Information Retrieval: Rozšířený booleovský model

Popis projektu

Cílem projektu je implementace rozšířeného booleovského modelu ukládání dat (tedy preprocessing těchto dat a indexování) spolu s možností dotazování z GUI.

Vstupem aplikace je booleovský dotaz skládající se z termů, na které se uživatel dotazuje a ze tří logických spojek, které představují booleovské operace. Jedná se o spojky ˝&˝ (která reprezentuje operaci AND), ˝|˝ (která reprezentuje operaci OR) a ˝!˝ (která reprezentuje operaci NOT). Mezi každými dvěma složkami dotazu se musí nacházet mezera.

Výstupem aplikace je seznam dokumentů nacházejících se v databázi seřazený v klesajícím pořadí dle relevance těchto dokumentů vzhledem k dotazu. Každý dokument je na výstupu reprezentován jako položka seznamu, která je identifikována názvem dokumentu a umožňuje zobrazit jeho obsah.

Způsob řešení

Booleovský model je jeden ze způsobů, jak prohledávat kolekci dokumentů s cílem identifikace dokumentů obsahující termy odpovídající dotazu. Dotaz je přitom tvořen booleovským výrazem (popsáno v předchozí kapitole).

Extrakce a preprocessing termů z dokumentů

Dotaz je vyhodnocován oproti kolekci dokumentů, tedy každý dokument lze chápat jako objekt databáze. V případě (rozšířeného) booleovského modelu projde každý dokument nejdříve fází preprocessingu, kdy jsou z dokumentu odstraněna nevýznamová slova (tedy slova, která nesou málo informací, jako jsou např. spojky a předložky) a významová slova jsou stemmována či lemmatizována za účelem získání základních tvarů slov. V našem případě byl v rámci preprocessingu uplatněn stemming termů.

Efektivní uložení dokumentů pomocí invertovaného seznamu

Po preprocessingu máme tedy k dispozici kolekci slov, kterou je třeba uložit takovým způsobem, aby v ní šlo efektivně vyhledávat. U booleovského modelu je každý dokument uložen jako binární vektor, čímž dostáváme tzv. term-by-document matici, kde na i-tém řádku v j-tém sloupci je 1 právě tehdy, když je term i obsažen v dokumentu j. Takový přístup však nedokáže rozlišit, jak moc daný term vystihuje dokument, ve kterém se nachází. Nelze říct, zda se term i vyskytuje v dokumentu pouze okrajově, nebo je celý dokument právě o tomto termu. Z tohoto důvodu jsou v term-by-document matici reálné hodnoty v rozmezí 0 až 1, definující váhu (důležitost) termu pro dokument. Určování vah je typicky založeno na frekvenci výskytu termu v dokumentu a výskytu termu přes celou kolekci. Nejznámější schéma založené na tomto principu se nazývá tf-idf (term frequency – inverse document frequency) schéma. Každý dokument je pak možné popsat n-dimenzionálním vektorem, kde n je velikost slovníku, a tedy lze chápat jako bod v n-dimenzionálním prostoru.

Schéma tf-idf pro vážení termů v dokumentech

Pro získání váhy termů v dokumentech bylo využito již zmiňované tf-idf schéma. Váha termu i v dokumentu j byla získána následujícím vztahem.

$$w_{ij} = tf_{ij} * \ idf_i = tf_{ij} * \log_2\left(\frac{n}{df_i}\right) = \frac{f_{ij}}{\max(f_{ij})} * \ \log_2\left(\frac{n}{df_i}\right)$$

- $f_{ij} = frekventovanost\ termu\ i\ v\ dokumentu\ j$ $(počet\ jeho\ výskytů\ v\ dokumentu\ j)$
- $\max(f_{ij}) = maximální\ frekventovanost\ termu\ i\ napříč\ celou\ kolekcí\ dokumentů$
- $n = celkový\ počet\ dokumentů\ v\ kolekci$
- $df_i = frekv.termu\ i\ v\ dokumentech$ $(počet\ dokumentů\ obsahujících\ term\ i)$

Uložení do invertovaného seznamu

Namísto term-by-document matice, která obsahuje pro každý dokument vektor termů v něm se vyskytujících (včetně jeho vah), byl uplatněn invertovaný seznam. Invertovaný seznam přichází s odlišným přístupem k ukládání dat, který je založený na vytvoření struktury obsahující všechny termy vyskytující se ve slovníku. Ke každému termu je pak přiřazen seznam dokumentů, ve kterých se daný term vyskytuje spolu s váhou tohoto termu v daném dokumentu.

Vyhodnocovací/dotazovací modul

Stejně jako dokument lze reprezentovat i dotaz. Podle boolovského výrazu reprezentujícím dotaz lze bod dosadit na kraj prostoru (souřadnice dotazu můžou obsahovat pouze 1 nebo 0, protože jde stále o boolovské dotazování).

Definujeme-li pak nějakou vzdálenost mezi dvěma body v n-dimenzionálním prostoru (u boolovského modelu je to euklidovská (L2) nebo obecně Lp metrika), lze podobnost dokumentů chápat jako převrácenou vzdálenost bodů, které je reprezentují.

Vyhodnocení váhy dokumentu vůči dotazu

Pro operaci AND (tedy dotaz typu q = k_1 AND k_2):

$$relev(q, d_j) = 1 - \sqrt{\frac{(1-w_{1,j})^2 + (1-w_{2,j})^2}{2}}$$

Pro operaci OR (tedy dotaz typu q = k_1 OR k_2):

$$relev(q, d_j) = \sqrt{\frac{{w_{1,j}}^2 + {w_{2,j}}^2}{2}}$$

Pro operaci NOT (tedy dotaz typu q = NOT k):

$$relev(q, d_j) = 1 - w_j$$

Implementace

Stavba aplikace

Uložiště

Jako persistentní uložiště pro kolekci dokumentů byla zvolena dokumentová databáze MongoDB. Konkrétně se jedná o cloudovou databázovou službu MongoDB Atlas. Pro komunikaci s databází je využívána technologie Node.js, která poskytuje balíček pro jednoduchou práci s MongoDB. Tento balíček umožňuje manipulaci s databázovými daty ve formátu JSON. Datový základ aplikace je tedy uložen lokálně ve formátu JSON.

Pro práci s uložištěm slouží skripty fill_collecion.js pro upload dat na cloud a drop_collection.js pro smazání dat z cloudu.

Extrakce termů a preprocessing

Pro extrakci termů z dokumentů nacházejících se v kolekci a preprocessing je využíván programovací jazyk Python (verze 2.x). Python byl zvolen díky své jednoduchosti a také proto, že poskytuje vhodné moduly pro preprocessing. Mezi využité moduly patří nltk, z něhož byl převzat seznam stopwords obsahující nevýznamová slova a dále byl použit PorterStemmer pro stemming významových slov.

V rámci preprocessingu probíhá také výpočet vah jednotlivých termů v konkrétních dokumentech. Za tímto účelem probíhá výpočet potřebných údajů. Je nutné zjistit celkový počet dokumentů v kolekci, nejvyšší frekventovanost jednotlivých termů v celé kolekci, počet dokumentů obsahujících jednotlivé termy a počet výskytů jednotlivých termů v jednotlivých dokumentech. Na základě těchto údajů je pomocí vztahu z předchozí kapitoly spočítána váha každého termu v každém dokumentu, ve kterém je daný term obsažen. Tyto váhy jsou nakonec normalizovány na maximální hodnotu 1.

Implementace invertovaného seznamu

Z dat získaných během preprocessingu je nakonec sestaven invertovaný seznam jako pole všech termů vyskytujících se v získaném slovníku. Každý term v invertovaném seznamu má přiřazený seznam dokumentů, ve kterých se daný term vyskytuje a jeho váhu v tomto dokumentu.

Parsování a vyhodnocení dotazu

Pro parsování a vyhodnocení booleovského dotazu byl využit modifikovaný algoritmus pro vyhodnocování základních booleovských výrazů obsahujících pouze hodnoty true a false. Tento algoritmus je volně dostupný na internetu a bude uveden ve zdrojích. Parsování booleovského dotazu provádí rozklad vstupního dotazu do syntaktického stromu dotazu. Tento strom obsahuje čtyři typy uzlů. Jedná se o binární uzly (mají vždy dva potomky) reprezentující operace AND a OR, unární uzel (má vždy jednoho potomka) reprezentující operaci NOT a nakonec list obsahující vždy váhu konkrétního termu v konkrétním dokumentu. Během parsování jsou rovnou do těchto listů dosazovány potřebné váhy.

Jakmile je syntaktický strom dokončen, je možné získat hodnotu kořenového uzlu stromu. Uzly stromu jsou implementovány tak, aby volání hodnoty kořenového uzlu spustilo volání metod vracejících hodnoty uzlů na nižších hladinách stromu výrazu. To způsobí, že volání těchto metod se zanoří až do nejnižší hladiny stromu. Jakmile se volání zastaví, začnou se při rekurentním vynořování z volání počítat váhy daného dokumentu vůči dané části dotazu. Tyto vypočtené váhy se rekurentně dosazují, dokud se volání nevynoří zpět ke kořenovému uzlu a výsledkem není finální váha dokumentu vůči dotazu.

Takového vyhodnocení je provedeno pro každý dokument. Kolekce dokumentů je pak seřazena sestupně dle získaných vah a odeslána na výstup.

Pro výpočet vah jsou využívány vztahy z kapitoly pojednávající o způsobu řešení.

Webové rozhraní

Webové grafické rozhraní je realizováno pomocí technologií HTML, CSS3 a Angular5. Skládá se z inputu pro zadání booleovského dotazu. Výsledek vyhledávání je zobrazen pod inputem. Během vyhledávání je zobrazován rotující spinner implementovaný pomocí CSS3. Každá výsledku reprezentuje jeden dokument, je klikatelná a umožňuje zobrazit obsah vyhledaného dokumentu.

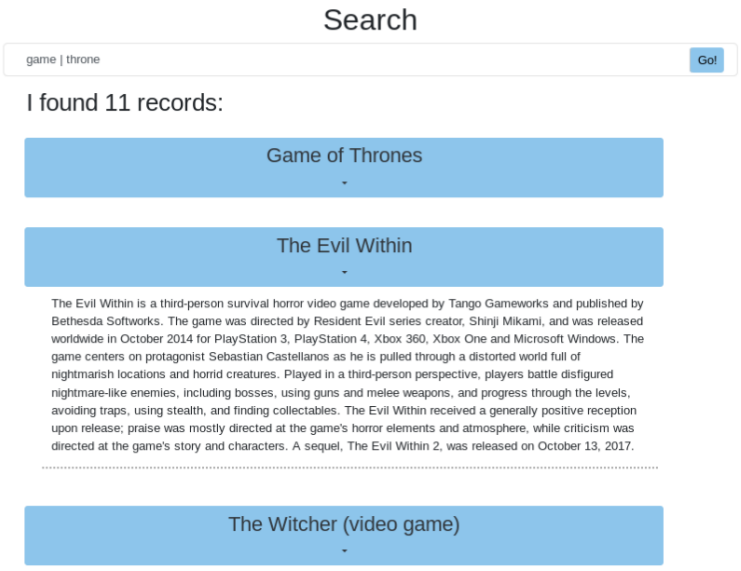
Požadavky na běh

- Python verze 2.7 a vyšší
- Angular5
- JavaScript NPM package manager
- Node.js mongodb driver
- Modul NLTK pro Python

Příklad vstupu/výstupu

Ukázka výstupu aplikace při zadání jednoduchého dotazu „game“.

Ukázka výstupu aplikace při zadání jednoduchého dotazu „game | throne“. Lze pozorovat prohození dokumentů na prvních dvou pozicích výsledku.



Ukázka výstupu aplikace pro ten samý dotaz, pouze se zobrazeným obsahem jednoho z dokumentů vyskytujících se ve výsledku.

Experimentální sekce

V této kapitole se zaměříme na porovnávání časové efektivity vyhledávání v booleovském modelu za využití invertovaného seznamu a za pomoci naivního řešení ve formě sekvenčního procházení kolekce. Za tímto účelem bylo naměřeno několik časových údajů vyjadřujících čas vyhledávání v databázi při zadání uvedených booleovských dotazů.

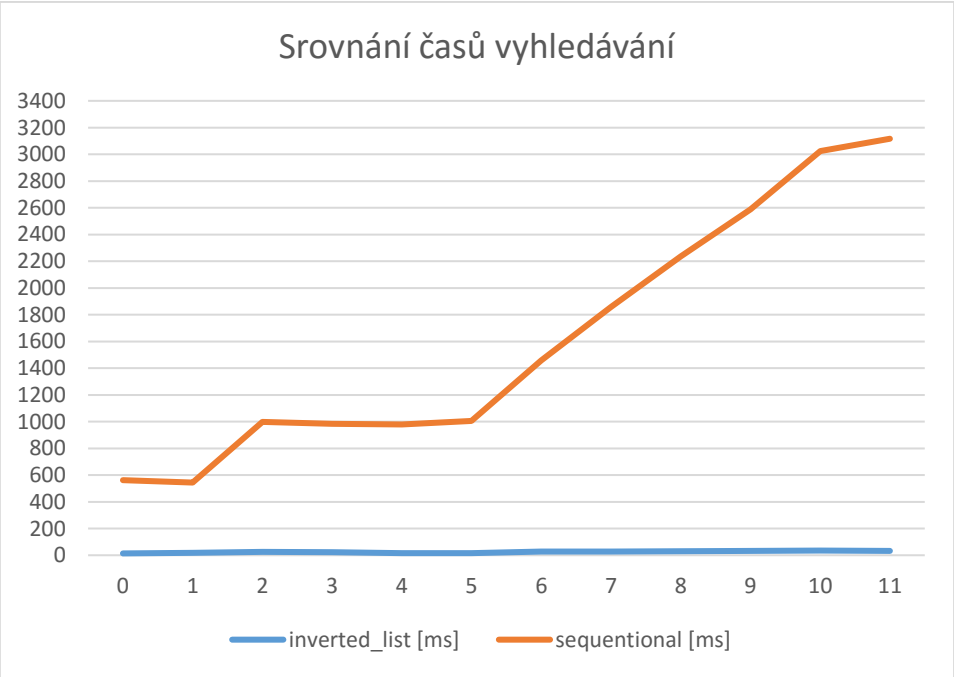
Naměřené hodnoty nezohledňují čas potřebný k získání dat z databáze na cloudu MongoDB Atlas, vyjadřují pouze čas strávený v kritické části programu, ve které probíhá sestavení výsledku vyhledávání.

Výsledky zanesené do tabulky

INDEX	DOTAZ
0	game
1	! game
2	game | throne
3	game & throne
4	game | ! throne
5	game & ! throne
6	(game | throne) & rock
7	(game | throne) & rock & metal
8	((game | throne) & rock & metal) | geralt
9	((game | throne) & rock & metal) | geralt & ciri
10	link | ((game | throne) & rock & metal) | geralt & ciri
11	! (link | ((game | throne) & rock & metal) | geralt & ciri)

INDEX	INVERTED LIST [ms]	SEQUENTIONAL [ms]
0	13,788	562,931
1	18,588	543,909
2	25,758	998,59
3	22,676	984,504
4	16,356	978,149
5	14,9	1003,836
6	27,386	1457,516
7	28,091	1858,04
8	30,407	2235,24
9	32,845	2589,324
10	35,098	3023,96
11	31,558	3116,379

Výsledky zanesené do grafu



Vyhodnocení měření

Z naměřených hodnot jednoznačně vyplývá, že řešení využívající invertovaný seznam je časově efektivnější než naivní sekvenční řešení. Dalším jednoznačným faktem je, že s rostoucí komplexností dotazu roste také čas vyhledávání. Zároveň lze vypozorovat, že v případě invertovaného seznamu způsobují operace OR a AND přibližně stejně zpomalení. V případě primitivních dotazů způsobí větší zpomalení operace OR. Časově nejméně náročná je operace NOT, která v určitých případech dokonce snížila čas vyhledávání. V případě sekvenčního hledání pak čas vyhledávání roste přibližně stejně pro všechny operace s výjimkou primitivního dotazu pro operaci NOT, která čas vyhledávání mírně snížila.

Diskuze

Tato práce není dokonalým řešením zadaného problému. Například ve fázi experimentování by bylo možné aplikaci dále časově ladit pomocí úpravy hodnot mocniny a odmocniny během počítání váhy dokumentů vzhledem k dotazu. Stejně tak by bylo možné ošetřit vstupy aplikace tak, aby v případě zadání chybně formátovaného dotazu tuto chybu odhalovala. To by vyžadovalo zásah do parsování dotazu.

Závěr

Projekt byl velice zajímavým seznámením s konceptem NoSQL, jelikož jsme využívali databázi MongoDB. Poprvé jsme se v něm seznámili s teorií vyhledávání v dokumentech, která je až překvapivě rozsáhlá, zformovaná a je zajímavou aplikací matematiky. Zároveň je však rozumně implementovatelná a vcelku snadno se dalo ověřit, že implementace odpovídá předpokladům. Během práce bylo důležité hledat ty správné technologie, které umožnily rozumné řešení problémů a následně je propojovat dohromady.

Zdroje

How To Build A Boolean Expression Evaluator. *Unniked.org* [online]. 2015 [cit. 2018-05-11]. Dostupné z: https://unnikked.ga/how-to-build-a-boolean-expression-evaluator-518e9e068a65