodnota bude vyšší. Existují různé vzorečky pro výpočet TF hodnoty. Vždy se bude jednat o nezápornou reálnou hodnotu v intervalu $tf(w,d) \in \langle 0;1 \rangle$:

$$tf(w,d) = \frac{c(w,d)}{|d|} \tag{2.2}$$

nebo

$$tf(w,d) = 1 + \log(c(w,d)) \tag{2.3}$$

kde $c(w,d)$ je četnost slova w v dokumentu d a $|d|$ je počet slov v dokumentu d.
V našem projektu byl vybrán první vzorec pro implementaci.

IDF hodnota

IDF je zkratka pro *Inverse document frequency*. Na rozdíl od TF hodnoty IDF hodnota určuje důležitost slova v rámci celé kolekce dokumentů. Jsou to slova, která se vyskytují vícekrát ve více dokumentech. Jsou to převážně předložky, spojky a zájmena. Zde platí opak. Slovo, které se vyskytuje všude, bude méně důležité a bude tedy mít hodnotu IDF nižší.
Pro výpočet IDF hodnoty existuje tento vzorec. Znovu se bude jednat o nezápornou hodnotu:

$$idf(w) = \log \frac{|\mathbf{D}|}{|\{d \in \mathbf{D} : w \in d\}|} \tag{2.4}$$

kde $\mathbf{D}$ je kolekce dokumentů a $|\mathbf{D}|$ je počet dokumentů v kolekci. V jmenovateli je počet dokumentů, kterých obsahuje slovo w.

N-Gramové algoritmy

U těchto metod nejčastěji hledáme shody n-gramů. N-gramy jsou posloupnosti slov jdoucí za sebou o přesném počtu n. Máme tedy unigramy pro n rovno 1, bigramy pro n rovno 2, atd.

Příklad:

Původní věta {"This is an example."}
Trigramy {"This", "is", "an"}{"is", "an", "example"}{"an","example","."}

Jaccard index

Také známý jako *Jaccard koeficient podobnosti* je hodnota používaná k porovnávání podobnosti a rozmanitosti různých množin. Tento koeficient měří podobnost mezi dvěma konečnými množinami slov a je definován jako velikost průniku těchto množin děleno velikost jejich sjednocení.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{2.5}$$

kde A a B jsou množiny n-gramů ve zdrojovém a porovnávacím dokumentu.

Containment

Podobně jako *Jaccard index*, *Containment measure* počítá velikost průniku, ale ta se poté znormalizuje podezřelou (porovnávací) množinou. Tato míra je vhodná pro nalezení párů dokumentů (podobné dokumenty) s různými délkami.

$$C(A,B) = \frac{|A \cap B|}{|B|} \tag{2.6}$$

kde A a B jsou opět množiny n-gramů ve zdrojovém a porovnávacím dokumentu.

Greedy String Tilling

Tento algoritmus se používal k detekci plagiátorství. Identifikuje nejdelší plagovanou sekvenci podřetězců z textu dokumentu pro podrobnější porovnání. Nevýhodou tohoto algoritmu je výrazně pomalejší rychlost zpracovávání.

Word2Vec a Glove

Word2Vec

Word2Vec je dvouvrstová neuronová síť, která zpracovává texty.
Přijme do vstupu textový korpus a jako výstup vrátí skupinu vektorů: *Feature vectors* nebo-li *Příznakové vektory* pro každé jednotlivé slovo vyskytující se v korpusu. *Word2Vec* převede text do numerické podoby, které *deep nets* rozumí a dokáže s tímto formátem dále pracovat.

Cílem *Word2Vec* je seskupování vektorů slov s podobnými významy ve vektorovém prostoru. To znamená, že detekuje podobnost slova matematicky. Vytváří vektory, které numericky znázorňují příznaky slova, například kontexty jednotlivých slov. To všechno zpracovává bez lidského zásahu.
Pokud bude k dispozici dostatečné množství dat a kontextů, *Word2Vec* může velmi přesně odhadnout význam slov na základě jejich posledních výskytů. Tyto odhady mohou být použity k seskupování různých vektorů slov, například vektor *muž - žena* a vektor *král - královna* nebo k shlukování dokumentů a klasifikací dle témat.
Výstup z neuronové sítě *Word2Vec* je slovník, v kterém je ke každému prvku přiřazen jeden vektor, který lze dále využívat k pokročilejšímu strojovému učení nebo pro jednoduchou detekci vztahů mezi slovy. K detekci vztahů dvou vektorů používáme kosinovou vzdálenost.
Word2Vec používá stejný vzorec pro výpočet skóre podobnosti jako *Glove*. Tento vzorec bude popisován níže.

GloVe

GloVe, nebo-li *Global Vectors for Word Representation* [9] je algoritmus učení pro získávání vektorů reprezentující slova. Podobně jako *Word2vec*, pro tento algoritmus existuje slovník nejčastějších slov, ke kterým je přeřazen příslušející vektor.
Pro výpočet skóre podobnosti u *Word2Vec* a *Glove* použijme tento vzorec:

$$\vec{A} = \sum_{j=1}^{n} tf(w_j, d) \cdot idf(w_j) \cdot \vec{v}_j \tag{2.7}$$

Kde w_j je slovo na indexu j ve větě d a A je výsledný vektor pro tuto větu. Výsledek získáme výpočtem kosinové vzdálenosti mezi prvním a druhým vektorem.

Meteor

Jde o algoritmus pro automatické vyhodnocování metriky strojového překladu podle zařazování zdrojů k jednomu nebo více překladu. Zařazování je založeno na různých kriteriích mezi slovy a frázemi jako například synonymy. *Meteor* [3] nabízí vlastnost identifikace parafráze pro následující jazyky: angličtina, čeština, němčina, francouzština, španělština a arabština. *Meteor* je napsáný v jazyce Java a nevyžaduje instalaci nebo závislosti k vyhodnocování skóre metriky.

3 Míry pro měření úspěšnosti

Pro měření výsledků se dají v oblasti Machine Translation využívat různé míry a metody. Lze použít klasickou *úspěšnost* anebo míra F_1. Obě míry byly naimplementovány do tohoto projektu.

Úspěšnost

Nejjednodušší způsob vyjádření výsledku je úspěšnost nebo-li *Classification Accuracy*. Je to reálné číslo pohybující mezi 0 a 1 a vypočítává se jako počet správně klasifikovaných případů děleno celkovým počtem případů. Výsledek se dále převede do formátu procenta po vynásobení číslem 100.

F_1 míra

Další používaná míra pro měření úspěšnosti je známá jako *F measure* nebo *F score*. Tato míra využívá dvě hodnoty *Precision* a *Recall* pro vypočítání skóre úspěšnosti.

- Precision

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{3.1}$$

- Recall

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{3.2}$$

Kde

- True Positives (TP) - Počet správně klasifikovaných parafrází.
- False Positives (FP) - Počet špatně klasifikovaných parafrází.
- False Negatives (FN) - Počet špatně klasifikovaných ne-parafrází.

Výsledný vzorec pro míru F_1 je:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.3}$$

Vektorová vzdálenost

Výstupem algoritmů, které byly popisovány v předchozí kapitole je reálná hodnota, která reprezentuje skóre podobnosti mezi dvěma větami. Existují však i algoritmy, které vrátí dva reálné vektory, například *TF-IDF*, *Glove* nebo *Word2Vec*. Pro identifikaci budeme potřebovat najít takovou hodnotu, která bude jako u ostatních algoritmů popisovat podobnost, popřípadě vzdálenost mezi těmito vektory. V matematice existuje pojem *„Vektorová vzdálenost“*.
Vektorová vzdálenost popisuje vzdálenost dvou vektorů a mezi nejznámější vzdálenosti patří *Euklidovská vzdálenost* a *Kosinová vzdálenost*.

Euklidovská vzdálenost

Pro *Euklidovskou vzálenost* si lze představit jako *Pythagorovu větu* aplikovanou pro dva body ležící v n-rozměrném prostoru:

$$m_e(A, B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2} \tag{3.4}$$

kde A_i a B_i jsou prvky vektoru A a B.

Kosinová podobnost

Kosinová podobnost je velmi populární a hodně používaná metoda v oblasti *Paraphrase identification* k výpočtu podobnosti. Určuje kosinus úhlu mezi dvěma vektory.
Pro výpočet kosinové vzdálenosti použijeme tento vzorec:

$$cos(\theta) = \frac{A \cdot B}{\parallel A \parallel \cdot \parallel B \parallel} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.5}$$

kde podobně jako u Euklidovské vzdálenosti jsou A_i a B_i prvky vektoru A a B.
Přestože kosinová podobnost má jednoduchou implementaci, tak funguje velice dobře a byla využita v této bakalářské práci k výpočtu vzdálenosti mezi vektory, respektive podobnosti dokumentů nebo vět.

4 Data a Korpusy

Pro realizaci tohoto projektu jsou nezbytné trénovací a testovací data. Tyto soubory se využívají v různých případech, které budou později zmíněny.

CzEng 1.0

Přesnější název zní *Czech-English Parallel Corpus, verze 1.0*[2].
CzEng 1.0 je čtvrté vydání paralelního anglicko-českého korpusu zkompilovaného na ÚFAL (Ústav formální a aplikované lingvistiky). Je volně k dispozici pro výzkumné nekomerční účely. CzEng 1.0 obsahuje 15 milionů paralelních vět, respektive 233 milionů anglických a 206 milionů českých tókenů.
V tomto projektu byla použita **modifikovaná verze** tohoto korpusu, kdy z korpusu byly odstraněny veškeré české věty a tókeny. To znamená, že obsahuje už jen anglické věty.
Nová verze korpusu obsahuje 14 milionů řádků textu s celkovou velikostí 1.2GB. Nová verze je oproti originálu obsahově poměrně menší, ale stále se jedná o soubor s obrovským množstvím textu a má tedy nevýhodu s přenositelností.
Texty uvnitř souboru prošly procesem zvaným *Tokenizace textu*, který znamená rozdělení korpusu na jednotlivé slovní tvary - *tókeny*.

Microsoft Research Paraphrase Corpus

Již z názvu je patrné, že se jedná o korpusy[2] od velice známé společnosti zvanou *Microsoft*. Respektive tyto korpusy pocházejí od výzkumné divize společnosti *Microsoft*, která se jmenuje *Microsoft Research*. Byla založena v roce 1991. Zabývá se řešením obtížných problémů světa díky technologickým inovacím ve spolupráci s akademickými, vládními a průmyslovými výzkumníky.
Na webové stránce[1] jsou tři korpusy ke stažení:

- *msr_paraphrase_data.txt* - 10949 trénovacích vět.
- *msr_paraphrase_train.txt* - 2753 parafrází z celkových 4076 párů.

[1] http://research.microsoft.com/en-us/downloads/607D14D9-20CD-47E3-85BC-A2F65CD28042/default.aspx

- *msr_paraphrase_test.txt* - 1147 parafrází z celkových 1725 párů.

Soubor s názvem *msr_paraphrase_data.txt* se používá podobně jako již zmíněný soubor *CzEng 1.0* k učení a vytváření slovníku. Liší se však v obsahu. Texty v tomto korpusu ještě neprošly procesem *Tokenizace*, tedy slova v textu nebudou ještě rozdělena. Podobně tak platí i pro soubory *msr_paraphrase_train.txt* a *msr_paraphrase_test.txt*. Kromě toho, texty v korpusu jsou v určitém formátu. V jednom řádku jsou texty rozděleny tabulátorem do sloupců: Sentence ID, String, Author, URL, Agency, Date a Web Date.
Soubory *msr_paraphrase_train.txt* a *msr_paraphrase_test.txt* mají stejný formát obsahu, texty uvnitř jsou rozděleny opět tabulátorem do sloupců: Kvalita, id první věty, id druhé věty, první samostatná věta a druhá věta. Soubor *msr_paraphrase_train.txt* je pro učení hranice, která se potom použije pro identifikaci parafráze u souboru *msr_paraphrase_test.txt*.

5 Zpracování dat

Fáze zpracování

Jako první bylo nutné mít předzpracovaná data, která budeme využívat v fázi trénování a testování.
Zde máme možnost zpracovávat data z korpusu MRPC, který obsahuje poměrně malé množství dat. Má tak velkou výhodu v rychlosti zpracování, ale z pohledu přesnosti budeme potřebovat obsahově rozsáhlejší trénovací korpus. Jako náhradu jsem si vybral korpus *CzEng 1.0*, který je obsahově mnohem rozsáhlejší a tudíž i vhodnější pro tento projekt.
V této fázi je cílem projít trénovací korpus a vybrat 100 000 nejčetnějších slov. Zároveň si u každého slova poznamenávat počet vět, ve kterých se dané slovo vyskytovalo. Budeme potřebovat vhodnou datovou strukturu pro ukládání těchto slov ve tvaru slovníku, kde klíče budou jednotlivá slova a hodnoty budou počty zmíněných vět. Nejvhodnější datová struktura v tomto případě je *Hashmap*.

Poznámky:

- Protože procházení trénovacího korpusu může trvat poměrně dlouhou dobu, je vhodné tento slovník ukládat do textového souboru a v případě potřeby ho znovu načíst ze souboru (V projektu je slovník předzpracovaný a uložený v souboru *dict.txt*).
- Jako první experiment do součtu pro dané slovo byl vybrán počet vět, ve kterých se slovo vyskytovalo, ale po konzultaci se zadavatelem jsme dospěli k tomu, že bude vhodnější jako součet požadovat počet výskytu slova v trénovacím korpusu.
- Jednotlivé věty jsou rozděleny tečkou, ale pro zjednodušení a rychlejší zpracování je každá věta považována jako jeden samostatný řádek.

Trénování a testování

K trénování budeme potřebovat soubor *msr_paraphrase_test.txt* a k testování soubor *msr_paraphrase_test.txt*. Níže budou uvedeny kroky pro procesy trénování a testování zmíněných souborů. Jako příklad byl vybrán algoritmus *TF-IDF*, který bude popisován později.
Jak již víme, pro testování pomocí algoritmu *TF-IDF* je třeba spočítat *IDF* a *TF* hodnotu. Hodnota *IDF* je v rámci celého dokumentu, kdežto hodnota *TF* je zase vázána na konkrétní větu. To znamená, že hodnota *IDF* v našem případě bude konstantní a nebude se měnit, neboť jako trénovací soubor budeme používat jen soubor s názvem *en.txt*. Toto tvrzení však neplatí pro hodnotu *TF*, která závisí na danou testovací větu, proto se tato hodnota vypočítává až po výběru dvou testovacích vět.
Prvním krokem v této fázi bude procházení slovníku uloženým v datové struktuře *Hashmap* pro výpočet *IDF* hodnot. Výsledkem tohoto kroku je znovu *Hashmap*, který bude obsahovat jednotlivá slova jako klíč a jednotlivé hodnoty budou *IDF* hodnoty.
V druhém kroku se postupně vybírají páry vět pro výpočet *TF* hodnot. Tyto hodnoty tvoří x-rozměrný vektor (rozměr vektoru závisí na počet unikátních slov v jednom testovacím páru), který se poté vynásobí s určitými *IDF* hodnotami a výsledný vektor bude obsahovat *TF-IDF* hodnoty.
Zde v posledním kroku použijeme zmíněnou kosinovou podobnost pro výpočet vzdáleností dvou vektorů. Výsledkem bude reálná hodnota v intervalu $cos(\theta) \in \langle -1; 1 \rangle$, o které budeme následně rozhodovat, zda je parafráze, nebo není, podle námi zvolené hranice. Ovšem máme možnost využít trénovací soubor *msr_paraphrase_train.txt* pro výpočet aritmetického průměru hodnoty *TF-IDF* a ten považovat jako hranice pro testovací případy.

Lemmatizace

Trénovací a testovací korpusy

- *msr_paraphrase_train.txt*
- *msr_paraphrase_train.txt*
- *msr_paraphrase_train.txt*

jsou jednoduše soubory s anglickými větami. Aby tyto věty dávaly smysl, jsou některá slova těchto větách v různých tvarech. Ale pro naše využití, je třeba, aby tato slova byla v základním tvaru, tzv. - lemma. Konkrétně se jedná o základní podobu lexému (tedy slova nebo fráze), která se uvádí jako reprezentativní ve slovnících.
Například v angličtině máme slovo - *play*, které může mít různé tvary - *plays, playing, played*, a podobně. To samé v češtině, kdy pro slovo *hrát*, máme další tvary - *hraje, hrál* a jiné. Je velice nevhodné, aby náš klasifikátor tato slova identifikoval jako různá slova. Řešením by mohl být lemmatizace.
V procesu lemmatizace se dané slovo převede do základního tvaru, které umožní lépe porozumět textu klasifikátoru. K lemmatizaci je možné používat nástroj zvaným *CoreNLP*. Tento nástroj kromě lemmatizace nabízí ještě další různé úpravy a práce s texty, jako například tokenizace textu, rozdělení vět, parsování závislosti, apod. Nástroj lze stáhnout z oficiální stránky Stanfordu.

6 Způsoby reprezentace příznaku

Teď už víme, že do klasifikátoru *Brainy* vstupuje víc příznaků od různých algoritmů. Každý algoritmus vyhodnotí nejen jiné výsledky, ale dokonce i v různých intervalech, tedy výsledky jednotlivých příznaků nejsou na sobě nějak závislé a mohl by to být pro *Brainy* problém.
Naštěstí *Brainy* umí přijmout výsledky příznaků i ve vektorové podobě. A nejen to, dokáže zkombinovat víc formátů. To znamená, že do *Brainy* můžeme poslat reálné hodnoty a zároveň také vektory.

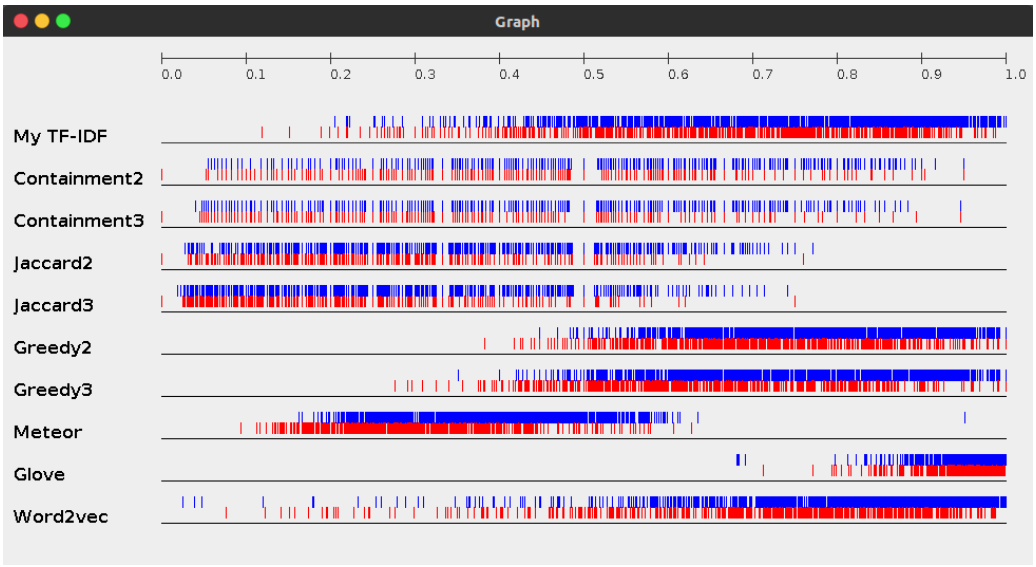
Příznak jako reálná hodnota

Nejjednodušší případ je nechat výsledky z příznaků beze změn, tedy nechat je v reálné číselné podobě. Výsledky budou mít různé rozsahy. U jednodušších algoritmů bývá horní hranice rovna 1, u jiných (např. BLEU Gain) může horní hranice být ostře větší než 1.
Zde tedy čelíme problémům s normalizováním a vztahy mezi výsledky od různých příznaků.

Příznak jako binární vektor

Jak už bylo zmíněno, reálné hodnoty mají několik problémů, a proto je třeba zavést jiný způsob reprezentace výsledky příznaku, další způsob je výsledek reprezentovat binárním vektorem.

K vytváření vektoru je třeba mít trénovací data. Z těchto dat nasbíráme seznam hodnot, který nazveme histogramem a z něho určíme interval, ve kterém se budou výsledky jednotlivých příznaků vyskytovat. Abychom mohli převést reálnou hodnotu na vektor, budeme muset určit dvě hranice intervalu (dolní a horní). Dále je třeba tolik hranic v závislosti na počtu dimenze vektoru. Je tedy třeba rozdělit získaný interval na určitý počet menších podintervalů. Existují dva způsoby rozdělení intervalu.



Obrázek 6.1: Ukázka výskytu hodnot z různých příznaků. Modrá barva reprezentuje výskyt parafráze a červená zase ne-parafráze.

O konstantní délce

První způsob rozdělení je rozdělení do menších intervalů o konstantní délce. Je to nejjednodušší způsob, kdy nám stačí znát jen dolní a horní hranici intervalu.
U vektoru o konstantní délce nebude ani zapotřebí mít seznam seřazený. Ke zjištění nejmenšího a největšího prvku stačí seznam procházet jen jednou. Poté spočítáme vzdálenost mezi těmito dvěma hodnotami a vydělíme počtem dimenzí. Pro převod reálné hodnoty je třeba zjistit, do jakého podintervalu daná hodnota patří. V tomto místě přiřadíme vektoru hodnotu 1, všude jinde hodnotu 0.
Výhodou toho způsobu je jednoduchá a rychlá implementace. Nevýhodou je nízká přesnost. Pokud zvolíme menší počet dimenzí, vektory budou velice podobné, tedy nepřesné.

S relativními odchylkami

Podle získaných hodnot lze zjistit, že vrácené hodnoty se nevyskytují v intervalu rovnoměrně. Po vizualizaci (Obrázek 6.1) lze vidět například u algoritmu *TF-IDF* nebo *Greedy String Tilling*, že všechny hodnoty jsou v intervalu $< 0; 1 >$. Ale většina z nich se vyskytuje v podintervalu $< 0.4; 1 >$. Je tedy velice nevhodné, aby měly všechny podintervaly stejnou délku. Proto je vhodné další způsob rozdělení. A tedy rozdělení s relativními odchylkami.

U tohoto způsobu rozdělení budou třeba určité procesy s histogramem. Cílem je najít pro každý podinterval vhodnou délku a ta závisí na frekvence výskytu. Chceme, aby každý podinterval měl stejný počet výskytů. Ten vypočítáme tak, že celkový počet výskytů vydělíme počtem podintervalů. Další krokem je uspořádání seznamu a zapisování hranic podintervalů ve chvíli, kdy se v nich nachází potřebný počet výskytů.
Po rozdělení bude mít každý interval jinou délku, tedy v místě, kde se vyskytují nejvíce hodnot, bude interval kratší a delší intervaly se budou nacházet v místě, kde se hodnoty vyskytují méně.

7 Experimenty

V této části budou uvedeny výsledky s použitím jednotlivých algoritmů. Pro měření budou použity míry F_1 a klasická úspěšnost.
Dříve než se začne měřit, je třeba zmínit drobné shrnutí informace o trénovacích a testovacích datech.

Název souboru	Popis
CzEng 1.0	14 milionů řádků, proběhla tokenizace, angličtina
msr_paraphrase_data.txt	10949 vět, angličtina
msr_paraphrase_train.txt	2753 parafrází z 4076 párů, angličtina
msr_paraphrase_test.txt	1147 parafrází z 1725 párů

Tabulka 7.1: Tabulka se shrnutí informace o datech.

Podle informace uvedené z tabulky, je třeba provést tokenizace textu pro tyto tři soubory:

- *msr_paraphrase_data.txt*
- *msr_paraphrase_train.txt*
- *msr_paraphrase_test.txt*

Dále také proces lemmatizace pro všechny čtyři uvedené soubory, tedy data budeme lemmatizovat při trénování i testování. Díky těmto procesům bude klasifikátor *Brainy* fungovat mnohem lépe a spolehlivěji. Výsledky budou tedy výrazně lepší.

TF-IDF

Jako první byl naimplementovaný algoritmus *TF-IDF*. Je to algoritmus, který byl do projektu naimplementován dřív než samotný klasifikátor. Níže uvedené výsledky jsou postupně výsledky algoritmu *TF-IDF* bez použití (Tabulka 7.2) a s použitím klasifikátoru *Brainy* (Tabulka 7.3).

Úspěšnost 66.49% a F_1 míra 79.87% jsou poměrně nízké výsledky. Lze si všimnout, že není vhodné používat klasifikátor *Brainy* jen s jedním algoritmem. Je třeba do klasifikátoru naimplementovat další sofistikovanější algoritmy.

Bez Brainy	0,6514	0,5	0,401
TF-IDF	63,65%	69,22%	70,72%
Dummy	66,49%	66,49%	66,49%

Tabulka 7.2: Tabulka s úspěšností (*Accuracy*) při použití metody *TF-IDF* bez klasifikátoru *Brainy* v porovnáním s metodou *Dummy*.

S Brainy	Úspěšnost	F_1 míra
TF-IDF	66.49%	79.87%

Tabulka 7.3: Tabulka s úspěšností a *mírou* F_1 s použitím klasifikátoru *Brainy*.

Kromě algoritmu *TF-IDF* v první tabulce, najdete i výsledky pro další algoritmus zvaný **Dummy**. Jedná se o algoritmus, který vyhodnocuje testované páry vždy pozitivním výsledkem bez ohledu na význam nebo jakékoliv vlastnosti vstupních dat. To znamená, že výsledek identifikace je vždy pravdivý. Tento algoritmus slouží jen pro porovnání úspěšnosti ostatních algoritmů a není zcela doporučen jako samostatné řešení.

N-Grams a TF-IDF

Při dalším měření byla použita knihovna *DKPro*, která nabízí již zmíněné N-Gramové algoritmy a míry - *Jaccard index*, *Containment measure*, *Greedy String Tilling*. Protože se jedná o N-Gramové algoritmy, je třeba uvést přesný počet daných n-gramu. Zde je tabulka výsledků výše zmíněných algoritmů s použitím n-gramy od 2 do 5 společně s algoritmem *TF-IDF*.

	Reálná hodnota		Vektor	
n	Úspěšnost	F_1 míra	Úspěšnost	F_1 míra
2	67.77%	79.08%	71.42%	80.19%
3	66.67%	79.90%	72.23%	80.91%
4	68%	80.07%	72.64%	80.95%
5	67.65%	80.03%	72.06%	80.75%

Tabulka 7.4: Tabulka s úspěšností a míry F_1 s použitím klasifikátoru *Brainy* a n-gramové algoritmy + *TFIDF*.

Tabulka je rozdělena do dvou části. Levá část jsou výsledky vrácené klasifikátorem *Brainy* s použitím způsobu reprezentace příznaku jako reálná

hodnota. Pravá část tabulky jsou zase výsledky s použitím druhého způsobu reprezentace příznaku, tedy jako binární vektor. První sloupec udává informaci o počtu n-gramů.
Po určitém počtu měření s různými počty rozdělení binárního vektoru se objevovaly nejlepší výsledky, jejichž počet byl roven 8. Budou tedy používány binární vektory s dimenzí 8.
Z tabulky si můžeme všimnout, že nejlepší výsledky se objevovaly u *unigramů* a *trigramů* s použitím způsobu reprezentace příznaku jako binární vektor o dimenzi 8. A jelikož klasifikátor *Brainy* umí kombinovat reálné hodnoty s vektory, bude vhodné tuto možnost využívat a nejlepší nalezená kombinace bude nadále použita jako základ pro další experimenty.

Baseline

V tomto projektu, kromě algoritmů z knihovny *DKPro*, budou použity další algoritmy, a proto je vhodné mít nějaký předdefinovaný základ, tedy skupinu algoritmů, která bude nadále neměnná. Nazveme jí *Baseline*.
Také je třeba zmínit, že všechny příznaky, které využívají způsob reprezentace jako binární vektor, budou mít výsledky jako vektor o dimenzi 8.
Do *Baseline* zařadíme doposud nejlepší kombinaci algoritmů. Jsou to tyto algoritmy:

#	Název algoritmu	n	Příznak
1	TF-IDF		reálná hodnota
2	N-Gram Containment	2	reálná hodnota
3	N-Gram Containment	3	reálná hodnota
4	N-Gram Containment	2	binární vektor
5	N-Gram Containment	3	binární vektor
6	Jaccard index	2	reálná hodnota
7	Jaccard index	3	reálná hodnota
8	Jaccard index	2	binární vektor
9	Jaccard index	3	binární vektor
10	Greedy String Tilling	2	reálná hodnota
11	Greedy String Tilling	3	reálná hodnota
12	Greedy String Tilling	2	binární vektor
13	Greedy String Tilling	3	binární vektor

Tabulka 7.5: Tabulka algoritmu zařazené do *baseline*.

Výsledek našeho *baseline* je - *úspěšnost* **74.26%**, F_1 *míra* **82.05%**.

V porovnáním s původním algoritmem *TF-IDF*, který měl *úspěšnost* 66.49% a F_1 *míru* 79.87% je naše *baseline* s kombinací 13 algoritmů výrazně lepší. To dokazuje to co jsem již zmínil dříve, že klasifikátor *Brainy* funguje lépe s použitím více algoritmů.

Baseline s Phrasal

Další algoritmy nalezené z knihovny zvanou *Phrasal* jsou:

- *BLEU*
- *BLEU Gain*
- *SLTER Gain*
- *TERp*

Z výše uvedených algoritmů byl *BLEU* a *TERp* uveden jako součást obalovacího algoritmu zvaným *MTMETRICS*. Je to algoritmus, který se obsahuje mimo *BLEU* a *TERp* ještě další šest jiných algoritmů. Z něho autor vytěžil až 77.4% úspěšnosti a 84.1% F_1 *míry*. Je to 1 z nejlepších výsledků podle zebřičku na stránce *aclweb.org*.

Název algoritmu	n	Příznak	Úspěšnost	F_1 míra
BLEU	3	Reálná hodnota		
BLEU	3	Binární vektor		
BLEU	4	Binární vektor	74.67%	82.02%
TERp		Reálná hodnota		
TERp		Binární vektor		

Tabulka 7.6: Tabulka s nejlepším nalezeným výsledek při použití *Prasal*.

V tabulce 7.6 jsou uvedeny, které algoritmy z knihovny *Phrasal* byly použity. Jsou tam tři příznaky algoritmu *BLEU* a 2 příznaky algoritmu *TERp*. S pomocí těchto příznaků nám klasifikátor *Brainy* vrátil výsledek - 74.67% úspěšnost a F_1 *míra* 82.02%. Je velice zajímavé, že do nejlepší kombinace nepatří algoritmy *SLTER* a *BLEU Gain*, protože klasifikátor při použití těchto dvou algoritmů vrátil překvapivě nižší výsledek.
Po tomto měření jsme získali lepší výsledek úspěšnosti (*accuracy*), než původní *baseline*, ale F_1 *míra* je stále stejná.

Baseline s Meteor

Jako další algoritmus do experimentu byl vybrán algoritmus *Meteor*. Na rozdíl od *Phrasal* je *Meteor* samostatný algoritmus.

Název algoritmu	Příznak	Úspěšnost	F_1 míra
Meteor	Reálná hodnota	74.61%	82.20%
Meteor	Binární vektor	73.68%	81.53%
Meteor	Reálná hodnota	73.91%	81.65%
Meteor	Binární vektor		

Tabulka 7.7: Tabulka výsledků s použitím algoritmu *Meteor*.

Byly změřeny tři případy. První případ, u kterého jsem získal nejlepší výsledky byla kombinace *baseline* a algoritmus *Meteor* s příznakem jako reálná hodnota. Zde jsem dosáhl až 74.61% úspěšnosti. Je to srovnatelné s předchozím měřením s *Phrasal* a dokonce u F_1 *míry* jsem získal drobné zlepšení. U ostatních případů byly výsledky zhruba o 1% nižší, nicméně výsledky jsou stále vysoké a tedy můžeme považovat algoritmus *Meteor* za velice kvalitní.

Baseline s Phrasal a Meteor

Máme již změřeno *Phrasal* a *Meteor*. Tyto algoritmy a knihovny nám vyhodnotily poměrně dobré výsledky. Zde jsou výsledky získané při používání *Phrasal* a *Meteor* zároveň.

Název algoritmu	n	Příznak	Úspěšnost	F_1 míra
BLEU	3	Reálná hodnota	75.42%	82.62%
BLEU	3	Binární vektor		
BLEU	4	Binární vektor		
TERp		Reálná hodnota		
TERp		Binární vektor		
Meteor		Binární vektor		

Tabulka 7.8: Tabulka s nejlepším nalezeným výsledkem při použití *Prasal* a *Meteor*.

Pokud bude použita kombinace algoritmů uvedených v tabulce 7.8, tedy tři příznaky algoritmu *BLEU*, dvě příznaky algoritmu *TERp* a jeden příznak algoritmu *Meteor*, získáme až 75.42% úspěšnosti a 82.62% F_1 *míry*. To je doposud nejlepší získaný výsledek jak u úspěšnosti, tak i u F_1 *míry*.

Baseline s Glove a Word2Vec

Nyní se podíváme na algoritmy založené na *word embedding*, tedy *Glove* a *Word2Vec*.

Název algoritmu	Příznak	Úspěšnost	F_1 míra
Glove	Reálná hodnota	73.22%	81.48%
Glove	Binární vektor	73.33%	81.53%
Glove	Reálná hodnota	71.48%	80.95%
Glove	Binární vektor		

Tabulka 7.9: Tabulka výsledků s použitím algoritmu *Glove*.

Název algoritmu	Příznak	Úspěšnost	F_1 míra
Word2Vec	Reálná hodnota	74.38%	82.15%
Word2Vec	Binární vektor	74.32%	82.06%
Word2Vec	Reálná hodnota	73.68%	81.75%
Word2Vec	Binární vektor		

Tabulka 7.10: Tabulka výsledků s použitím algoritmu *Word2vec*.

První tabulka (Tabulka 7.9) je tabulka výsledků při měření *baseline* s algoritmem *Glove*. Nejlepší výsledky se pohybují okolo úspěšnosti 73.33% a 81.53% F_1 *míry*. Je to určitě vyšší výsledek než u *TF-IDF*, ale celkově nižší než samostatný *baseline*.
Zlepšení se naopak objevuje u *Word2Vec*, kdy podle výsledků uvedených v druhé tabulce 7.10 jsme dosáhli 74.38% úspěšnosti a 82.15% F_1 *míry*.

Kombinace všech algoritmů

Je na čase najít nejlepší kombinaci všech algoritmů. Zde jsem se pokoušel zkombinovat různé algoritmy, abych získal co nejvyšší výsledky.
Protože počet naimplementovaných algoritmů je poměrně vysoký, bude velice těžké najít tu nejlepší kombinaci. A proto je vhodné používat předchozí experimenty. Do toho seznamu algoritmů určitě zařadíme společně s *baseline* algoritmy *Phrasal*, *Meteor* a *Word2Vec*, u kterých jsme získali velice dobré výsledky.
V tabulce 7.11 jsou uvedeny všechny algoritmy, které byly použity k získání konečného výsledku - úspěšnost **75.94%** a F_1 *míra* **82.80%**. Do tohoto seznamu překvapivě patří mimo jiné i algoritmus *SLTER Gain* a *Greedy String Tilling*. U těchto dvou algoritmů se objevovala zhoršení při měření s *baseline*.

Název algoritmu	n	Příznak	Úspěšnost	F_1 míra
Baseline				
Greedy String Tilling	4	Binární vektor		
Glove		Reálná hodnota	75.94%	82.80%
Word2Vec		Reálná hodnota		
Meteor		Reálná hodnota		
SLTER Gain		Binární vektor		

Tabulka 7.11: Tabulka s doposud nejlepším výsledkem.

Je velice pravděpodobné, že existuje jiná kombinace s vyšším výsledkem, kterou jsem doposud nevyzkoušel. Nicméně tento výsledek je velice dobrý a může se zařadit do lepších algoritmů podle zebříčku na stránce *aclweb.org*, kde zatím kraluje algoritmus *TF-KLD* s neuvěřitelnými 80.4% *accuracy* a 85.9% F_1 *míry*.

Porovnávání s dalšími algoritmy

Na webové stránce [1] lze najít tabulku, ve které jsou uvedeny algoritmy pro identifikace parafráze. Tabulka obsahuje krátké popisy algoritmů. Dále i dosažené výsledky jak úspěšnost tak i F_1 *míra*. Všechny algoritmy na této stránce používají stejná data a korpusy, tedy trénovací a testovací data *Microsoft Reasearch Paraphrase Corpus*.

[1] http://aclweb.org/aclwiki/index.php?title=Paraphrase_Identification_(State_of_the_art)

Na další stránce je tabulka (7.12) s vybranými algoritmy ze výše zmíněné webové stránky.

Pořadí	Název algoritmu	Úspěšnost	F_1 míra
?	Dummy	66.5%	nezměřena
11	Vector-based similarity	73.0%	82.0%
?	**Náš algoritmus**	75.9%	82.8%
6	WDDP	75.6%	83.0%
4	MTMETRICS	77.4%	84.1%
1	TF-KLD	80.4%	85.9%

Tabulka 7.12: Tabulka s vybranými algoritmy z webové stránky *aclweb.org*.

První je algoritmus *Dummy*, který má jen 66.5% úspěšnosti a F_1 *míry*. Opět je třeba zmínit, že se jedná jen o algoritmus pro porovnání. Oproti tomuto algoritmu, má náš algoritmus zhruba o 10% vyšší úspěšnost.
Dále lze na stránce najít například velice známý algoritmus *Vector Based Similarity* [8], tedy algoritmus s využitím kosinové podobností.
Algoritmus, který obsahuje výsledky těsně nad námi se jmenuje *WDDP* [10]. Má sice nižší úspěšnost, ale naopak F_1 *míra* je u něho mírně vyšší.
Dále algoritmus *MTMETRICS* [6], který společně s naším algoritmem sdílí podobné míry. *MTMETRICS* dosáhl výsledek 77.4% úspěšnosti a 84.1% F_1 *míry*. Má tedy celkově lepší výsledek a umístil se na 4. místě.
Jako nejlepší algoritmus, který je v porovnáním s naším algoritmem výrazně lepší, se nazývá *TF-KLD* [12]. Tento algoritmus se umístil na prvním místě už od roku 2013.

Použité nástroje

Celá práce byla vyvíjena pod systémem *Ubuntu 16.04*. Zdrojové kódy jsou v kódování *UTF-8*. Jedná se o projektu v Maven a byla vytvořena ve vývojovém prostředí *Eclipse verze 4.5.2* společně s *JDK 1.7*.

8 Závěr

Nejprve jsem použil algoritmus *TF-IDF*. Výsledná úspěšnost byla okolo 69%. Po té jsem přidal klasifikátor *Brainy* a různé algoritmy spadající do *baseline* a došel jsem k lepšímu výsledku. Zvolil jsem si několik dalších algoritmů, pomocí kterých bych se dopracoval k lepším výsledkům a povedlo se mi dosáhnout finálního výsledku, který jsem zde uvedl.
Vím, že existuje ještě více možností k dosažení "dokonalosti", ale bohužel se mi dosud nepovedlo tyto další možnosti použít. Přesto si myslím, že jsem během své práce dosáhl uspokojivých výsledků a nových poznatků.

Literatura

[1] Bär, D. – Zesch, T. – Gurevych, I. DKPro Similarity: An Open Source Framework for Text Similarity. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, s. 121–126, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. Dostupné z: `http://www.aclweb.org/anthology/P13-4021`.

[2] Bojar, O. – Žabokrtský, Z. – Dušek, O. – Galuščáková, P. – Majliš, M. – Mareček, D. – Maršík, J. – Novák, M. – Popel, M. – Tamchyna, A. The Joy of Parallelism with CzEng 1.0. In *Proceedings of LREC2012*, Istanbul, Turkey, May 2012. ELRA, European Language Resources Association. In print.

[3] Denkowski, M. – Lavie, A. Meteor Universal: Language Specific Translation Evaluation for Any Target Language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*, 2014.

[4] Green, S. – Cer, D. – Manning, C. D. Phrasal: A Toolkit for New Directions in Statistical Machine Translation. In *In Proceddings of the Ninth Workshop on Statistical Machine Translation*, 2014.

[5] Konkol, M. Brainy: A Machine Learning Library. In Rutkowski, L. – Korytkowski, M. – Scherer, R. – Tadeusiewicz, R. – Zadeh, L. A. – Zurada, J. M. (Ed.) *Artificial Intelligence and Soft Computing*, Lecture Notes in Computer ScienceL. : Springer International Publishing, 2014. s. 490–499. Dostupné z: `http://dx.doi.org/10.1007/978-3-319-07176-3_43`.

[6] Madnani, N. – Tetreault, J. – Chodorow, M. Re-examining Machine Translation Metrics for Paraphrase Identification. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, s. 182–190, Montréal, Canada, June 2012. Association for Computational Linguistics. Dostupné z: `http://www.aclweb.org/anthology/N12-1019`.

[7] Manning, C. D. – Schütze, H. *Foundations of statistical natural language processing*. MIT Press, 1999. ISBN 0-262-13360-1.

[8] Milajevs, D. – Kartsaklis, D. – Sadrzadeh, M. – Purver, M. Evaluating Neural Word Representations in Tensor-Based Compositional Settings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, s. 708–719, Doha, Qatar, October

2014. Association for Computational Linguistics. Dostupné z: http://www.aclweb.org/anthology/D14-1079.

[9] Pennington, J. – Socher, R. – Manning, C. D. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, s. 1532–1543, 2014. Dostupné z: http://www.aclweb.org/anthology/D14-1162.

[10] Wan, S. – Dras, M. – Dale, R. – Paris, C. Using Dependency-Based Features to Take the 'Para-farce' out of Paraphrase. In *Proceedings of the Australasian Language Technology Workshop 2006*, s. 131–138, Sydney, Australia, November 2006. Dostupné z: http://www.aclweb.org/anthology/U06-1019.

[11] Wikipedie. Klasifikace (umělá inteligence) — Wikipedie: Otevřená encyklopedie, 2016. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Klasifikace_(um%C4%9Bl%C3%A1_inteligence)&oldid=13447861. [Online; navštíveno 18. 06. 2016].

[12] Yin, W. – Schütze, H. Discriminative Phrase Embedding for Paraphrase Identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, s. 1368–1373, Denver, Colorado, May–June 2015. Association for Computational Linguistics. Dostupné z: http://www.aclweb.org/anthology/N15-1154.

Obsah přiloženého CD

Projekt

V CD najdeme zdrojové kódy projektu a informační textové soubory. Jelikož se jedná o projektu s obrovským množstvím dat, jsou přiloženy jen zdrojové kódy projektu. Tedy v CD není:

- Testovací a trénovací korpusy.
- Externí knihovny.
- Spustitelná aplikace.

Potřebné odkazy jsou již uvedeny v předchozích kapitolách nebo v textovém souboru přiložené v CD.

Dokumentace

- Text - PDF soubor s textem bakalářské práce
- Zdrojové soubory - Soubor v Texu a obrázky.