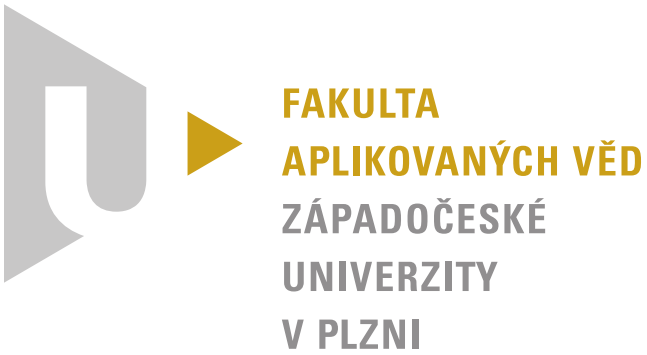
Západočeská Univerzita v Plzni
Fakulta Aplikovaných Věd



Vektorový model pro vyhledávání informací

Filip Jašek

Předmět: SZPJ
Vyučující: Doc. Ing. Ircing Pavel, Ph.D.

Datum: 7. dubna 2024

Obsah

1 Zadání **2**

2 Úvod **7**

3 Dokumentace skriptu **7**
3.1 Předzpracování 7
3.2 Vektorový model 7

4 Závěr **8**

1 Zadání

Semestrální práce: Vektorový model pro vyhledávání informací

Pavel Ircing

ircing@kky.zcu.cz

Základní zadání – I.

Vytvořte v jazyce Python program, který implementuje vektorový model s tf-idf vahami pro vyhledávání informací.

Systém bude pracovat s testovací kolekcí, kterou je k dispozici na v Google Classroom (SZPJ_SP1_collection.zip)

Tato kolekce obsahuje:

- 3204 dokumentů v adresáři **`documents`**
  - jde o "hlavičky" článků z časopisu Communications of the ACM ze 70. let – časopis se v té době zabýval hlavně různými počítačovými algoritmy
  - dokumenty obsahují název článku, často jména autorů a někdy i abstrakt a klíčová slova – rozhodně se ale nelze spolehnout na úplně pevnou strukturu
- 30 „vývojových" témat pro vyhledávání v souboru *query_devel.xml*
  - začátek každého tématu je uvozen tagem `<DOC>`, konec tagem `</DOC>`. Identifikátor tématu je mezi tagy `<DOCNO>` a `</DOCNO>`
- soubor s posouzením relevance (relevance judgments) **`cacm_devel.rel`** obsahující výčet dokumentů, které jsou pro dané téma relevantní – např. řádek:

```
4 Q0 CACM-1749 1
```

značí, že dokument `CACM-1749` je relevantní pro téma 4 – druhý a čtvrtý sloupec můžete bez obav ignorovat

Základní zadání – II.

Výstup programu bude soubor, ve kterém je pro každý zpracovávaný dotaz vygenerován seznam 100 dokumentů, které jsou dle algoritmu nejpodobnější danému dotazu (tj. nejrelevantnější). Seznam je seřazen sestupně dle skóre podobnosti – např:

```
1       CACM-1938       0.24284285661
1       CACM-2319       0.230932347805
1       CACM-1657       0.227669418393
1       CACM-2371       0.19772801487
…
1       CACM-1647       0.0912735727289
1       CACM-0866       0.0905080935677
2       CACM-3078       0.0930908542064
2       CACM-2434       0.084884674854
```

…

- čili formát „`ID` tématu `<tabelátor> ID dokumentu <tabelátor> sk`óre podobnosti"

(vzorový soubor **`vzor_vystupu_vyhledavaciho_programu.txt`**)

Máte k dispozici také skript **`compute_score.py`**, který na základě porovnání tohoto výstupu se souborem **`cacm_devel.rel`** automaticky vyhodnotí střední průměrnou přesnost (Mean Average Precision – *MAP*)

Detaily implementace

- návrh metod pro předzpracování dokumentu a dotazu je zcela na vás – použijte, co uznáte za vhodné

- konkrétní varianta implementace vektorového modelu je také volitelná a to jak z hlediska toho, jaké knihovny Pythonu použijete, tak i konkrétních formulí pro výpočet *tf* a *idf*

- soubor dotazů, který máte k dispozici, slouží pro vyhodnocení úspěšnosti jednotlivých postupů a výběru těch nejvhodnějších. Aby se však odhalilo případné přílišné „naladění“ na konkrétní soubor dotazů, reálná úspěšnost se většinou zjišťuje pomocí další sady dotazů, která není při vývoji systému k dispozici – tzv. **evaluačních datech**. Tak to uděláme i v našem případě – úspěšnost na evaluačních datech vyhodnotím s použitím vašich systémů sám.
  - prosím berte na vědomí, že čísla témat se budou lišit od *development* dat – pozor na správné parsování souboru s dotazy !

Odevzdání semestrální práce

Jako výsledek své práce odevzdejte:

- program spolu s návodem, jak jej použít pro vyhledávání – kolekci dokumentů předpokládejte neměnnou, měnit se mohou pouze zadávané dotazy. I u těch však bude samozřejmě zachován formát souboru.
- krátkou dokumentaci, která kromě výše zmíněného návodu bude obsahovat i popis použitých metod předzpracování dat, vyzkoušených variant vektorového modelu, případně dalších zajímavostí.
- program bude považován za úspěšně fungující, pokud jeho výstup pro **vývojová** (devel) data – tj. ta, která máte k dispozici – dosáhne hodnoty MAP alespoň **0.3**

Termín odevzdání a bodování:

- termín odevzdání: **7.4.2024**
- maximální možný počet bodů: **10 + „bonus za umístění“:**
  - všechny v termínu odevzdané programy budou seřazeny podle hodnoty MAP dosažené na **evaluačních** datech (tj. těch, která nemáte k dispozici)
  - autor nejlepšího systému získá jako bonus navíc **9** bodů, autor druhého nejlepšího **8.5**, atd. Čili v případě, že práci *v termínu* odevzdají všichni, kdo předmět prokazatelně studují, získá autor systému na posledním místě 0.5 bonusového bodu.

- penalizace za pozdní odevzdání či odevzdání nesprávně fungující práce: **2** body

2 Úvod

Cílem této semestrální práce bylo implementovat v jazyce Python vektorový model s tf-idf vahami pro vyhledávání informací. Skript byl napsán tak aby zpracovával dotazy zadané v xml souboru a nikoliv interakvivní formou. Čistě pro testování přístupu a jeho benchmark.

Zdrojový kód je dostupný i s návodem k použití na GitHub repository (https://github.com/Fiiila/SZPJ.git).

3 Dokumentace skriptu

3.1 Předzpracování

Při prvním kroku implementace bylo potřeba správně naparsovat zdrojové soubory html a identifikovat tak jen relevantní obsah. Jako první byly vymazány řádky obsahující tagy například `<html>,</html>,<pre>,</pre>`. Dále pak byly identifikovány pro vyhledávání nerelevantní řádky obsahující pouze číselné hodnoty a také řádek s informací pravděpodobně o přístupu ke článku ve tvaru například *CA581203 JB March 22, 1978 8:28 PM*.

Po vynechání zmíněných řádků byly zbývající zbaveny prázdných charakterů na okrajích funkcí `strip()` a spojeny do skupin dle rozdělení prázdnými řádky. Tato funkcionalita je sice nadbytečná ale v případě vyhledávání například ve specifickém roce nebo v jedné části textu by mohla zjednodušit a zrychlit vyhledávání zejména při použití většího zdrojového datasetu za pomoci filtrace nebo vytváření subsetů pro vektorizaci.

Při samotném vytvoření datasetu byla navíc odstraněna základní interpunkce, aby se eliminovala souvislost hledaných výrazů a interpunkce. V tomto duchu byla později implementovaná i metoda stemming z knihovny NLTK[1], která v nejrůznějších tvarech slov nalezne kořen slova čímž je pak původní slovo v dokumentu nahrazeno. Jednotlivá slova byla rozdělena tokenizací pomocí knihovnou NLTK[2] Stejný postup se použije i při načítání jednotlivých dotazů pro vyhledávání. Celkové MAP bylo tak zlepšeno o $\approx 10.62\%$.

3.2 Vektorový model

Zvolený vektorový model byl z knihovny skicit-learn[3] prezentovaný na přednáškách pro jeho jednoduchost. Při experimentaci s parametry bylo dosaženo nejlepších výsledků s následujícím nastavením.

```
# Specify vectorizer object and transform data
tfidf = TfidfVectorizer(norm=None, use_idf=True, smooth_idf=True,
                                                  sublinear_tf=True,
                                                  stop_words='english'
                                                  )
```

Normalizace výstupních řádků nepřinesla žádné zlepšení, naopak `use_idf` pomohlo z podstaty věci aktivovat kdy při deaktivaci je $idf(t) = 1$. Vyhlazení vážení idf pomocí `smooth_idf` také zlepšilo MAP podobně jako použití logaritmického škálování `tf` jako $1 + \log(tf)$. Výhoda byla také při použití parametru `stop_words='english'` která při tokenizaci eliminuje slova které nemají velký vliv na samotnou informaci obsaženou v textu jako je 'the', 'and', 'him'.

[1] https://www.nltk.org/howto/stem.html
[2] https://www.nltk.org/api/nltk.tokenize.word_tokenize.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

4 Závěr

V této semestrální práci byly aplikovány znalosti z přednášek předmětu SZPJ, zejména pro rozdělení informace na tokeny a samotné vektorové vyhledávání. Testováním a použitím všech zmíněných argumentů a metod při parsování a výběru podstatných informací bylo dosaženo na testovacím datasetu skóre $MAP \approx 0.3625$.