Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Semestrální práce z KIV/IR

Systému automatické indexace a vyhledávání dokumentů

Jakub Schenk

A21N0069P

schenkj@students.zcu.cz

18. května 2022

Obsah

1 Zadání

1.1 Zadání IR

Implementace vlastního systému automatické indexace a vyhledávání dokumentů.

Minimální nutná funkčnost semestrání práce pro získání 15 bodů/zápočtu:

Tokenizace, Preprocessing (stopwords remover, stemmer/lemmatizer), vytvoření in-memory invertovaného indexu, tf-idf model, cosine similarity, vyhledávání pomocí dotazu vrací top x výsledků seřazených dle relevance (tj. vektorový model - vector space model), vyhledávání s pomocí logických operátorů AND, OR, NOT (booleovský model), podrobná dokumentace (programátorská i uživatelská), podpora závorek pro vynucení priority operátorů.

Semestrální práce musí umožňovat zaindexování poskytnutých dat a dat stažených na cvičení (1. cvičení Crawler) a dat z portálu (složka TREC). Obě sady dat je možné zaindexovat nezávisle na sobě.

Semestrální práce musí umožňovat zadávat dotazy z GUI nebo CLI (command line interface), při zadávání dotazů je možné vybrat index a model vyhledávání (vector space model vs boolean model). Výsledky vyhledávání obsahují i celkový počet dokumentů, které odpovídají zadanému dotazu.

Nadstandardní funkčnost (lze získat až dalších 15 bodů):

- File-based index (1b),
- pozdější doindexování dat (1-2b) - přidání nových dat do existujícího indexu.
- ošetření např. HTML tagů (1b),
- detekce jazyka dotazu a indexovaných dokumentů (1b),
- vylepšení vyhledávání (1-?b),
- vyhledávání frází (i stop slova)(1b),
- vyhledávání v okolí slova (1b),
- více scoring modelů (1b),
- indexování webového obsahu (2-3b) - zadam web, program stahne data a rovnou je zaindexuje do existujícího indexu,
- další předzpracování normalizace (1b),

- GUI/webové rozhraní (2b),
- napovídání keywords (1b),
- podpora více polí pro dokument (např. datum, od do)(1b),
- CRUD indexovaných dokumentů (2b),
- zvýraznění hledaného textu v náhledu výsledků (1b),
- dokumentace psaná v TEXu (1b),
- Vlastní implementace parsování dotazů bez použití externí knihovny (1-?b)
- implementace dalšího modelu (použití sémantických prostorů, doc2vec, Transformers (BERT)) (1-?b),
- atd. (xb)

Konečné posouzení rozsahu práce a bodového hodnocení je plně v kompetenci cvičícího.

2 Analytická část

Práce probíhá nad dokumenty psané v českém jazyce a analytická část nebere v potaz slova anglická. Pro tuto práci je potřeba aby program byl schopný zpracovat a vyhledávat v dokumentech psaných česky a všechny informace níže předpokládají pouze české dokumenty.

2.1 Tokenizace

Povinnou součástí práce je tokenizace obdržených dokumentů a dotazu. Jedná se o rozdělení textu dle nějakých pravidel na tokeny, které jsou dále zpracovávány. Jelikož pracujeme s daty získané pomocí crawlování z reálných webů, jedná se o data, která obsahují přirozený jazyk. Nabízí se tedy tokenizovat dokumenty po slovech, slovních spojeních, nebo větách.

Osobně jsem se rozhodl pro tokenizace po jednotlivých slovech. I když ztrácím informaci o ustálených slovních spojeních, jedná se o cestu nejmenšího odporu s výbornými výsledky. Vyhýbám se tak i možným problémům, které by s indexováním slovních spojení nastaly, jako je předpoklad existence ustáleného slovního spojení, které jím není, nebo rozpoznávání slovních spojení tam, kde nejsou.

2.2 Preprocessing

Preprocessing, nebo také předzpracování, je metoda úpravy vstupu tak, aby umožnila nebo zefektivnila algoritmu následnou práci s daty. V tomto případě se jedná o způsob, jak z obdržených dokumentů odebrat slova, která nepomáhají vyhledávání a upravit slova do jejich základní/základnější podoby pro lepší výsledky indexace.

2.2.1 Stopwords

Prvním krokem v preprocessingu bývá odebrání tzv. stopslov, tedy slov získaných z nějakého seznamu, které nemají dostatečný vliv na obsah dokumentu a jsou tedy pro vyhledávání přebytečná. Takovými slovy jsou v češtině zejména spojky, předložky, zájmena apod. Odebráním stopslov se výrazně sníží počet slov v dokumentu, ale minimálně se zmenší velikost slovníku. Zároveň delší dokumenty s hodně stopslovy nezískávají výhodu při použití těchto slov při vyhledávání (ze kterého jsou také filtrována).

2.2.2 Stemmer/Lemmatizer

Stemming a případně Lemming jsou metody zobecnění tvaru slova. Jedná se o způsob jak vylepšit výsledky vyhledávání zvláště v kolekcích s málo dokumenty.

Stemming z tokenu odebere rozeznávané koncovky čímž zajistí, že slovo bude v základním tvaru případně v prvním pádě, nebo se mu bude podobat. Jelikož se tento postup aplikuje na všechny výskyty slova, nezáleží na tom, jestli je ve výsledné tabulce slovo uloženo špatně, protože výsledný odkaz bude na neporušený dokument.

Lemming má podobný cíl jako stemming, ale ten slova převádí do základního tvaru. Slovesa do infinitivu, ostatní slova do jejich nevyskloňované formy, případně některá slova do synonym, kterými je nahrazuje. Výhodou oproti stemmingu je, že se jedná o metodu s razantnějšími výsledky. Nevýhodou je odebírání někdy důležitých předpon a náročnost implementace. Ke každému slovu existuje několik možností jak jej napsat a často i několik synonym. Implementace pořádného Lemmatizeru je tedy velice zdlouhavá, stejně jako jeho procházení při indexaci.

2.2.3 Invertovaný index

Populární způsob pro ukládání předzpracovaných dat. Jedná se o Hashtabulku, nebo HashSet, kde klíčem je jedinečné slovo ze sady dokumentů a

jeho hodnotou je seznam indexů dokumentů, ve kterých se slovo vyskytuje (případně kolikrát).

Pokud by se jednalo o systém, který by obsahoval velké množství dokumentů nebo dokumentů v různých jazycích, slovník (invertovaný index) by narostl do větších rozměrů a musel by být uložen někde na disku a načítán do paměti po částech. Pro účely této práce nicméně stačí mít invertovaný index uložený v paměti.

2.3 TF-IDF

Zjednodušeně řečeno se jedná o způsob jakým získat vektor dokumentu. Tento vektor je vypočítán pomocí invertovaného indexu nebo taky pomocí slov která se v dokumentu vyskytují. Jedná se tedy o způsob jakým lze popsat dokument tak, aby byla zahrnuta všechna slova, která dokument obsahuje. To je důležité pro následné vyhledávání.

TF-IDF bere v potazu jak všechna jednotlivá slova dokumentu, tak jejich četnost. Při vyhledávání je pak umožněno uživateli vrátit dokument, který je více relevantní k jeho dotazu, protože klíčové slovo obsahuje vícekrát a nejedná se pouze o náhodný výskyt v dokumentu, který se jinak problematikou nezabývá.

2.4 Kosínová podobnost

Součástí TF-IDF modelu je také vyhledávač, který s těmito vektory pracuje. Získaný dotaz převede na vektor stejně jako zpracované dokumenty. Následuje porovnání přes kosínovou podobnost se všemi dokumenty, čímž se zbavíme nepříjemnosti, že dotaz je mnohem kratší než dokumenty. Dokumenty, které jsou dotazu kosínově nejpodobnější jsou označeny za výsledky vyhledávání a případně seřazeny podle podobnosti.

2.5 Vyhledávání pomocí AND, OR, NOT a závorek s vynucením priority operátorů

Důležitou vlastností systému je umožnit uživateli přesněji specifikovat, jaké výsledky by si přál od systému získat. Toho lze docílit zavedením možnosti psát do vyhledávače logické operátory a k nim závorky.

Cílem je vyhledávač implementovat tak, aby použití logický operátorů fungovalo intuitivně a konzistentně. Z toho důvodu je nutné určit priority operátorů. Samozřejmě závorky mají nejvyšší prioritu a jsou vyhodnocovány předně. Jednou z možností je určit, který z operátorů má nejvyšší prioritu a je vyhodnocen jako první, který má prioritu nižší a je vyhodnocen jako

druhý, a který bude vyhodnocen až nakonec (v rámci příkazu nebo závorky). Druhou možností je procházet dotaz sekvenčně a vyhodnocovat operátory tak, v jakém pořadí jsou v příkazu napsány.

2.6 GUI

Součástí aplikace je také jednoduché grafické prostředí, pod kterým logická část programu poběží. Pravděpodobně nejnápadnější volbou je vytvoření jednoduché okenné aplikace, která bude obsahovat pole pro vložení nových dokumentů, pole pro vyhledávání a pole s výsledky.

Případnými vylepšeními by bylo například zobrazování náhledu nejlepšího dokumentu s vyznačenými klíčovými slovy, nebo získání statistik o výsledku vyhledávání (doba hledání, počet dostatečně relevantních výsledků apod.).

Pro jednoduchost a z časové tísně jsem se rozhodl pouze pro řádku na dotazy, počet dokumentů, které má systém vrátit a tlačítko pro odevzdání. Samozřejmě níže je pak místo pro obdržení výsledků.

2.7 Testy

Aplikace by měla být alespoň do určité míry otestována automatickými testy, které do jisté míry zaručují funkčnost a konzistenci řešení.

2.8 Programovací jazyk

Pro implementaci této aplikace bylo doporučeno využít předpřipravené kostry v jazyce Java a následně v tomto jazyce také aplikaci napsat.

Jelikož se jednalo o aplikaci, která z velké části odpovídala zadání předmětu KIV/NET, rozhodl jsem se dodat několik funkčností do této aplikace a rozšířit jí tak, aby odpovídala druhému zadání. Součástí tohoto zadání je, že aplikace musí být alespoň částečně napsaná v programovacím jazyce C#. Z tohoto důvodu je tato aplikace psána v s použitím technologie ASP.NET a jazyka C#.

Z tohoto důvodu není využita poskytnutá kostra.

3 Implementace

3.1 Data

Data využívána v této práci byla crawlována v rámci prvního cvičení a pocházejí z webu https://prima-receptar.cz/. Další data, na kterých byla aplikace

testována jsou data obdržená z již zmíněné kostry projektu ve formátu .json.

3.2 Načítání dat

V rámci implementace program obsahuje způsob jednorázového načtení dokumentů, jejich předzpracování včetně odstranění stopslov a vytvoření invertovaného indexu. Nad takto zpracovanými daty lze pokládat dotazy a očekávat výsledek, kterým je index několika dokumentů s největší kosínovou podobností k dotazu. V případě dotazování se pomocí logických operátorů, výsledek bude obsahovat několik prvních dokumentů, které budou nalezeny, nikoliv ty nejlépe hodnoceny.

3.3 Booleaovský model vyhledávání

Poté co aplikace od uživatele obdrží vyhledávací dotaz, je tento dotaz prozkoumán, zda neobsahuje klíčové slovo "AND", "OR", nebo "NOT". Pokud je v dotazu jedno z těchto slov alespoň jednou uvedeno, je na dotaz nasazen booleovský vyhledávací model.

V rámci tohoto typu vyhledávání je dotaz nejprve rozdělen podle závorek na části s přiřazenou prioritou zpracování. Dotazy hlouběji v závorkové struktuře mají vyšší prioritu než ty mimo závorky a dotazy vlevo mají vyšší prioritu než ty vpravo.

Následně je každá část dotazu zpracována a nahrazuje prázdné místo dalších dotazů. Snad pro přiblížení funkčnosti algoritmu přikládám jednoduchý příklad nějakého pseudokódu.

```
(pes OR kocka) AND morce ->
-> pes OR kocka (2) | AND morce (1)
vec1 = querry(pes OR kocka)
vec2 = querry(vec1 AND morce)
result = vec2
```

Zde je vidět, jak se části kódu rozdělí a jak je jim do závorek přiřazena priorita. Pro tuto prioritu hraje největší roli zanoření dotazu vzhledem k závorkování, a pak dotazy na stejné úrovni jsou řazeny zleva. Operátory na tyto dotazy nemají vliv, není tedy žádný z operátorů upřednostněn.

Výsledkem hledání je jeden vektor se kterým jsou dokumenty porovnávány a indexy těch, které odpovídají dotazu jsou uloženy pro následné vypsání výsledku.

3.4 GUI

Grafické rozhraní je vytvořeno pomocí nástroje WPF, které je součástí ASP.NET. Rozhraní se skládá pouze z jednoho hlavního okna. To obsahuje dvě textová pole, kam uživatel zadává dotaz a počet dokumentů, které chce obdržet. Dále je zde tlačítko pro inicializaci vyhledávání a velká část okna vyhrazena pro výsledky. Poslední částí je pravý panel, který obsahuje historii vyhledávání (dotaz, čas zpracování a počet získaných výsledků, případně nejlepší výsledek).

3.5 Předzpracování

Pro implementaci předzpracování jsem využil programu naprogramovaného pro úkol z KIV/IR. Velká část tohoto programu není napsána mnou, ale byla mi poskytnuta KIV/IR za účelem vypracování domácího úkolu a následně této práce.

Pro předzpracování jsem si vybral pouze potřebné části a přepsal je do jazyka C#. Těmito částmi jsou "AdvanceTokenizer", "CzechStemmer"a "BasicPreprocessingExample". Tyto třídy jsem přepsal a upravil pro potřeby této práce, ale některé jejich části zůstali neporušené.

3.6 Tf-Idf

Při implementaci Tf-Idf jsem postupoval podobně jako u preprocessingu. Jedná se o jeden z úkolů pro KIV/IR s tím rozdílem, že celá část Tf-Idf byla psána mnou rovnou v jazyce C#. Nebylo tedy tak obtížné její upravení pro účely této práce a zasazení do projektu.

Třída obdrží předzpracované dokumenty k indexaci. Nejprve je vytvořen invertovaný index (jako HashMap) a následně jsou vygenerovány vektory k jednotlivým dokumentům. To je uděláno nejdříve získáním všech Idf a následným spočítáním vah. Tyto váhy pak tvoří jednotlivé složky vektorů.

Součástí Tf-Idf je i možnost stejným způsobem zpracovat obdržený dotaz a následné dotázání se na vektory souborů na jejich kosínovou podobnost. Čím vyšší je kosínová podobnost dotazu a dokumentu, tim více dokument dotazu odpovídá.

3.7 Spuštění programu

Výsledkem práce je spustitelný .exe soubor. Tento soubor automaticky indexuje data získaná z prvního úkolu z crawlování. Pokud by chtěl uživatel

zaindexovat jiná data, musí program spustit z příkazové řádky a vložit za něj parametr s cestou ke složce se soubory.

Soubor se kterým se tento program pouští automaticky obsahuje přibližně 700 dokumentů o celkové velikosti cca 4MB. Pro představu rychlosti přečtení a zaindexování souborů, json soubory poskytnuté KIV/IR jsou čteny přibližně 4 minuty (81 735 .jsou dokumentů). Po tuto dobu bohužel aplikace nejeví známky spuštění a pracuje pouze na pozadí.

Po spuštění jsou přečteny poskytnuté dokumenty, zaindexovány a je zobrazeno okno aplikace. Okno a postup je intuitivní a je popsán v kapitole GUI.

Pokud by číslo pro počet dokumentů nebylo vyplněno, nebo v něm nebylo celé číslo, je automaticky nastaveno na hodnotu 5.

Výsledek obsahuje čas za který byl výsledek vyhledán, pak na se skládá z několika řádek (podle toho kolik jich uživatel požadoval) strukturovaných do vět informujících uživatele, jaké dokumenty a s jakou přesností odpovídají jeho dotazu. Tyto dokumenty jsou popsány indexem dokumentu a desetinným číslem, které bylo získáno kosínovou podobností (pokud bylo vyhledáváno pomocí logických operátorů, tato informace ve výpisu není). Pod těmito výsledky je ještě jednou vyzdvižen nejlepší výsledek a počet dokumentů, které odpovídají dotazu.

4 Nadstandartní implementace

Krom základní požadované funkčnosti je v rámci aplikace dále implementováno:

- Grafické rozhraní GUI (i když v poměrně jednoduché formě)
- Dokumentace psaná v TEXu
- Vlastní implementace parsování dotazů bez použití externí knihovny (více v kapitole Booleavský model vyhledávání)
- Jednoduchá forma historie vyhledávání.

5 Zdroje

Při práci jsem vycházel převážně ze znalostí získaných z předmětů KIV/IR a KIV/NET. Pro analytickou a částečně implementační část jsem vycházel převážně ze cvičení a domácích úkolů KIV/IR. Pro zbytek implementace a grafické rozhraní jsem vycházel z přednášek a cvičení KIV/NET.

6 Závěr

Aplikace obsahuje několik designových rozhodnutí, které nejsou ideální a jedná se o možný budoucí vylepšení. Mezi to se řadí například čtení indexování souborů před otevřením okna, načítání souborů pouze při startu, nebo zobrazení náhledu výsledku.

Naopak co aplikace dělá dle mého názoru správně, je relativně rychlá indexace a vyhledávání.

Bohužel se mi nepodařilo projekt napojit na Vámi poskytnutou kostru, která obsahovala i ohodnocení získaných výsledků. Je tomu tak z důvodu absence/neznalost knihoven, které jsou v kostře použity. Ekvivalent k těmto knihovnám pro C# se mi bohužel nepodařilo najít. Věřím, že kdybych na toto zprovoznění měl více času, nebyla by to taková překážka.