Semestrální práce z předmětu KIV/IR

Indexer

Zdeněk Valeš

26.5. 2020

1 Zadání

V jazyce Java vytvořte indexer, který bude schopný indexovat zadané dokumenty a následně nad nimi provádět vyhledávání. K realizaci práce použijte připravená rozhraní.

2 Analýza

Aplikace se skládá ze dvou částí. První část tvoří jádro, které indexuje dokumenty a vyhledává na základě dotazu, druhou část tvoří jednoduché grafické rozhraní pro práci s indexerem.

Jádro aplikace se skládá z textového preprocesoru, indexu a vyhledávače. Preprocesor je použit k převedení textu (obsah dokumentu nebo vyhledávací dotaz) na tokeny a ty pak do jejich základní formy.

2.1 Preprocessing

V průběhu pre-processingu textu dojde k jeho rozdělení na tokeny, vyřazení stop-slov a konečně k převodu do základní formy. Před rozdělením na tokeny jsou všechna písmena v textu převedena na malá a text je zbaven diakritiky.

Stop-slova jsou slova, jež typicky patří k nejběžnějším v jazyce, ale samotnému textu nedávají přílišný význam a jejich indexace je tedy zbytečná. Seznam stop-slov je pro každý jazyk jiný a neexistuje jeden univerzální. Protože jsem v průběhu cvičení stahoval anglický text, pre-processor primárně pracuje s anglickými stop- slovy[1]. Aby bylo možné provést TREC evaluaci, je k dispozici také soubor s českými stop slovy. Algoritmus odstranění stop-slov spočívá v iterování přes text rozdělený na tokeny a v případě, že daný token odpovídá některému ze stop-slov, je z výsledného seznamu tokenů odstraněn.

Stop-slova jsou uložena v 'nezákladní' formě (nejsou tedy zpracována stemmerem, nebo lemmatizérem) a proto se jejich vyhledání a odstranění z textu provádí ještě před jeho převedením na základní tvar.

Token lze na základní tvar převést stematizací, nebo lemmatizací. Lemmatizace je o něco složitější a i když vrací gramaticky správný základní tvar (stemmer vrací kmen), není tento potřeba a proto jsem se rozhodl použít stemmer, místo lemmatizéru.

2.2 Index

Datová struktura pro ukládání zpracovaných dokumentů se nazývá index. Nejjednodušší formou indexu je tzv. incidenční matice o rozměrech $|D| \times |T|$, kde D je množina uložených dokumentů a T množina termů. V incidenční matici je pak každé dvojici dokument-term přiřazena hodnota 1 nebo 0, pokud se daný term v dokumentu nachází nebo ne.

Tento způsob ukládání je značně neekonomický, protože většina hodnot v matici je 0 a ukládat gigabyty nul není žádoucí. Řešením je tzv. invertovaný index, což je struktura udržující seznam termů a pro každý term, seznam dokumentů, ve kterých se daný term

[1] https://gist.github.com/sebleier/554280

vyskytuje – tzv. postings list. Výhodou invertovaného indexu oproti incidenční matici je mimo jiné také možnost uložit počet výskytů termu v dokumentu, čehož jsem v práci využil při počítání TF-IDF vah.

2.2.1 TF-IDF váhy

Na dokument je, v rámci, indexu nahlíženo jako na množinu výskytů termů a konkrétní pozice termu v dokumentu není zohledněna. Tento přístup se nazývá *bag of words model* a je založen na předpokladu, že dva dokumenty reprezentované podobnými množinami termů si budou také podobné.

Jedním z klíčových ukazatelů podobnosti je počet výskytů $\mathrm{TF}_{t,d}$, termu t v dokumentu d. Ukazatel TF nicméně nezohledňuje důležitost termů a na všechny nahlíží jako na stejně rovné. Řešením je přidání váhy *inverse document frequency*, definované jako $\mathrm{IDF}_t = \log \frac{N}{df_t}$. Tato přiřazuje vzácnějším termům vyšší váhu a naopak běžným termům vyskytujícím se ve velkém množství dokumentů váhu nižší. Výsledná TF-IDF váha termu t v dokumentu d je definovaná jako $\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$.

Pro zefektivnění vyhledávání jsou v indexu také uložené předpočítané IDF váhy termů a TF-IDF váhy termů v dokumentech. Původní verze programu tyto počítala během vyhledávání, což vedlo na velmi dlouhou dobu zpracování dotazu při TREC vyhodnocení (jednotky až desítky minut).

2.3 Vyhledávání v indexu

Vyhledávání v indexu pracuje ve dvou režimech: boolean retrieval a ranked retrieval. Boolean retrieval pracuje s booleovskými dotazy a vrací dokumenty, které přesně odpovídají dotazu. Oproti tomu ranked retrieval pracuje s výše zmíněnými TF-IDF vahami, na základě kterých počítá podobnost dotazu s dokumentem a vrací dokumenty s nejvyšší podobností.

2.3.1 Boolean retrieval

- intersection jednotlivých množin pro operátory AND, OR, NOT

2.3.2 Ranked retrieval

- kosínová podobnost

Optimalizace: - předpočet IDF pro termy - předpočet TF-IDF pro Postings - relativní cos similarity

3 Popis implementace

Java aplikace, ne webová

3.1 Jádro

lucene na parsování ukládání do souboru srkze java serializaci

3.2 Uživatelské rozhraní

Swing Možnost výběru mezi boolean/ranked retrieval progress bary pro jednotlivé akce (indexování/vyhledání) dokumenty lze indexovat po jednom dokumenty lze indexovat hromadně z json souboru (podporovaný formát viz moje data z prvních cvičení)

4 Závěr