ČVUT, Fakulta informačních technologií
Vyhledávání na webu a v multimediálních databázích
Letní semestr 2019/2020
Závěrečná zpráva k projektu

LSI vektorový model

David Mašek a Kristýna Klesnilová

8. května 2020

OBSAH

1 Popis projektu

V tomto projektu implementujeme *LSI vektorový model* sloužící k podobnostnímu vyhledávání v databázi anglických textových dokumentů. Tuto funkcionalitu následně vizualizujeme pomocí webového interface, který uživateli umožňuje procházet databázi článků na základě doporučování nejpodobnějších článků k právě čtenému.

V experimentální části projektu jsme se dále zaměřili na:

- Určení optimálního počtu konceptů
- Porovnání vlivu LSI na kvalitu výsledků vyhledávání s ohledem na výskyt synonym a homonym
- Porovnání průchodu pomocí LSI vektorového modelu se sekvenčním průchodem databáze
- Vliv různých vnitřních parametrů na výkon algoritmu (změna počtu extrahovaných termů, použití lemmatizace namísto stemmingu, odstranění číslovek při preprocesingu...)
- Jak se změní výsledky při použití jiného vzorce na výpočet vah termů

Celý náš projekt je volně dostupný na: (Odkaz na gitlab?).

2 Způsob řešení

2.1 Preprocesing dokumentů

Jako první v naší aplikaci začínáme s preprocesingem dokumentů. Slova z jednotlivých dokumentů převedeme na malá písmena a odstraníme z nich nevýznamová slova a interpunkci. K identifikaci nevýznamových slov používáme seznam anglických nevýznamových slov. Jako parametr programu posíláme také, zda má z dokumentů odstranit i číslovky. Následně na zbylé termy aplikujeme *stemming* či *lemmatizaci*. Tím se snažíme slova, která mají stejný slovní základ, vyjádřit pouze jedním termem. Stemming to dělá pomocí algoritmu, kterým odsekává přípony a koncovky slova. Lemmatizace na to jde o něco chytřeji, podle kontextu slova se pokusí určit, o jaký slovní druh se jedná, a podle toho ho zkrátit.[1] Porovnání jejich použití v programu se dále věnujeme v experimentální části.

2.2 Výpočet vah termů

V aplikaci vytváříme matici M_w, která má v řádcích jednotlivé termy a ve sloupcích jejich váhy v jednotlivých dokumentech.

Začneme tím, že si vytvoříme matici počtu výskytů jednotlivých termů v jednotlivých dokumentech. Počet termů v této matici poté dále zredukujeme, abychom pracovali jen s těmi nejdůležitějšími. Funkci pro redukci termů posíláme následující parametry:

- *max_df* - termy nacházející se ve více % dokumentů, než udává číslo 100 * *max_df*, z matice odstraníme
- *min_df* - termy nacházející se v méně nebo stejně dokumentech, než udává číslo *min_df*, z matice odstraníme
- *max_terms* - maximální počet termů, které si v aplikaci necháme
- *keep_less_freq* - udává, zda si při výběru *max_terms* termů nechat ty nejméně či nejvíce často zastoupené v dokumentech

[1] nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

Zkoumání vlivu změny jednotlivých parametrů na výsledek LSI se dále podrobněji věnujeme v experimentální části. V programu vždy nastavujeme *min_df* alespoň na 1, abychom odstranili termy nacházející se pouze v 1 dokumentu, které nám do LSI nepřidávají žádné užitečné informace. (pravda?)

Z této zredukované matice poté již spočteme matici M_w. Pro výpočet vah jednotlivých termů používáme metodiku *tf-idf*. Váhu termu t_i v dokumentu d_j spočítáme podle vzorce:

$$w_{ij} = tf_{ij} \cdot idf_{ij} \tag{2.1}$$

kde tf_{ij} reprezentuje normalizovanou četnost termu t_i v dokumentu d_j a spočítá se podle vzorce[2] :

$$tf_{ij} = \frac{f_{ij}}{nt_j} \tag{2.2}$$

kde f_{ij} je četnost výskytu termu t_i v dokumentu d_j, kterou normalizujeme číslem nt_j vyjadřujícím celkový počet termů v dokumentu d_j. V přednášových slidech je použita normalizace jiná, tf_{ij} se tam počítá podle vzorce:

$$tf_{ij} = \frac{f_{ij}}{max_i\{f_{ij}\}} \tag{2.3}$$

kde $max_i\{f_{ij}\}$ vrací nejvyšší četnost termu t_i přes celou kolekci dokumentů. Tento způsob normalizace nám vrací spíše horší výsledky, jejich porovnáním se zabýváme v experimentální části. idf_{ij} reprezentuje převrácenou četnost t_i ve všech dokumentech a spočítá se podle vzorce:

$$idf_{ij} = \log_2(\frac{n}{df_i}) \tag{2.4}$$

kde n je celkový počet dokumentů a df_i reprezentuje celkový počet dokumentů obsahujících term t_i.

2.3 Implementace LSI

Jakmile máme vytvořenou matici vah termů M_w, můžeme přistoupit k samotné implementaci LSI. Princip LSI spočívá v tom, že s pomocí *singulárního rozkladu (SVD)* seskupíme tematicky podobné články do jednotlivých k konceptů. Vlivem počtu konceptů na kvalitu výsledků se dále zabýváme v experimentální sekci.

Singulární rozklad nám matici M_w rozloží následovně:

$$M_w = U \cdot S \cdot V^T \tag{2.5}$$

kde řádky matice U jsou obrazy řádků matice M_w, sloupce matice V jsou obrazy sloupců matice M_w a matice S obsahuje na diagonále *singular values (absolutní hodnoty vlastních čísel)* matice M_w v sestupném pořadí. Z těchto matic získáme *concept-by-document* matici M_{cd} jako:

[2] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

$$M_{cd} = S[k,k] \cdot V^T[k,:] \tag{2.6}$$

kde $S[k,k]$ značí prvních k řádků a sloupců matice S a $V^T[k,:]$ značí prvních k řádků matice V^T, kde k je počet konceptů. Nenásobíme tedy celou maticí M_{cd}, ale pouze její část podle počtu konceptů.

Matici projekce dotazu do prostoru konceptů M_q pak získáme jako:

$$M_q = U^T[k,:] \tag{2.7}$$

kde $U^T[k,:]$ značí prvních k řádků matice U^T.

2.4 Vyhodnocení dotazu

Při dotazu na nejpodobnější dokumenty k dokumentu d_j převedeme dotaz do prostoru konceptů na vektor V_c pomocí vzorce:

$$V_c = M_q \cdot M_{w_{:,j}} \tag{2.8}$$

kde $M_{w_{:,j}}$ značí j-tý sloupec matice M_w.

Vektor V_c poté pomocí *kosinové podobnosti* porovnáme se sloupcovými vektory matice M_{cd}. Indexy nejpodobnějších sloupcových vektorů matice M_{cd} pak vrátíme jako indexy nejpodobnějších dokumentů k dokumentu dotazu d_j.

3 Implementace

Celý projekt jsme programovali v jazyce *Python*. Práci nám velmi usnadnila jeho knihovna *NLTK*[3] nabízející rozsáhlou funkcionalitu pro práci s přirozeným jazykem. Využili jsme například *WordNetLemmatizer* pro lemmatizaci či *SnowballStemmer* pro stemming. Dále jsme v programu hojně využívali Python knihovny *pandas*[4] a *numpy*[5].

Ukládání dat v projektu řešíme přes CSV soubory, ke kterým přistupujeme přes pandas funkce. V jednom souboru máme uložený dataset nad kterým provádíme LSI. V dalších souborech pak máme uložené matice M_w, M_{cd} a M_q, abychom je mohli cachovat a přepočítavat jen při změně LSI parametrů, které máme uložené v souboru *server/lsa_config.json*.

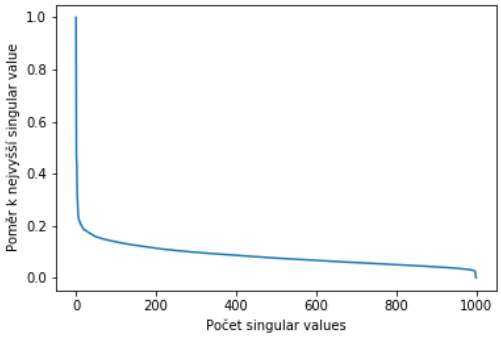
Procházení článků vizualizujeme v prohlížeči pomocí *Flask*[6] web serveru. Jako dataset v našem projektu používáme anglicky psané novinové články stažené z *kaggle.com*[7]. Dataset nepoužíváme celý, vybrali jsme z něj pouze 961 článků.

[3] https://www.nltk.org/
[4] https://pandas.pydata.org/
[5] https://numpy.org/
[6] https://flask.palletsprojects.com/en/1.1.x/
[7] https://www.kaggle.com/snapcrack/all-the-news

4 Příklad výstupu

5 Experimentální sekce

5.1 Určení optimálního počtu konceptů



Obrázek 5.1: Důležitost konceptů

Vezmeme si singular values, které nám vrátil singulární rozklad v rovnici 2.5. Vizualizujeme-li si, jak klesá poměr nejvyšší singular value vůči zbylým singular values, vidíme z toho také, jak klesá důležitost konceptů v datasetu. Počet konceptů k v naší aplikaci tedy podle tohoto grafu určíme jako 400.

5.2 Porovnání vlivu LSI na kvalitu výsledků vyhledávání s ohledem na výskyt synonym a homonym

Pro zjištění kvality výsledků vyhledávání s ohledem na výskyt synonym a homonym jsme si vytvořili testovací dataset 10 článků z anglické Wikipedie... TO BE CONTINUED!!!

6 Diskuze

7 Závěr