FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Západočeská univerzita v Plzni

Vyhledávání informací

KIV/IR

Dokumentace semestrální práce

Vojtěch Danišík
A19N0028P
danisik@students.zcu.cz

June 1, 2020

Contents

1 Zadání

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
- Vyhledávání s pomocí logických operátorů AND, OR, NOT, podpora závorek pro vynucení priority operátorů
- Možnost zaindexování poskytnutých dat a dat stažených na cvičení (1. cvičení Crawler)

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),
- Implementace dalšího modelu (použití sémantických prostorů) (1-?b),
- Atd. (xb)

2 Analýza

2.1 Invertovaný index

Invertovaný index je datová struktura používaná pro fulltextové vyhledávání v rozsáhlých kolekcích dokumentů. Obvykle se jedná o seznam významných slov (nejsou součástí množiny stop slov), upravených pomocí stematizace (nalezení kmene slova) a lemmarizace (nalezení základního tvaru slova). Ke každému takovému slovu je přiřazen seznam dokumentů, ve kterých se slovo vyskytuje. Účelem invertovaného indexu je umožnit rychlé vyhledávání za cenu dodatečných operací v okamžiku přidání dokumentu do databáze. Příklad invertovaného indexu lze vidět na obrázku 2.

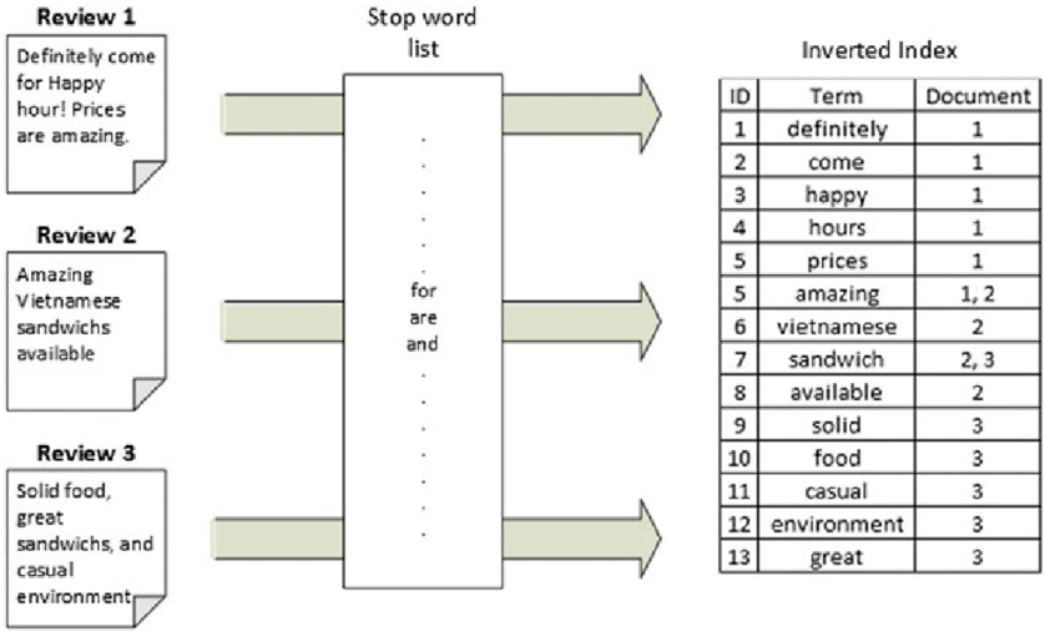


Figure 1: Hlavní okno

2.2 TF-IDF

TF-IDF, neboli **T**erm **F**requency - **I**nverse **D**ocument **F**requency, je numerické číslo, které reprezentuje důležitost slova pro daný dokument v kolekci nebo v korpusu. Při výpočtu hodnoty TF-IDF se používají 2 hodnoty:

- TF - Term Frequency- četnost slova v dokumentu.
- IDF - Inverse Document Frequency - Převrácená četnost slova ve všech dokumentech (vzoreček viz 1, kde N je počet všech dokumentů v korpusu, d je specifický dokument v seznamu dokumentů D a t je dané slovo v dokumentu d).

$$idf(t, D) = log_{10} \frac{N}{|d \in D : t \in d|} \tag{1}$$

Výslednou hodnotu TF-IDF lze vypočítat pomocí vzorečku 2.

$$tfidf(t, d, D) = log_{10}(1 + tf(t, d)) \cdot idf(t, D) \tag{2}$$

TF-IDF je v dnešní době nejčastěji používaná hodnotící metodika u webových vyhledávačů (Google, Seznam , ...) pro zadaný textový řetězec. Pro výpočet TF-IDF hodnot lze použít více variant (více sofistikované metody jsou pouze upravené varianty základního modelu).

2.3 Vector Space Model

Vector Space Model je algebraický model, který reprezentuje objekty (např. textové dokumenty) jako vektory reálných hodnot (např. TF-IDF hodnoty). Využívá se hlavně pro filtraci / získávání informací. indexaci a zjišťování relevance objektů.

Ve VSM jsou všechny objekty reprezentovány jako objekty, kdy každá dimenze vektoru odpovídá jednomu slovu. Pokud se dané slovo vyskytuje v dokumentu, je příslušná dimenze nenulová. Příklad vektoru dokumentu viz příklad 3.

$$\begin{aligned} d_\mathrm{j} &= (w_{1,\mathrm{j}}, w_{2,\mathrm{j}}, ..., w_{\mathrm{t,j}}) \\ q &= (w_{1,\mathrm{q}}, w_{2,\mathrm{q}}, ..., w_{\mathrm{n,q}}) \end{aligned} \tag{3}$$

Pro zjištění přesnosti mezi dvěma dokumenty se používá Kosinova podobnost (Cosine similarity). Kosinova podobnost je metoda pro zjištění hodnoty podobnosti mezi dvěma nenulovými vektory v unitárním prostoru. Vzoreček pro výpočet kosinovy podobnosti viz vzorec 4.

$$cos(d_\mathrm{j}, q) = \frac{d_\mathrm{j} \cdot q}{||d_\mathrm{j}||||q||} = \frac{\sum_{i=1}^{N} w_{\mathrm{i,j}} w_{\mathrm{i,q}}}{\sqrt{\sum_{i=1}^{N} w^2{}_{\mathrm{i,j}}} \sqrt{\sum_{i=1}^{N} w^2{}_{\mathrm{i,q}}}} \tag{4}$$

3 Implementace

3.1 Indexace

3.1.1 Preprocessing

V aplikaci byli použity 2 typy dokumentů:

- **Školní** - Jako data použitá pro preprocessing byly použity - titulek a text dokumentu.
- **Crawlered** - Jako data použitá pro preprocessing byly použity všechny položky dokumentu.

Preprocessing se skládá z několika kroků:

1. **Převedení velkých znaků na malé znaky**
2. **Tokenizace** - Při tokenizaci dat dokumentu se použije speciální regex, který data rozparsuje na jednotlivá slova (případně celé webové stránky). Pokud tato slova se nevyskytují v seznamu stopslov a nejsou jednoznaková, tak se přidají do seznamu slov.
3. **Stematizace** - Všechna slova která prošla tokenizací jsou následně ostemována. Jako použitý stemmer byl využit **CzechStemmerAggresive** od pana Ljiljana Dolamiče. Stemmer se snaží najít nejlepší možný kmen daného slova.
4. **Odstranění diakritiky** - Po stematizaci je ze slova odstraněna veškerá diakritika pomocí speciálního regexu. Po odstranění diakritiky je slovo uloženo do invertovaného indexu, který s ním dále pracuje.

3.1.2 Invertovaný index

Invertovaný index je v aplikaci reprezentován speciální HashMapou, kde klíč je dané slovo a hodnota reprezentuje IDF hodnotu slova a seznam dokumentů obsahující dané slovo.

Každý dokument v sobě obsahuje vlastní slovník, kdy klíč je slovo z dokumentu a hodnota reprezentuje TF-IDF pro dané slovo v daném dokumentu. Zároveň v sobě obsahuje i unikátní indetifikátor na původní dokument a střední hodnotu (která se poté použije při počítání kosinovy podobnosti). Struktura celého invertovaného indexu lze vidět na obrázku 2.

Sestavování invertovaného indexu se skládá ze tří kroků:

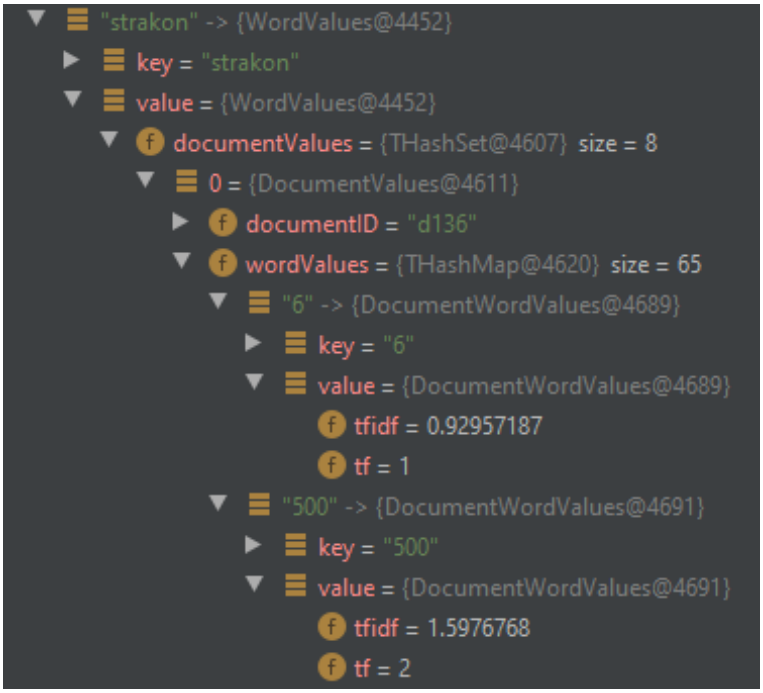


Figure 2: Struktura invertovaného indexu

1. **Vytvoření slovníku** - Vytváření slovníku probíhá při samotném preprocessingu, kdy se již upravené slovo ukládá jak do hlavního slovníku, tak i do slovníku aktuálně zpracovávaného dokumentu (objekt, který obsahuje hodnoty pro indexaci, ne přímo dokument). Zároveň se také uloží reference na tento dokument do seznamu dokumentů pro dané slovo.

2. **Výpočet IDF hodnot** - Po vytvoření slovníku se provede výpočet IDF hodnoty pro všechny slova v hlavním slovníku pomocí vzorečku 1.

3. **Výpočet TF-IDF hodnot** - Po vypočítání IDF hodnot pro slova ve slovníku se provede výpočet TF-IDF hodnot pro všechny slova ve slovníku ve všech dokumentech podle vzorečku 2. Zároveň se vypočítává i střední hodnota pro daný dokument.

3.2 Vyhledávání

Pro vyhledávání v dokumentech byly implementovány 2 vyhledávací modely - Vector Space Model a Boolean.

3.2.1 VSM

Pokud si uživatel navolí vyhledávání v dokumentech pomocí Vector Space Modelu, tak se nad zadanou query provede stejný proces jako v případě indexace dokumentů (tokenizace, stematizace, ...). Po zaindexování query se provede výpočet TF-IDF hodnot pro všechny slova v dané query.

Samotné vyhledávání probíha porovnáváním kosinových podobností mezi jednotlivými dokumenty a zadanou query. Před samotným počítáním kosinovy podobnosti jsou vybrány ty dokumenty, který obsahují alespoň jedno slovo ze zadané query. Mezi těmito dokumenty a zadanou query jsou následně vypočítány kosinovy podobnosti (i se skalárním součinem, viz vzorec 4). V aplikaci jsou následně zobrazeny pouze nejvíce relevantní dokumenty k zadané query (počet nejrelevantnějších dokumentů si uživatel vybírá sám).

3.2.2 Boolean

Pokud si uživatel zvolí Boolean vyhledávání, je očekávána validní query. Lze používat operátory **AND**, **OR** a **NOT**.

Pro parsování boolean query byla použita třída **PrecedenceQueryParser** z **Lucene** knihovny. Bohužel se zde naskytl jeden veliký problém - dotazy obsahující *... AND NOT ...* nebo *... OR NOT ...* byly špatně zpracovány. Proto ještě před samotným použitím parseru se daná query musela upravit tak, aby zadaný dotaz byl dobře zpracovatelný. To zahrnovalo obalení daného operátoru **NOT** společně s následující klauzulí do jednoduchých závorek. Po této úpravě již parser neměl sebemenší problém správně rozparsovat danou query.

Samotné vyhledávání probíhá následovně:

1. **Úprava dotazu** - Samotná úprava spočívá v obalení **NOT** operátoru a klauzule do jednoduchých závorek. Tato úprava funguje tak, že daný dotaz rozparsuje na jednotlivé znaky pomocí mezer. Následně se tyto tokeny kontrolují na prezenci tokenu **NOT**, a pokud se takový token najde. Pokud se za operátorem **NOT** nevyskytuje žádná závorka, je obalení do závorek jednoduché. Pokud se zde objevuje závorka, tudíž klauzule, tak je provedeno rekurzivní volání tatéž metody, která provede následující část query.

2. **Parsování dotazu** - Parsování se provede pomocí **PrecedenceQueryParser**, který daný dotaz rozparsuje (vynucuje prioritu přes závorky a upřednostňuje AND před OR).

3. **Zpracování dotazu** - Zpracování dotazu probíhá rekurzivně, kdy se daný dotaz rozparsuje na klauzule, které jsou následně dále zpra-

covávány. Pokud je klauzule pouze **Term**, tak se vrátí seznam dokumentů obsahující daný term. Pokud klauzule obsahuje podklauzule, je provedeno další rekurzivní volání pro všechny podklauzule. Po zpracování všech klauzulí probíhá vyhodnocování operátorů. Jeden výsledek je roven ID dokumentu.

- *AND* - Nad výsledky dvou **Term**ů je proveden průnik hodnot (ID dokumentů).
- *OR* - Výsledky dvou **Term**ů se sloučí.
- *NOT* - Provede se negace výsledků - Ze všech zaindexovaných dokumentů se vyloučí vrácený výsledky.

3.3 Nadstandardní funkčnost

V aplikaci jsou implementovány 3 nadstandardní funkčnosti.

3.3.1 File-based index

Aplikace využívá načítání indexu ze souboru. Pokud tento soubor s indexovanými daty neexistuje, je provedena indexace a ta se následně uloží do souboru.

Ukládání je implementováno pomocí interface s názvem **Serializable**. Třída, která má být uložena musí implementovat toto rozhraní a zároveň definovat speciální identifikátor. Zároveň všechny objekty (vlastní objekty) musí taktéž implementovat toto rozhraní a definovat speciální identifikátor.

Načítání a ukládání do souboru je zařízeno pomocí tříd **FileInputStream / FileOutputStream**, **BufferedInputStream / BufferedOutputStream** s 16MB velikostí bufferu a třídou **ObjectInputStream / ObjectOutputStream**.

3.3.2 GUI

Pro aplikaci bylo navrženo GUI (vzhled viz Uživatelská příručka). Pro implementaci byla použita *JavaFX* a designový návrhář *SceneBuilder*.

3.3.3 Zvýraznění hledaného textu v náhledu výsledků

Při zvýrazňování textu je daný text nejdříve rozparsován na tokeny pomocí mezer. Každý token je uložen ve 3 proměnných: první obsahuje původní obsah tokenu, druhý obsahuje token bez čárek (pokud je token obsahoval), třetí je původní token, který byl v preprocessingu a také mu byli odstraněny

čárky (pokud je token obsahoval). Následně je třetí token, který byl v preprocessingu, porovnán s tokeny v dané query (při indexování query jsou všechny slova, která byla zaindexována, uložena do speciálního listu). Pokud se dané slovo nachází v query, tak je token bez čárek vložen do čtverečku s červenou barvou. V případě, že daný token obsahoval čárku, je mu na konci přidána zpět (ale už není obarvena).

4 Uživatelská příručka

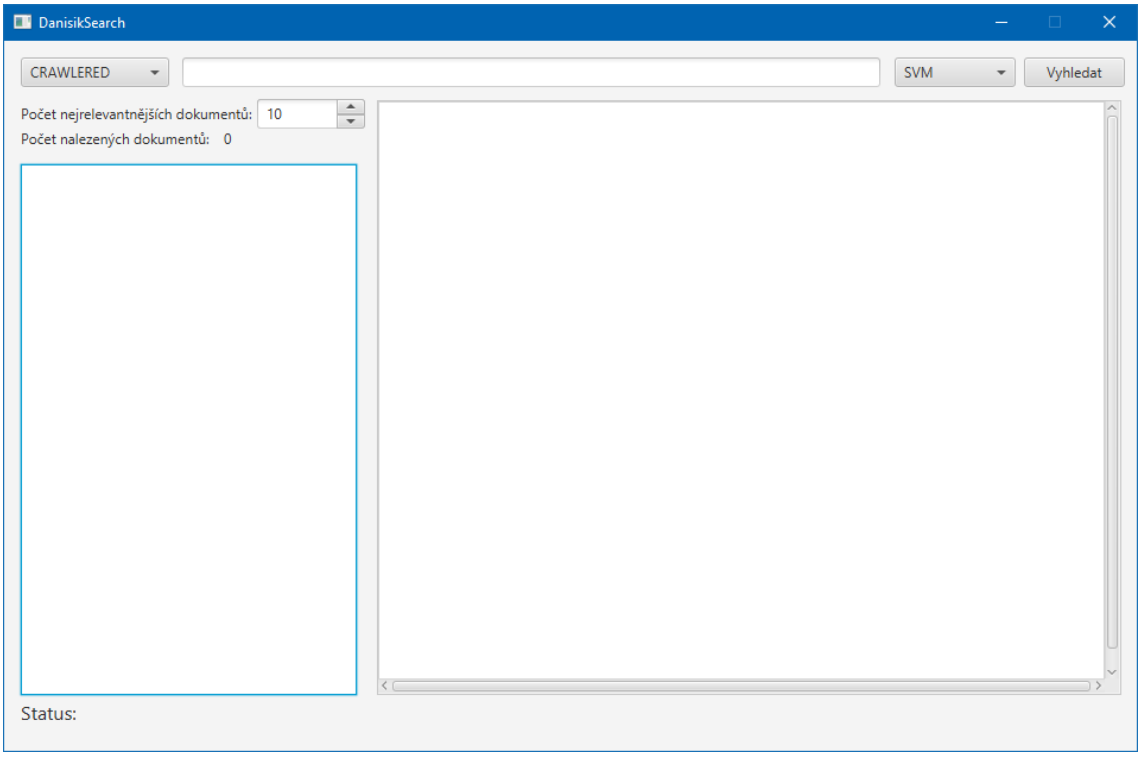


Figure 3: Hlavní okno

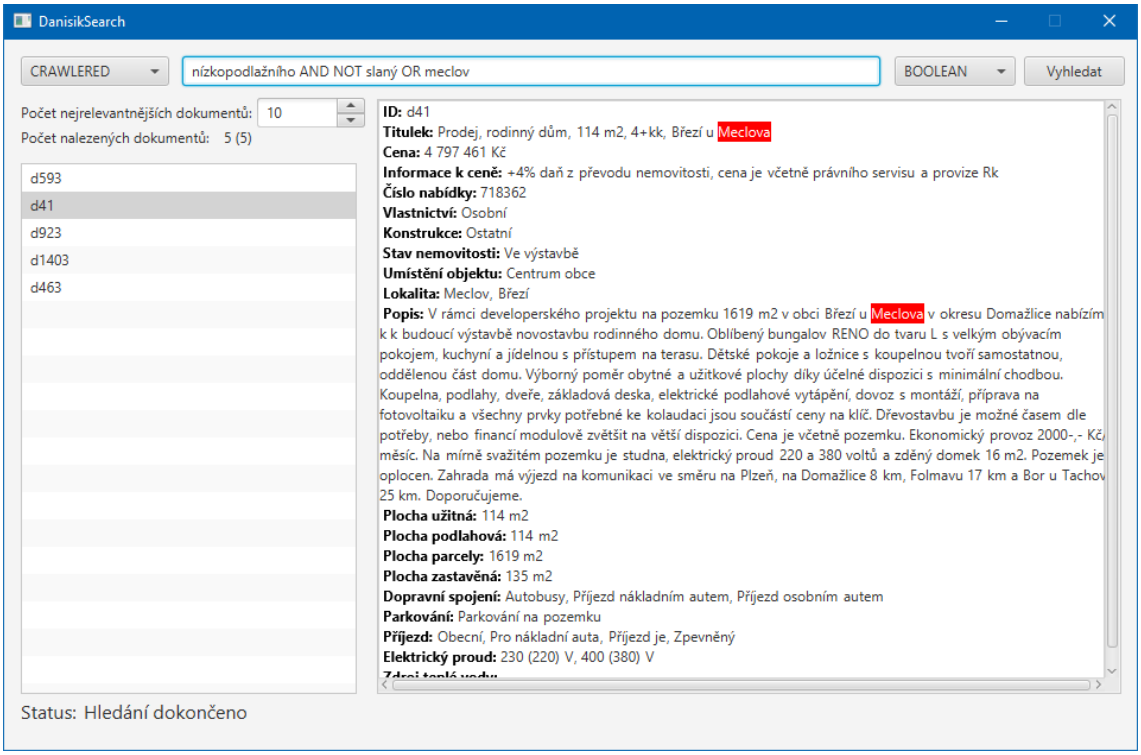


Figure 4: Boolean vyhledávání

5 Závěr

Použitá pole	MAP
Title + Description + Narrative	0.2289
Title + Description	0.1805
Title + Narrative	0.2172
Description + Narrative	0.2065

Table 1: MAP pro kombinace polí v query

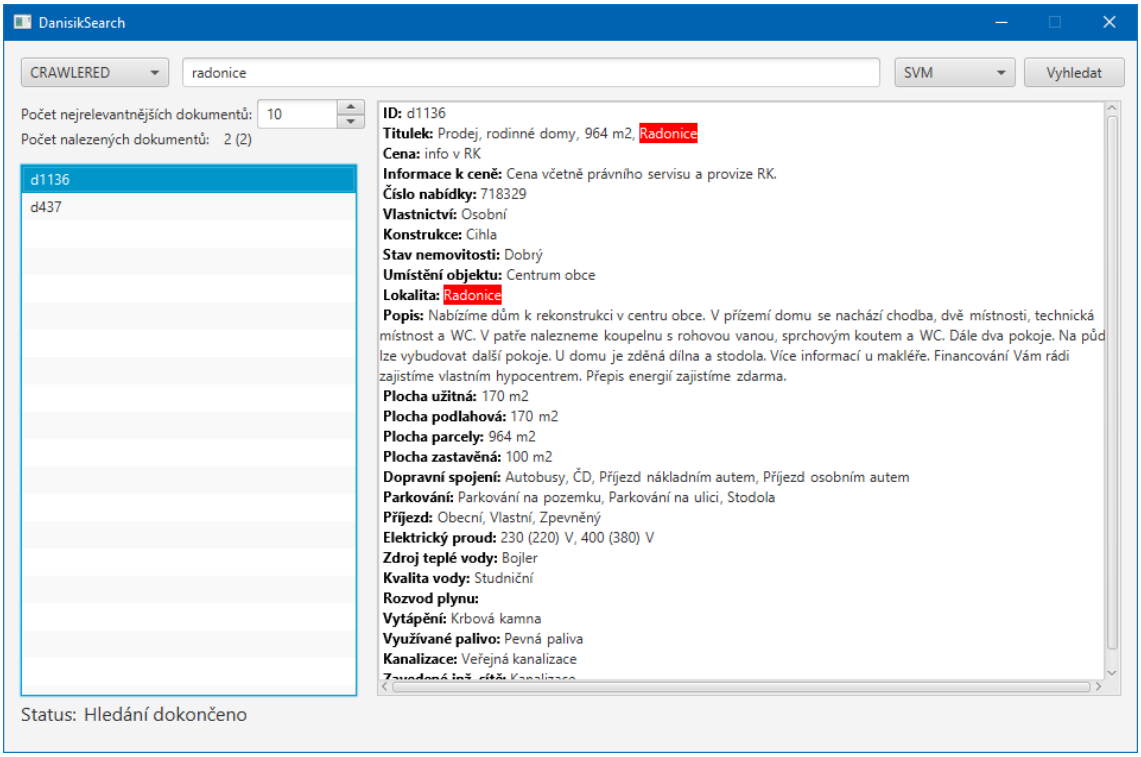
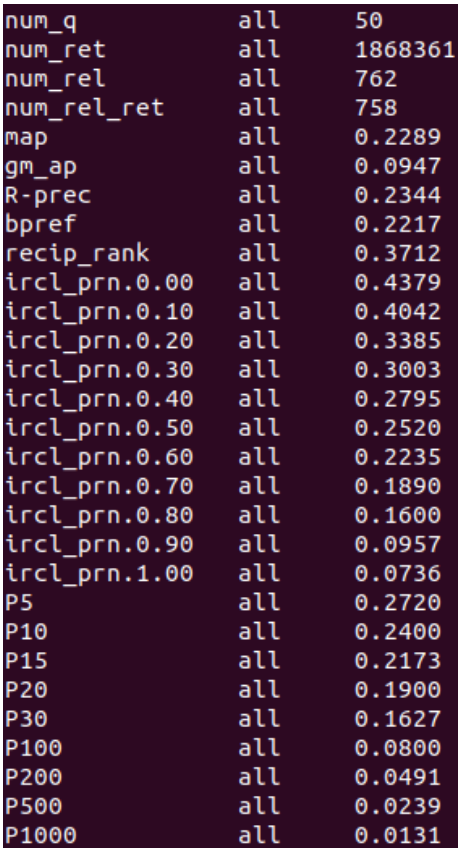


Figure 5: SVM vyhledávání



```
num_q             all     50
num_ret           all     1868361
num_rel           all     762
num_rel_ret       all     758
map               all     0.2289
gm_ap             all     0.0947
R-prec            all     0.2344
bpref             all     0.2217
recip_rank        all     0.3712
ircl_prn.0.00     all     0.4379
ircl_prn.0.10     all     0.4042
ircl_prn.0.20     all     0.3385
ircl_prn.0.30     all     0.3003
ircl_prn.0.40     all     0.2795
ircl_prn.0.50     all     0.2520
ircl_prn.0.60     all     0.2235
ircl_prn.0.70     all     0.1890
ircl_prn.0.80     all     0.1600
ircl_prn.0.90     all     0.0957
ircl_prn.1.00     all     0.0736
P5                all     0.2720
P10               all     0.2400
P15               all     0.2173
P20               all     0.1900
P30               all     0.1627
P100              all     0.0800
P200              all     0.0491
P500              all     0.0239
P1000             all     0.0131
```

Figure 6: Výsledek testu