FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZPRÁVA K PROJEKTU I-2
INFORMATION RETRIEVAL – ROZŠÍŘENÝ BOOLOVSKÝ MODEL

Vítězslav Hušek, Jan Flajžík

2022

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

OBSAH

Úvod do problematiky

Každý se denně setkáváme s vyhledávači, které nám pomáhají vyhledat nějaký konkrétní dokument na základě jeho obsahu. Ať už vyhledáváme internetovou stránku přes vyhledávače jako seznam.cz nebo google.cz, nebo pouze hledáme dlouho zapadlý email ve své emailové schránce. Pokud pro zjednodušení budeme chápat každou stránku, každý email pouze jako dokument obsahující text, získáme jeden ze stavebních kamenů Boolovského modelu. Samozřejmě s přibývajícím počtem dokumentů také (v závislosti na typu dokumentů) roste počet slov (termů), které budeme muset nějakým způsobem zpracovávat. Právě toto předzpracování dokumentů je přitom zásadní pro to, aby se celý proces vyhledávání dal stihnout v nějakém rozumném čase. Každý jazyk má slova, která se velice často opakují, a přitom nejsou nijak vypovídající o obsahu daného dokumentu (v angličtině mezi tato slova patří: a, the, are, have, ...) a také slova, která v různých tvarech znamenají to samé (go, went, going). Zbavení se těchto slov zjednoduší jak ukládání dokumentů, tak i následné vyhledávání. Základní ukládání se pak dá realizovat maticí, kde sloupce představují jednotlivé termy a řádky jednotlivé dokumenty. Pro rozšířený boolovský model se pak pro každý dokument ukládá počet výskytů daného termu (to umožňuje srovnávat relevance dokumentů pro jednotlivé termy). Vzhledem k tomu, že by taková matice nejspíše byla z velké části prázdná (specifické termy se v spoustě dokumentech nevyskytnou vůbec), využívá se tzv. invertovaný index, kde každý term má seřazený seznam dokumentů, ve kterých se vyskytuje. Následuje předzpracování a předpočítání váhy jednotlivých termů pro dokumenty, se kterou se dále pracuje při uživatelském vstupu a podle které se poté počítají relevance jednotlivých dokumentů pro výstup. Přístup, který jsme zvolili pro náš projekt dále rozebírám v popisu projektu a způsobu řešení.

Popis projektu

Pro náš projekt jsme se rozhodli vytvořit vyhledávač písniček. Snad každému se někdy stane, že zapomene název písničky, kterou si chce pustit, a pamatuje si pouze útržky z textu. Z tohoto důvodu jsme se shodli, že by bylo zajímavé takový vyhledávač v menším měřítku implementovat. Projekt tedy tvoří webové grafické rozhraní, které bere na vstupu od uživatele boolovský dotaz (tedy jednotlivá vyhledávaná slova spojená logickými spojkami AND a OR – i s možností složených dotazů v závorkách) a na jeho základě prohledá naši předzpracovanou kolekci písniček a vrátí nejvíce relevantní písničky pro uživatele. V případě, že uživatel zadá velice specifické slovo, které se nevyskytuje v žádné ze zpracovaných písniček (nebo nechá svoje domácí zvíře projít se po klávesnici), bude uživateli doporučeno několik náhodných písniček (a samozřejmě bude upozorněn že se jedná o náhodný výběr, nikoliv o jakékoliv relevantní výsledky vzhledem k vstupu).

Způsob řešení

Na úplném začátku jsme se rozhodli rozdělit si práci tak nějak rovnoměrně, abychom každý udělali nějakou část, které bychom poté spojili do funkčního celku. Celý projekt jsme si tak rozdělili do tří částí:

1) Získání písniček, extrakce termů, identifikace nevýznamových slov, stemming/lematizace
2) Výpočet vah termů, implementace invertovaného seznamu, parsování uživatelského dotazu, vyhodnocení dotazu
3) Webový interface

Vzhledem k tomu, jak jsme si rozdělili práci, vznikal druhý bod paralelně s prvním, takže bylo potřeba najít nějakou testovací kolekci dokumentů (písniček), nad kterou by výpočet vah termů a implementace invertovaného seznamu byl možný. Většinu času vývoje se tak pracovalo s testovacími daty ze stránky Milion Song Dataset, kde se ale postupem času projevila jako velká nevýhoda nemožnost porovnávat výsledky vyhledávání našeho modelu s textem písniček, ze kterých je kolekce dokumentů na této stránce tvořena (texty na aktuálních stránkách mohou být upravovány a korigovány každým dnem, takže původní data nemusí odpovídat původním, ze kterých byl dataset tvořen).

K výpočtu váhy jednotlivých termů využíváme výpočetní schéma *tf-idf,* který je uváděno v BI-VWM přednáškách vektorového modelu. To vyžaduje statickou kolekci dokumentů, nad kterou my pracujeme. Každé přidání nové písničky by totiž v našem smyslu znamenalo po úvodním předzpracování dále upravit relevance i pro ostatní termy a dokumenty, což může vzhledem k rostoucímu počtu dokumentů (písniček) být celkem časově náročná operace, takže jsme raději zvolili statickou kolekci, kde se toho zpracování provede pouze jednou a pak lze (poměrně rychle, oproti délce tohoto předzpracování) už pouze vyhledávat.

Implementace

Projekt je realizován kompletně v Pythonu (předzpracování, backend). Frontend je realizován ve frameworku Django.

ZÍSKÁVÁNÍ DAT

Pro získávání dat, ve kterých bude náš program vyhledávat jsme využili API LyricsGenius https://lyricsgenius.readthedocs.io , které nám umožňovala získavat texty k písničkám uložených v databázi textů https://genius.com/. Pro připojení bylo třeba přes správce balíčků pip stáhnout balíček lyricsgenius, a poté přes free access token zísakt připojení do databáze.

Pro vybrání testovcích písní jsme využili možnost vyhledávat podle autora. Na stránce https://genius.com/verified-artists je seznam autorů s „nejvetším IQ“, což jsou body, kterými správci této

stránky označují, kolik informací mají o daných hudebnících. Vybrali jsme tedy několik z těch nejlepších. Dále jsme si k tomu přidali několik našich oblíbenců. Zbytek jsme vyplnili texty ze písní na seznamu vytvořeném v původním testovacím datasetu ze stránky Milion Song Dataset, který je popsaný v 1. části dokumentu. Toto nám vystačilo na 30000+ písní. Písně jsme ukládali do 1 textového souboru oděleného separátorem ----NEWSONG---. Každá píseň má před začátkem textu hlavičku ve tvaru *songid<SEP>autor<SEP>nazev*, kde songid je id převzaté z původní database genius. Tento způsob uložení dat jsme zvolili pro jednodušší stemmingu a taky jednoduchosti vyhledávání přes příkaz grep.

STEMMING

Pro stemming využíváme knihovnu nltk a její modul Snowball stemmer, který je pro naše účely vhodnější než porter stemmer. Protože naše písničky nejsou jen v angličtině, bylo třeba také zvolit knihovnu pro detekci jazyka. My jsme se rozhodli pro knihovnu langdetect https://pypi.org/project/langdetect/, protože funguje dobře pro delší texty. Knihovna ovšem vrací názvy jazyků ve formátu zkratek ISO 639-1, i když stemmer potřebuje celý název jazyka, proto jsme museli do kódu pro stemming přidat externí slovník, který tyto dvě položky mezi sebou převáděl. Když funkce pro stemming dostala text písně jako argument, probíhal algoritmus následovně: Nejdříve jsme z textu odstranili hlavičku, kterou přidávalo api ke každému textu. Potom jsme z textu odstranili veškerý obsah v závorkách, protože závorky obsahují věčně opakující se slova jako chorus nebo počet opakování verše. Poté jsme detekovali jazyk, kterým je píseň zpívaná. Dále jsme odstranili interpunkce, stop-slova a přestemmovali to, co z textu zbylo. Knihovna pro rozpoznání jazyka bohužel není vždy stoprocentně přesná třeba proto, že někteří umělci mají tendence měnit jazyk uprostřed písně, takže bylo třeba uložit defaultně stop-slova jako anglická a pokud byl detekován jiný jazyk došlo k sjednocení těchto dvou množin. Riziko odstranění důležitého slova pro jiné jazyky bylo zanedbatelné.

Po provedení stemmingu bylo třeba spočítat top-words a rozdělit výstup do 3 souborů. Jako top-words jsme zvolili polovinu všech slov, která zbyla po stemmingu s nejvyšším výskytem v dokumentu. Výstup rozdělil do tří souborů. V souboru songlist.txt je uložen seznam všech písní ve fromátu *songid<SEP>autor<SEP>nazev.* V souboru occSet.txt je uložen výskyt jednotlivých top-words pro každý song určený songid. V souboru topWords.txt už jsou pak seřazena všechna topWords podle miry výskytu.

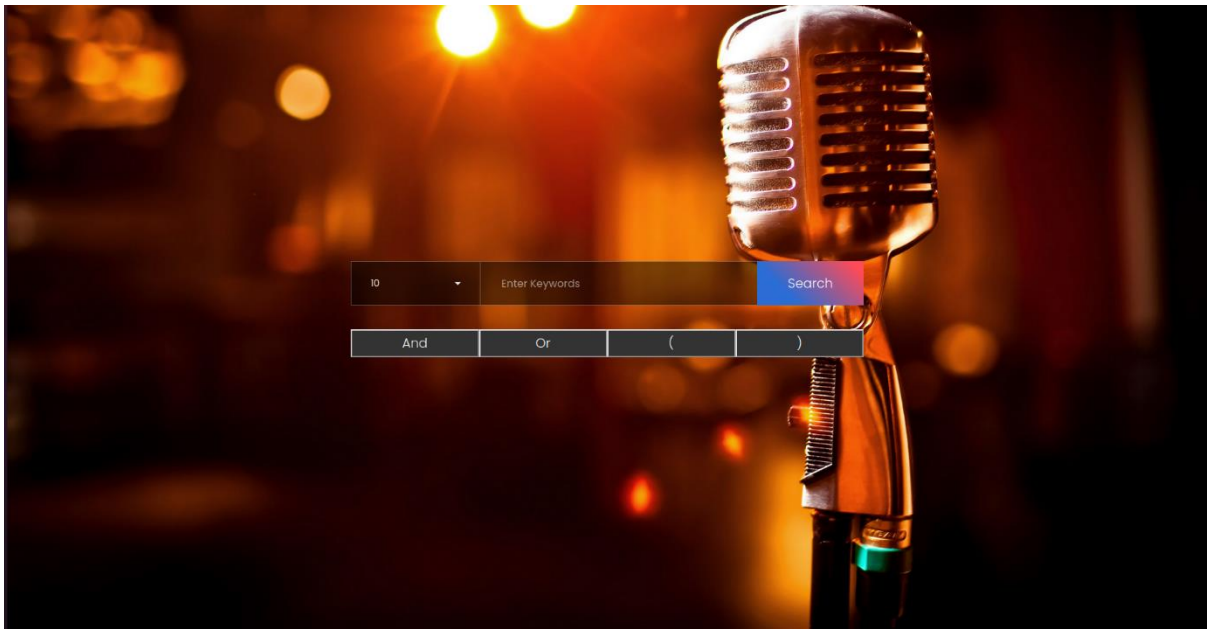
VYHODNOCENÍ VÝRAZU

K vyhodnocování vstupu od uživatele je využívána Python knihovna PyEDA, respektive její část zaměřující se na Boolskou algebru. Vstup od uživatele je zpracován touto knihovnou a převeden do disjunktivní normální formy, pro zjednodušení následné práce s tímto vstupem.

Pro zjednodušené ukládání záznamů je využíván container *defaultdict* z modulu collections. Ten se chová v zásadě jako klasický *dict* s výjimkou, že pokud záznam s vyhledávaným klíčem v daném containeru není, nedojde k chybě, ale místo toho se vrátí záznam s výchozí hodnotou.

WEBOVÉ ROZHRANÍ

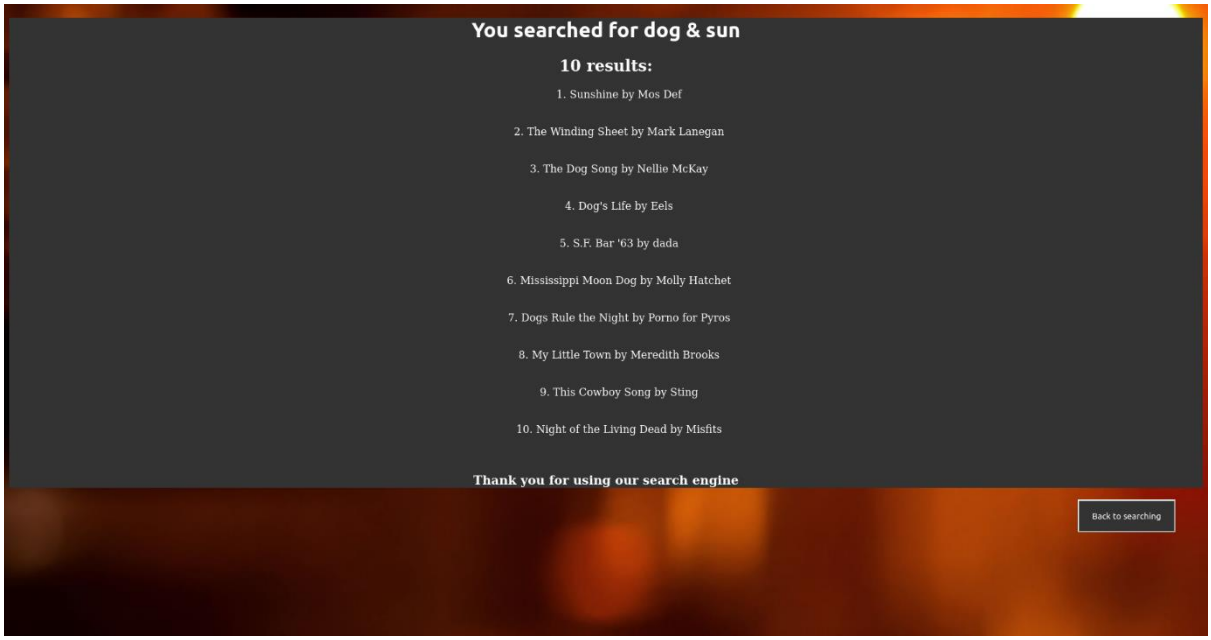
K vytvoření webového rozhraní jsme využili pythonový framework Django. Celý web je rozdělen na 2 stránky. Úvodní stránku s textovým polem pro zadaný dotaz. K tomu jsou na této stránce 4 tlačítka pro přidávání symbolů AND, OR a závorek pro intuitivnější práci s vyhledáváním a panel, který umožňuje po rozkliknutí uživateli zvolit počet vybraných výsledků. Šablonu pro tuto rozhraní jsme našli na https://colorlib.com/etc/searchf/colorlib-search-25/ s tím, že bylo třeba přidat tlačítka pro symboly a upravit pozadí. Celá šablona je vytvořena kombinací html + css + javascript pro práci s tlačítky.



Druhá stránka s výsledky je celá z naší dílny. Slouží čistě k tomu, aby byly uživateli zobrazeny výsledku jeho dotazu. Opět využíváme html +css jen pro zobrazení výsledků vyhledávání. Pod polem s výsledky se nachází tlačítko pro návrat na úvodní obrazovku s vyhledáváním. Ukázka stránky s výsledky je ukázána v části „Příklad výstupu“ i s konkrétním vstupem od uživatele.

Příklad výstupu

Na následujícím obrázku lze vidět příklad výstupu z aplikace pro uživatelský vstup dog & sun a zvolení top 10 nejpřesnějších dokumentů (písniček s příslušnými autory), jejichž text by měl odpovídat zadanému dotazu. Při vypisování výsledků jsme se zaměřili na jednoduchost a výpis je tedy poměrně skromný a zobrazuje vyhledávaný výraz, počet vypsaných výsledků a očíslovaný seznam vybraných písniček – jejich název následovaný autorem.



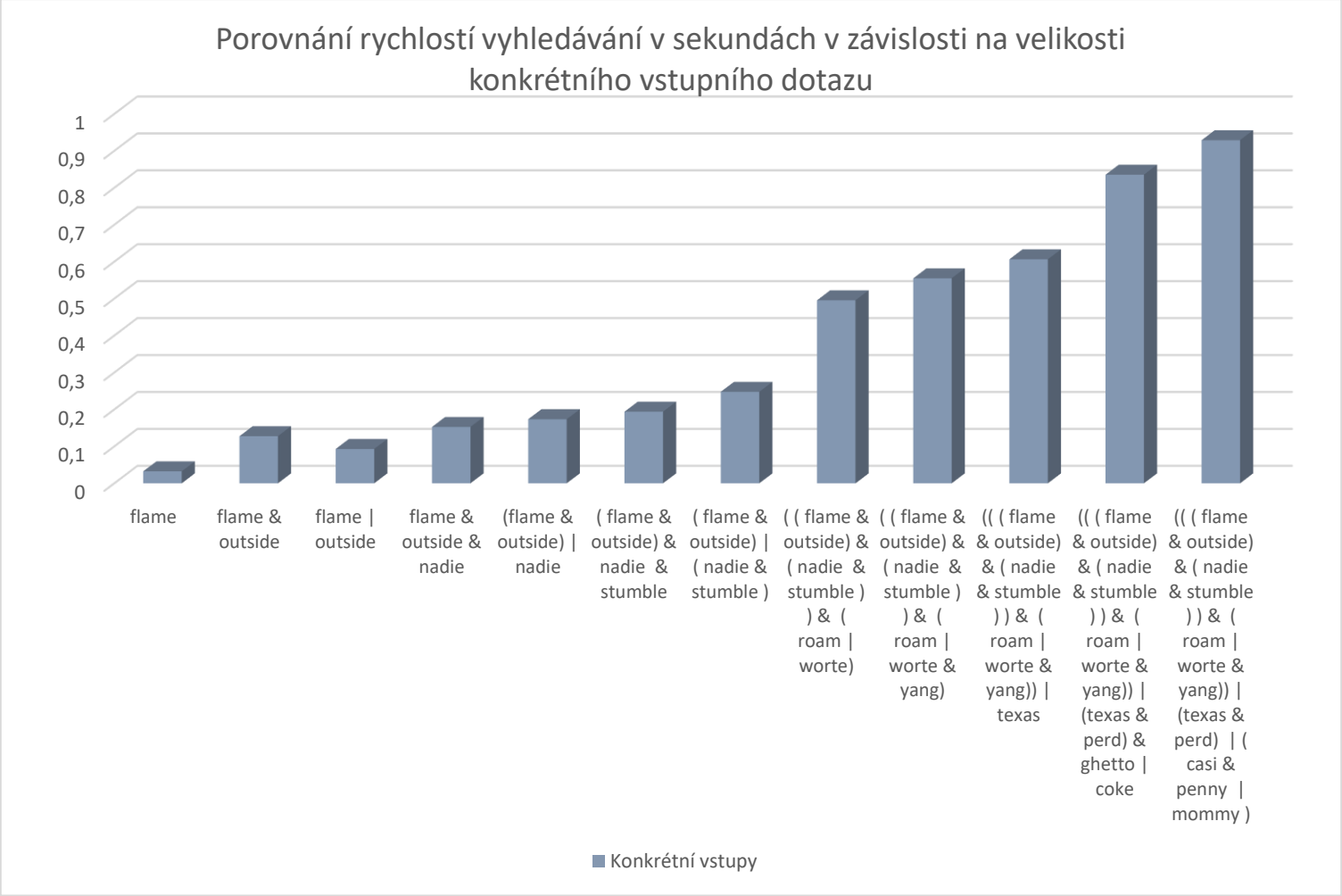
Experimentální Sekce

V experimentální sekci se zaměříme na rychlost vyhledávání v závislosti na počtu argumentů od uživatele (tedy na délce dotazu). Rychlost je závislá také na celkové kolekci dokumentů, nad kterou vyhledávání probíhá a na konkrétním hledaném výrazu. Časové hodnoty na následujícím grafu jsou zprůměrované hodnoty z několika nezávislých běhů vyhledávání. Na grafu jde vidět, že pro větší a složitější vstupy délka počítání relevance dokumentů narůstá.

Rychlost je závislá i na objemu dat, nad kterými vyhledávání probíhá. Toto vyhledávání bylo testováno nad více než 200 000 dokumenty (písničkami) s 5000 nejčastějšími slovy vyskytujícími se v těchto dokumentech. V případě, že by počet dokumentů byl menší, resp. větší, výsledky by byly uživateli vráceny dříve, resp. později.

Pro různá slova na vstupu se taktéž liší rychlost. Slovo, které se v objevuje v hodně dokumentech vede k tomu, že se musí hodnotit relevance více dokumentů než v případě například slova, které se v kolekci

dokumentů vůbec nevyskytuje (nebo vůbec neexistuje). V takovém případě totiž není potřeba počítat celkové relevance dokumentů ku tomuto vstupu, jelikož všechny dokumenty budou stejně nerelevantní.



Další část, na kterou bychom se chtěli podívat je rozdíl výsledků dotazů na našich a externích datech. Písně patří mezi jedny z nejkratších textů a navíc mají umělci zvyk používat v textech citoslovce a další slova, která jsou při stemmingu mazána. Zajímavý rozdíl se vytvořil už při předzpracování, kdy externí dataset na 200 000 písních měl celkem stabilně vysoký počet topWords v textech. Náš dataset na 30 000 textech se omezoval na jednotky, maximálně nízké desítky slov. Proto i výpočet relavance a následné vyhledávání přinášelo kvalitativně horší výsledky na spodních příčkách vyhledaných než u předzpracovaných dat. Na druhou stranu byla práce s nimi o poznání rychlejší a pravděpodobně by pro koncového uživatele bylo toto řešení jednodušší, protože by mu pro každý výraz ukázala jen omezený počet správných výsledků a bylo by tedy méně náročné danou píseň vybrat.

Diskuze

Rozšířený boolovský model je poměrně užitečný na vyhledávání relevantních dokumentů v statické kolekci, ale má některé nevýhody, které jsou vidět i nad datasetem, se kterým jsme pracovali. První z nich je poměrně velká výpočetní náročnost. Jak je patrné z experimentální sekce, větší vstupy se i pro 200 000 dokumentů (což v světovém měřítku není vlastně vůbec hodně) potýkají s delší dobou celkového vyhledávání. Počítání relevancí jednotlivých dokumentů je poměrně náročné vzhledem k mocninám, odmocninám a logaritmům, na jejichž základě funguje tf-idf model pro hodnocení dokumentů, který jsme pro tuto práci používali.

Druhým problémem je právě statičnost datasetu, nad kterým se vyhledává. Jakékoliv přidání dokumentu by znamenalo muset celý preprocessing spustit znovu pro všechna data, což by při přidávání dokumentů do většího počtu mohlo trvat až zbytečně moc času pro získání pár dalších dokumentů. Především pak, pokud by k přidávání dalších dokumentů docházelo často. Abychom se tomuto vyhnuli, bylo by potřeba přidávat nové dokumenty stylem, který by upravil pouze nutné dokumenty a relevance, nikoliv všechny. I tak by byl ale celý proces poměrně výpočetně náročný a čím častěji by se prováděl, tím méně praktická by byla volba právě tohoto vyhledávacího modelu.

Závěr

Na tomto projektu jsme si vyzkoušeli implementaci vyhledávací aplikace pro statickou kolekci dokumentů. Zjistili jsme, co všechno taková aplikace obnáší a získali jsme náhled do toho, jak je možné jednotlivé dokumenty hodnotit na základě jejich obsahu (textu) vůči ostatním. Jelikož jsme se s programovacím jazykem, ve kterém je většina aplikace psaná (Python) oba teprve učili, proběhlo během celého vývoje aplikace poměrně hodně změn v kódu a jeho funkčnosti, ať už šlo o zjednodušování složitých konstrukcí, zrychlování kódu (kvůli původnímu špatnému návrhu trvalo vyhledávání 1 argumentu více jak 30 sekund, což se s postupem času a četnými úpravami povedlo stáhnout na cca 0.01 sekundy, tedy na přijatelnou hodnotu pro vyhledávání jednoduchého dotazu). Celkově nám však časté upravování kódu a přemýšlení nad konceptem rozšířeného boolovského modelu během celého semestru osvětlilo koncept za vyhledáváním dokumentů na základě obsahu. S výsledky vyhledávání naší aplikace jsme spokojeni, při našem osobním testování doporučovaných dokumentů (což vzhledem k tomu, že se jedná o písničky je poměrně jednoduché ověřit – stačí online vyhledat lyrics a ověřit si relevanci hledaných slov oproti danému songu) jsme nenarazili na žádný problém v případě vypsaných dokumentů.