Západočeská univerzita v Plzni

Fakulta aplikovaných věd

KIV/NET

Vyhledávací engine

Vojtěch Bartička, barticka@students.zcu.cz, A21N0038P

30. května 2022

Obsah

1 Zadání

Cílem semestrální práce je naučit se implementovat komplexní systém s využitím hotových knihoven pro preprocessing. Vedlejším produktem bude hlubší porozumění indexerům, vyhledávacím systémům a přednáškám.

1.1 Minimální požadavky

- Tokenizace, preprocessing (stopwords remover, stemmer/lemmatizer),
- vytvoření in-memory invertovaného indexu,
- tf-idf model, kosinová podobnost,
- vyhledávání pomocí dotazu vrací top k výsledků seřazených dle relevance (použití vektorového modelu),
- vyhledávání s pomocí logických operátorů AND, OR, NOT (booleovský model), podpora závorek pro vynucení priority operátorů,
- podrobná dokumentace (programátorská i uživatelská),
- zadávání dotazů z CLI nebo GUI, možnost vybrat index a model při zadávání dotazů,
- výsledky vyhledávání obsahují i celkový počet dokumentů, které odpovídají zadanému dotazu.

1.2 Nadstandardní funkčnost

- File-based index,
- pozdější doindexování dat,
- ošetření např. HTML tagů,
- detekce jazyka dotazu a indexovaných dokumentů,
- vylepšení vyhledávání,
- vyhledávání frází (i stop slova),
- vyhledávání v okolí slova,
- více scoring modelů,

- indexování webového obsahu,
- další předzpracování normalizace,
- GUI/webové rozhraní,
- napovídání keywords,
- podpora více polí pro dokument,
- CRUD indexovaných dokumentů,
- zvýraznění hledaného textu v náhledu výsledků,
- dokumentace psaná v TEXu,
- vlastní implementace parsování dotazů bez použití externí knihovny,
- implementace dalšího modelu (použití sémantických prostorů, doc2vec, Transformers (BERT)).

2 Analýza

2.1 Preprocessing

V rámci povinných požadavků je nutné implementovat nějakou formu předzpracování. Je potřeba vzít v dotaz časovou náročnost implementace nadstandardních funkcí. Jako standard lze brát tokenizaci, použití stop slov a nějakou formu normalizace slov.

Česká stop slova jsou volně dostupná na internetu a nebude problém je získat. Zároveň odstraňování stop slov je triviální záležitost. Je potřeba dát pozor na pořadí operací, resp. pokud budou stop slova odstraňována po normalizaci, je nutné aby byla stop slova také normalizována.

Normalizace slov se skládá z několika kroků - základního předzpracování jako je převod na malá písmena nebo odstranění diakritiky, a pokročilého předzpracování do kterého spadá stemování a lemmatizace. Převod na malá písmena a odstranění diakritiky jsou relativně nenáročné úlohy, které je možné v krátkém čase implementovat, ale vlastní implementace stemování nebo lemmatizace je časově náročná. S ohledem na to, že není zakázáno použití knihovních implementací, se zdá jako rozumné řešení použití některé z dostupných implementací.

Při rozhodování zda použít stemování nebo lemmatizaci je nutné brát ohled především na to, pro co je dostupná knihovní implementace pro .NET.

Dalším faktorem je časová náročnost - stemování je relativně časově nenáročná operace, zatímco lemmatizace může v závislosti na použitém lemmatizéru trvat poměrně dlouhou dobu. Navíc lemmatizace bere v potaz i kontext zpracovávaného slova - není tedy možné efektivně lemmatizovat samostatná slova bez kontextu, což může způsobit komplikace ve fázi implementace.

2.2 Indexování

Pro indexování je povinná implementace invertovaného indexu. Na jeho konstrukci nic složitého není, jedná se o dobře zdokumentovaný proces, který je navíc srozumitelně popsán v přednáškách.

2.3 Vyhledávání

2.3.1 Vektorový model

Je povinné implementovat TF-IDF vektorový model. Na jeho implementaci není nic složitého, protože většina implementace je úzce spojena s invertovaným indexem. Je potřeba spočítat dokumentovou frekvenci *df* pro každý term a frekvenci termu *tf* pro každou dvojici dokument - term. To je přirozeně možné udělat v rámci konstrukce invertovaného indexu z postingů.

2.3.2 Booleovský model

Booleovské vyhledávání je komplikovanější než vektorové vyhledávání. Je potřeba zpracovat dotaz, a poté je potřeba získat relevantní dokumenty. Získání relevantních dokumentů je popsáno v přednáškách a nemělo by být příliš problematické, ale náročnost implementace zpracování dotazů bude záviset na tom, jestli bude k dispozici knihovní implementace pro .NET. Pokud bude nutné zpracování implementovat od nuly, je potřeba počítat se zdržením. Jako možnost se jeví použití Lucene.Net pro zpracování dotazů, ale knihovna není primárně určena pro zpracovávání booleovských dotazů, takže těžko předvídat nakolik bude použitelná.

2.4 Architektura

Protože v rámci předmětu KIV/NET je nutné implementovat GUI, jako vhodná architektura se jeví standardní MVC. Indexování a předzpracování bude implementováno v modelu, rozhraní ve view a kontoler bude zprostředkovávat komunikaci mezi view a modelem.

3 Implementace

3.1 Předzpracování

Předzpracování je implementováno v projektu Core ve jmenném prostoru Preprocessing. Skládá se z několika částí:

- IAnalyzer - rozhraní, které zajišťuje celkové předzpracování dokumentů (implementováno třídou Analyzer),
- ITokenizer - rozhraní, které zajišťuje rozdělení dokumentu na slova (implementováno třídou Tokenizer),
- IStopwords - rozhraní, které zajišťuje načtení stop slov, jejich předzpracování a odstraňování z dokumentů (implementováno třídou Stopwords),
- IStemmer - rozhraní, které zajišťuje stemování slov (implementováno třídou Stemmer),
- Accents - třída se statickými metodami, která zajišťuje odstranění diakritiky,
- AnalyzerConfig - třída, která obsahuje konfiguraci analyzátoru.

Idea je taková, že se pro index bude při jeho vytváření specifikovat analyzátor, pomocí kterého index bude provádět předzpracování dokumentů před jejich indexováním. Jelikož různé indexy mohou vyžadovat různé způsoby předzpracování, je princip fungování takový, že analyzátoru se předá nastavení AnalyzerConfig. Toto nastavení je pak propagováno do IStopwords, aby byla zajištěna konzistence předzpracování mezi analyzátorem a stop slovy.

Všechny podstatné funkční celky jsou pro jistotu definovány jako rozhraní nebo abstraktní třídy, ačkoliv není v plánu v rámci práce vytvářet více implementací těchto funkčních celků.

Pro tokenizaci byly použity regulární výrazy implementované v rámci cvičení z KIV/IR. Protože zanechávaly nežádoucí artefakty, tak bylo provedeno další zpracování tokenů vytvořených regulárním výrazem.

V rámci normalizace tokenů bylo možné provést buďto lemmatizaci nebo stemování. Bylo zvoleno stemování, protože existuje knihovní implementace Snowball stemmeru pro .NET, která podporuje i češtinu. Navíc je stemování podstatně rychlejší než lemmatizace.

Bylo také implementováno odstraňování diakritiky. To je založeno na kódu ze StackOverflow (odkaz v kódu), který byl podstatně přepracován a zefektivněn. Podporuje kromě české diakritiky i např. německou nebo francouzskou.

Také je možné odstranit stop slova. Třída Stopwords obsahuje výchozí stop slova (ze stopwords-iso, odkaz v kódu), která je možné použít (metoda UseDefaults). Stop slova jsou (jak již bylo zmíněno výše) předzpracovány stejným způsobem jako tokeny dokumentů. Aplikace při vytváření indexu umožňuje specifikovat vlastní sadu stop slov.

3.2 Indexování

Indexování je implementováno v projektu Core ve jmenném prostoru Indexing. Ten se skládá pouze ze tří částí:

- AIndex - abstraktní předek pro obecné indexy,
- InvertedIndex - invertovaný index, dědící od AIndex,
- IndexConfig - konfigurace, která není používaná.

V rámci indexování dokumentů se nejprve zpracují všechny dokumenty a vytvoří se seznam postingů. V prvotní implementaci byl použit pouze invertovaný index, ale ukázalo se, že vytváření vektorů pro dokumenty je s použitím invertovaného indexu poměrně pomalé. Podle profileru procházení každého termu a zjišťování, jestli se v něm dokument vyskytuje, bylo jednou z časově nejnáročnějších operací. Předpočítat vektory dokumentů je pak s ohledem na velikost slovníku nereálné kvůli paměťové náročnosti. Kvůli tomu se kromě invertovaného indexu, který mapuje term na dokument, vytváří "neinvertovaný" index, který mapuje dokument na term. Tímto způsobem máme okamžitý přístup k termům, které dokument obsahuje, a podstatně se zrychlí vytváření vektorů dokumentů. Cenou za toto zrychlení je o trochu vyšší paměťová náročnost.

V invertovaném indexu se ukládá pro každý term seznam dokumentů a jejich term-frequency *tf*. Také se pro každý term ukládá document-frequency *df*. V rámci optimalizací se co nejméně používají slovníky - pouze při mapování termu na jeho ID se používá slovník, jinak se používají pole.

Slovník se skládá ze všech unikátních termů, které se v indexovaných dokumentech vyskytují. Neprovádí se žádná redukce velikosti slovníku. Index nepodporuje dodatečné indexování ani různá "pole".

3.3 Vyhledávání

Vyhledávání je definováno jako součást rozhraní IIndex a je s indexem úzce spjato. Je implementováno vektorové a booleovské vyhledávání.

3.3.1 Booleovské vyhledávání

Booleovské vyhledávání podporuje vynucení priority operátorů závorkami a operátory AND, OR a NOT. NOT má nejvyšší prioritu, poté AND a nejnižší prioritu má OR. Jako výchozí operátor se používá OR.

Zpracování dotazů využívá vlastní implementaci, která se nachází ve třídě Common.Utils.BooleanQueryParser. Nejprve se provede tokenizace dotazu do termů, závorek a operátorů. Poté jsou vloženy výchozí operátory. Tokeny v infixové notaci (infixovém pořadí v seznamu) jsou převedeny do postfixové notace, ze které je poté zkonstruován binární strom, který je pak snadné vyhodnotit. Operátory AND a OR mají dva operandy (tedy jejich uzel ve stromě má dva potomky) a operátor NOT má pouze jednoho potomka (vždy levý potomek uzlu).

S binárním stromem reprezentujícím dotaz je už poměrně jednoduché dotaz provést. Postupuje se rekurzivně od kořene směrem k listům (termům) a každý uzel se zpracovává. Pokud je uzel operátor, jsou nejprve vyhodnoceni jeho potomci. Pokud je uzel term, jsou vráceny všechny dokumenty, které term obsahují (viz. invertovaný index). Pokud term neexistuje, nejsou vráceny žádné dokumenty.

Pro negaci (operátor NOT) je vrácen doplněk všech dokumentů, získaných z jeho levého potomka. Pro operátor AND je vrácen průnik dokumentů z levého a pravého potomka. Pro operátor OR je vráceno sjednocení dokumentů z levého a pravého potomka.

3.3.2 Vektorové vyhledávání

Vektorové vyhledávání je definováno jako součást rozhraní IIndex, a tedy je implementováno ve třídě InvertedIndex. Dle zadání se používají TF-IDF vektory a kosinová podobnost.

V rámci rychlostních optimalizací bylo implementováno předfiltrování dokumentů booleovským dotazem. To má dvě fáze - nejprve se mezi jednotlivé tokeny dotazu vloží operátory AND a provede se booleovské vyhledávání. Pokud tento dotaz vrátí méně než 5 dokumentů, je vyhledávání opakováno s operátory OR. Důvodem je to, že operátory AND jsou příliš omezující především u dlouhých dotazů, kde takový booleovský dotaz nemusí vrátit žádné relevantní dokumenty. Proto je jako záložní řešení použit booleovský dotaz s operátory OR mezi jednotlivými tokeny dotazu. Ten pro dlouhé dotazy získá velké množství dokumentů, pro které je pak potřeba zkonstruovat vektory a spočítat jejich podobnosti vůči dotazu. Tento postup ale dává smysl, protože pro krátké dotazy výrazně zrychlí proces vykonávání dotazu. Pro dlouhé dotazy je to akceptovatelný kompromis.

Při konstrukci vektorů dokumentů se nepoužívá invertovaný index, ale "ne-invertovaný" index (viz. Indexování), který mapuje dokumenty na termy. Tím se podstatně zrychlila konstrukce vektorů dokumentů, protože není potřeba procházet každý term ve slovníku a zjišťovat, zda se v daném dokumentu vyskytuje. Byla také zvažována možnost předpočítat vektory dokumentů, ale ukázalo se, že paměťová náročnost by pak byla pro větší počty dokumentů neúnosná. Byly také prováděny experimenty s vektory, které podporují SIMD instrukce (balíček System.Numerics), ale ten je orientovaný spíše na počítačovou grafiku a nepodporuje vektory libovolné délky.

Jako nejproblematičtější místo co se týče výkonu se ukázalo počítání skalárního součinu vektorů při počítání kosinové podobnosti. To by bylo možné obejít například použitím lokálně senzitivního hashování (LSH) typu Minhash nebo Simhash nebo nějakou aproximací kosinové podobnosti.

Jako váhové schéma pro konstrukci vektoru dokumentu a vektoru dotazu je použito LTN.LTN. Term-frequency *tf* je logaritmována, je použita inverzní document-frequency *df* a není prováděna normalizace (tedy stejný postup, který byl použit ve cvičení TF-IDF v rámci KIV/IR). Pro dotaz i dokument je použito stejné schéma.

3.4 Testy

Byly implementovány jednotkové testy, ověřující funkcionalitu základních komponent řešení. Jako referenční výsledky byly použity manuálně vypočítané příklady z TF-IDF cvičení. Je testován tokenizér, odstranění diakritiky, odstraňování stop slov, booleovské vyhledávání a vektorové vyhledávání. Testy vektorového vyhledávání procházejí, což ukazuje správnost implementace vytváření TF-IDF vektorů a počítání jejich podobnosti.

Testy jsou projektu Test, třídě ModelTests.

3.5 GUI

Uživatelské rozhraní je implementováno pomocí WPF. Kvůli tomu je aplikace omezena na platformu Windows. Aplikace je implementována podle vzoru Model-View-Controller. Jednotlivé části jsou ve stejnojmenných projektech.

3.6 Historie dotazů

Aplikace využívá SQLite databázi a Entity Framework pro ukládání dotazů za běhu aplikace. Databáze je vyprázdněna při každém spuštění.

Při indexování dokumentů si index uloží uloží pro každý dokument jeho id. Id je pak vráceno jako součást seznamu relevantních dokumentů při vyhod-

nocení dotazu. Controller si pak uloží seznam id dokumentů a jejich pořadí do databáze spolu s časem provedení dotazu, textem dotazu, typem dotazu a indexem, nad kterým byl dotaz proveden.

Poté je možné pro každý index zobrazit historii dotazů. Při kliknutí na konkrétní dotaz z historie se v seznamu výsledků zobrazí relevantní dokumenty v původním pořadí, nastaví se text dotazu, typ dotazu a celkový počet relevantních dokumentů.

Kvůli nastavení textu dotazu a typu dotazu musí být Controlleru předány callbacky, přes které je možné tyto hodnoty ve View nastavit.

4 Uživatelská dokumentace

Aplikaci je možné spustit pouze na OS Windows kvůli frameworku WPF. Zároveň je potřeba mít nainstalovaný .NET 6 a stáhnout všechny NuGet balíčky.

4.1 Indexování

Aplikace podporuje vytvoření vícero indexů. Při vytváření indexu je možné nakonfigurovat předzpracování (lowercasing, stemming a odstranění diakritiky). Zároveň je možné specifikovat soubor se stop slovy nebo stop slova nepoužívat. Při vytváření indexu je nutné specifikovat složku, obsahující dokumenty k indexování. Složka musí obsahovat textové soubory a každý textový soubor musí odpovídat jednomu dokumentu (aplikace načte celý text souboru jako dokument). Data získaná v prvním cvičení jsou v předzpracované podobě dostupná na (viz. sekce 4.6).

4.2 Dotazování

Aplikace umožňuje provádět booleovské a vektorové dotazy nad libovolným vytvořeným indexem. Dokumenty, které jsou pro daný dotaz relevantní, se zobrazí v pravé části hlavního okna. Dvojitým kliknutím na dokument se dokument zobrazí celý. Pro booleovské dotazy jsou dokumenty seřazeny podle jejich ID. Pro vektorové dotazy jsou seřazeny podle kosinové podobnosti sestupně. Pro každý dotaz se zobrazí všechny relevantní dokumenty. Celkový počet relevantních dokumentů je zobrazen pod tlačítkem "Search".

4.3 Historie dotazů

Historie dotazů je v levém panelu. Dotazy jsou seřazeny od nejnovějšího. Při změně indexu se změní historie dotazů. Při kliknutí na nějaký z dotazů v historii dotazů se načte text dotazu, typ dotazu, celkový počet výsledků a výsledky dotazu v původním pořadí.

4.4 TREC

Je možné provést evaluaci na TREC datasetu, kterou ale není možné konfigurovat (používá stemming, lowercasing, odstranění diakritiky a výchozí stop slova). Je potřeba specifikovat složku, která obsahuje předzpracovaný TREC dataset - ten je možné stáhnout na adrese (viz. 4.5). Pro dotazování se používá pouze pole Title. Výstup evaluace je uložen v souboru results.txt ve stejné složce.

4.5 Obsah archivu s daty pro TREC evaluaci (Google Drive)

Archiv je dostupný na adrese viz. poznámka pod čarou[1]. Obsahuje složku trec, která obsahuje složky topics a documents. Stačí archiv rozbalit a při výběru složky s daty pro evaluaci vybrat složku trec.

4.6 Obsah archivu s nacrawlovanými daty (Google Drive)

Archiv je dostupný na adrese viz. poznámka pod čarou [2]. Obsahuje složky json-final, která obsahuje nepředzpracovaná data, a output-final, která obsahuje předzpracovaná data. Při vytváření nového indexu a výběru složky je potřeba vybrat složku output-final.

5 Závěr

Práce splňuje minimální požadavky v rámci předmětu KIV/IR. Povedlo se implementovat vlastní zpracování booleovských dotazů. Zároveň byla implementována historie dotazů nad rámec minimálních požadavků KIV/IR (ale v souladu s požadavky KIV/NET).

[1] `https://drive.google.com/file/d/1pt1xltkEjF9R7Jn0VAAsUAHLadrBTYy2/view?usp=sharing`

[2] `https://drive.google.com/file/d/1L15LbTGcChte-9UtB0IFkx4YL9mccL6H/view?usp=sharing`

Povedlo se zásadním způsobem optimalizovat rychlost indexování a vyhledávání. Je zde potenciál pro další optimalizace - chytré použití System.Numerics.Vector, použití Simhashe atd.