UNIVERZITA PALACKÉHO V OLOMOUCI
KATEDRA INFORMATIKY

Lukáš Havrlant

Vyhledávač založený na FCA

Soutěžní práce O cenu děkana 2012

Duben 2012

Jméno vedoucího: Mgr. Jan Outrata, PhD

Abstrakt

Napovídáním souvisejících dotazů může vyhledávač pomoci uživateli rychleji najít dokumenty, které potřebuje. Práce se zabývá tvorbou vyhledávače s webovým rozhraním, který pracuje nad uzavřenou sadou dokumentů. Po položení dotazu dokáže napovědět konkrétnější, obecnější a podobný dotaz, což je realizováno pomocí formální konceptuální analýzy.

1. Úvod

Současné vyhledávače si uchovávají obsah, nad kterým mají vyhledávat, ve formě indexu, což je struktura v principu podobná indexu v knize. Uživateli stačí vložit do vyhledávacího pole svůj dotaz a vyhledávač z indexu získá potřebná data a zobrazí uživateli výsledek, obvykle ve formě nějakého uspořádaného seznamu odkazů na dokumenty. Pokud není uživatel spokojený s výsledky, musí přeformulovat svůj dotaz tak, aby lépe vystihoval to, co chce najít.

Tento problém je typický pro slova, která mají několik významů. Například pokud ve webovém vyhledávači vyhledáme slovo „jaguár“, tak vyhledávač nemůže vědět, zda chceme hledat auto, zvíře nebo ještě něco jiného. Pokud má přístup k historii hledání daného uživatele, může pomocí ní přizpůsobit výsledky. Ale může také uživateli zobrazit návrhy na nový dotaz, například „jaguár auto“ nebo „jaguár zvíře“, po jejichž vyhledání se výsledky velmi zpřesní.

Otázkou je, jak tyto návrhy získat. Pokud si vyhledávač uchovává historii všech hledání, která uživatelé provádějí, může se je pokusit vytáhnout právě z této historie. Pokud ovšem tato data vyhledávač nemá, nebo jich má málo, musí se použít jiná metoda.

Práce popisuje metodu, jak získat podobné dotazy pomocí formální konceptuální analýzy (anglicky Formal concept analysis, dále jen FCA) pouze na základně znalosti obsahu dokumentů. FCA pracuje s tabulkovými daty, ve kterých hledá nějaké potenciálně zajímavé shluky dat. Tyto shluky pro nás budou představovat množiny dokumentů, které jsou nějakým způsobem podobné a které sdílí nějaká zásadní klíčová slova. Tato klíčová slova poté dále využijeme při generování návrhů na nové dotazy.

2. Information Retrieval

Tato sekce se zabývá budováním samotného vyhledávače, jehož výsledky budou zobrazeny uživateli a také budou použity jako vstup do formální konceptuální analýzy.

Cílem je popsat tvorbu vyhledávače, který bude umět stáhnout z webu požadované dokumenty, zaindexovat je a následně v nich vyhledávat. Dokumenty mohou být buď obyčejné webové stránky nebo složitější soubory, například PDF.

2.1. Jak funguje obecný vyhledávač

Moderní vyhledávače využívají index. Jedná se o podobnou strukturu, kterou můžeme najít v některých, obzvláště odborných, knihách. Protože v klasické tištěné knize nemůžeme nijak „vyhledávat“, dává se na konec knihy seznam nejdůležitějších slov, která kniha obsahuje, spolu s čísly stránek, na kterých se pojem vyskytuje. Takže chce-li uživatel nalézt stránky obsahující slovo „derivace“, podívá se do indexu, kde hned zjistí, že slovo se vyskytuje na té a té stránce.

Podobný princip můžeme použít i ve vyhledávačích. Nebudeme ale vytvářet index z důležitých slov, ale ze všech slov, která se v dokumentech vyskytují. Vytvoříme tak slovníkovou strukturu, kde klíčem bude slovo a hodnotou bude seznam dokumentů, které dané slovo obsahují. Při položení dotazu pak může vyhledávač rychle zjistit jaké dokumenty obsahují dané klíčové slovo prostým nahlédnutím do tohoto slovníku.

Spolu s tím si můžeme uložit informace i o tom, kolik daných slov daný dokument obsahuje, což můžeme dále využít při řazení dokumentů.

Celou práci vyhledávače tak můžeme rozdělit do dvou částí – budování a upravování indexu a hledání odpovědí na dotaz. Zde je nutné dát pozor na to, jak velká je sada dokumentů a zda je statická nebo dynamická. Samotné budování indexu můžeme ještě rozdělit na dvě části: předzpracování dokumentů a technická realizace indexu.

2.2. Vyhledávač CLaSeek

Součástí této diplomové práce je naprogramovaný vyhledávač CLaSeek (Concept Lattice Seeker), který bude v dalších částech textu popsán. CLaSeek je vyhledávač napsaný v Pythonu 3. Pracuje se statickou sadou dokumentů; předpokládá se, že se sada dokumentů bude měnit pouze nárazově jednou za čas. Samotná sada dokumentů, nad kterou má vyhledávač pracovat, nebude příliš velká, řádově stovky dokumentů.

Jedná se o vyhledávač používající index. Při budování indexu je vstupem seznam URL adres, který si vyhledávač sám stáhne a zaindexuje obsahy dokumentů. Výsledný index je pak uložen do několika souborů, ke kterým pak CLaSeek přistupuje.

CLaSeek má rozumět logickým operátorům AND, OR a NOT. Po zadání dotazu má vrátit výsledky seřazené podle relevance, kterou spočítá pomocí klasického tf-idf algoritmu. Výstup bude textový ve formátu JSON, se kterým pak mohou pracovat další programy, v tomto případě webové rozhraní, které je napsáno v PHP.

Další částí vyhledávače je hledání souvisejících dotazů. Tato část bude podrobně rozebrána v další kapitole. V této kapitole je dále popsáno, jak CLaSeek buduje index a jak vrací výsledky.

2.3. Předzpracování dokumentů

Při budování indexu máme na vstupu sadu dokumentů a na výstupu strukturu, která reprezentuje index této sady. Během samotného budování indexu procházíme jednotlivé dokumenty a upravujeme je do takové podoby, která se hodí pro uložení. Budeme uvažovat pouze textové dokumenty jako je HTML nebo PDF a budeme předpokládat statickou sadu dokumentů, takže se nebudeme příliš zatěžovat aktualizací indexu.

Na samotném začátku tak musíme upravit jednotlivé textové dokumenty do nějaké kanonické podoby. To budeme dělat postupnými úpravami jednotlivých dokumentů. S každým dokumentem

budeme provádět identické operace v identickém pořadí. Jednotlivé operace budou popsány v takovém pořadí, v jakém se aplikují ve vyhledávači.

Odstranění formátovacích prvků dokumentu jako například HTML značky.

Ponechání písmen Tj. odstranění interpunkce a jiných zbytečných znaků.

Odstranění bílých znaků Odstraníme zdvojené mezery, nové řádky apod.

Převod na malá písmena Velikost písmen nehraje u vyhledávání žádnou roli.

Odstranění stop slov jako například „ke“, „u“, „na“ a podobně.

Převod na stemy Stem je kořen, základ slova. Smyslem je, abychom si v indexu neuchovávali všechny tvary každého slova, ale abychom si od každé slova uchovávali ideálně jen jeden, základní tvar – místo „strom“, „stromy“, „stromu“ tak budeme mít pouze jeden tvar „strom“.

Odstranění diakritiky

2.4. Vyhledání atributů dokumentu

V druhé části vyhledávače, která se zabývá nalezením souvisejících dotazů, budeme potřebovat znát množinu slov, která nejvíce charakterizuje daný dokument. Těmto slovům budeme říkat „atributy dokumentu“.

2.4.1. Jak by měly atributy vypadat

Atributy jsou poměrně běžnou součástí různých vědeckých článků, kde je ale obvykle vyplňuje sám autor pod názvem „klíčová slova“. V případě vyhledávače stojíme před problémem, jak získat atributy z libovolného dokumentu nějakým obecným způsobem.

Jednoduchým způsobem je seřazení všech stemů v daném dokumentu podle jejich četnosti od nejvíce častého. Za atributy pak můžeme vzít ty stemy, které překročí nějakou absolutní hranici („alespoň 10 výskytů v dokumentu“) nebo nějakou relativní hranici („poměr počtu daného stemu ku počtu všech stemů v dokumentu je větší než 0,05“).

Tento postup může být úspěšný v případě, kdy máme pouze jeden dokument. Pokud ale máme sadu dokumentů, můžeme ještě zjistit vztah s dalšími dokumenty. Atribut pro daný dokument by totiž mělo být takové slovo, které je mezi ostatními dokumenty co možná nejvíce unikátní.

Pokud máme sadu dokumentů, která se zabývá analýzou dat, je možné, že by v každém dokumentu bylo nejčastější slovo právě „analýza“. Tím bychom dostali pro každý dokument stejný atribut a to není to, co bychom chtěli.

Tento problém vyřešíme tím, že při hledání atributů pro dokument vezmeme v potaz i to, jak často se daná slova vyskytují v ostatních dokumentech. Budeme tak hledat taková slova, která se v daném dokumentu vyskytují co nejčastěji a v ostatních dokumentech co nejméně často.

2.4.2. Algoritmus tf-idf

Tento postup má své jméno, jedná se o tf-idf algoritmus. Ten je rozdělený do několika částí. První je funkce $\mathrm{tf}_{t,d}$, která v základním nastavení vrací počet výskytů slova t v dokumentu d. Dále máme funkci df_t, která vrací počet dokumentů, které obsahují slovo t. Tuto funkci využijeme k tomu, abychom snížili skóre těch slov, která se vyskytují v příliš mnoha dokumentech.

Označme N počet všech dokumentů v naší sadě. Pak vydělením N/df_t získáme koeficient, který značí, jak moc je slovo t unikátní. Pokud se vyskytuje jen v jednom dokumentu, získáme maximální hodnotu N. Při vyšším výskytu slov v dokumentech by tento koeficient klesal příliš rychle, proto ještě použijeme logaritmus. Získáme funkci idf_t

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}.$$

Složením funkcí tf a idf získáme funkci tf-idf′ (za chvíli tuto funkci ještě vylepšíme, prozatím si ji označíme s apostrofem) definovanou jako

$$\text{tf-idf}'_{t,d} = \text{tf}_{t,d} \cdot \text{idf}_t.$$

Tato funkce přiřazuje slovu t a dokumentu d hodnotu, která je

- vysoká, pokud se slovo t často vyskytuje v malé množině dokumentů,
- nízká, pokud se slovo vyskytuje málo v dokumentu d nebo pokud se vyskytuje hodně v ostatních dokumentech.

2.4.3. Vylepšení algoritmu tf-idf

Problémem tohoto přístupu je, že příliš preferuje velké dokumenty, které obsahují mnoho slov. Máme-li například učebnici středoškolské matematiky, je pravděpodobné, že bude několikrát, řekněme 50krát, obsahovat slovo „kombinace“. Vedle toho můžeme mít desetistránkový dokument pojednávající čistě o kombinacích, ale slovo „kombinace“ obsahuje pouze 25krát. Podle stávající $\text{tf}_{t,d}$ funkce bude učebnice na klíčové slovo „kombinace“ dvakrát relevantnější než článek přímo se zaměřující na kombinace.

Tento problém můžeme zkusit vyřešit tím, že hodnotu $\text{tf}_{t,d}$ ještě vydělíme celkovým počtem slov v dokumentu. Získáme tak relativní zastoupení slova t mezi všemi slovy v dokumentu. Učebnice z předchozího příkladu pak bude mít mnohem nižší hodnotu $\text{tf}_{t,d}$, protože je mnohonásobně větší než článek. Vzorec funkce tf-idf by pak vypadal takto:

$$\text{tf-idf}''_{t,d} = \frac{\text{tf}_{t,d}}{|d|} \cdot \text{idf}_t,$$

kde $|d|$ je počet slov v dokumentu d.

Problém jsme tím ale ve skutečnosti nevyřešili, jen jsme ho obrátili – už nejsou preferované velké dokumenty, ale malé dokumenty, které dané klíčové slovo obsahují. Pokud bychom zpracovali dokument, jehož obsahem by byla pouze věta „Sázky a kurzy na severskou kombinaci.“, pak by hodnota $\text{tf}_{t,d}$ pro slovo „kombinace“ byla $\frac{1}{4}$ – v dokumentu jsou čtyři slova (slova „a“ a „na“ nepočítáme, jsou to stop slova) a jedno z nich je právě „kombinace“ (po převedení na stem). Pokud by měl mít zmíněný článek, který obsahuje 25krát slovo „kombinace“, alespoň stejnou hodnotu $\text{tf}_{t,d}$, nesměl by mít více než sto slov.

CLaSeek nakonec hodnotu $\text{tf}_{t,d}$ ještě dělí logaritmem počtu všech slov v dokumentu. Hodnota $\text{tf}_{t,d}$ u velkých dokumentů tak bude vydělena větší hodnotou než u malých dokumentů. Zároveň ale tato hodnota, kterou dělíme, nebude růst lineárně, takže desetkrát větší dokument nepotřebuje i desetkrát více klíčových slov, aby dosáhl na stejné hodnocení.

Nyní můžeme napsat finální verzi funkce tf-idf:

$$\text{tf-idf}_{t,d} = \frac{\text{tf}_{t,d}}{\log |d|} \cdot \text{idf}_t,$$

kde $|d|$ je počet slov v dokumentu d.

2.5. Inverzní stemovací funkce

Během zpracování textů jsme používali stemmer, který bral na vstup slovo a na výstupu vrátil základ slova. Bohužel pro poměrně velkou část slov vrací funkce takový základ slova, který je sám o sobě nesmyslný.

Například pro slovo „množina“ získáme s použitým stemmerem stem „mnoh“. Toto chování nám nevadí v případě budování indexu, ale vadilo by nám v druhé části. Nemůžeme uživateli napovědět, že má k dotazu přidat klíčové slovo „mnoh“, protože nebude vědět, jaké slovo ve skutečnosti přidává.

Vyhledávač vylepšíme tím, že se pokusíme najít inverzní funkci ke stemovací funkci, abychom mohli ke každému podobně špatnému stemu přiřadit nějaké reálné slovo. Bohužel inverzní funkce neexistuje, protože stemovací funkce není prostá. Sestrojíme takovou funkci, která bude mít na vstupu stem s a na výstupu jedno ze slov, které má s jako svůj stem. Mějme tak původní stemmer, který označíme jako funkci stem, která nám pro slovo vrací jeho stem. Budeme chtít sestavit funkci stem^{-1}, která nám pro stem vrací nějaké přirozené slovo.

Při používání funkce stem si tak u každého výsledného stemu s budeme uchovávat množinu původních slov P_s, která se zobrazují právě na stem s. Tedy pokud $s = stem(w)$, pak $P_s = P_s \cup \{w\}$. Množina P_s pak splňuje $\forall x \in P_s : \text{stem}(x) = s$.

Zároveň si uložíme počet jednotlivých slov ve všech dokumentech. K tomu účelu definujeme funkci $\text{sf}(w)$, která vrací počet výskytů slova w ve všech dokumentech. Budeme chtít, aby nám funkce stem^{-1} vrátila z množiny P_s takové slovo, které se v původní sadě dokumentů vyskytovalo nejčastěji.

Nejprve zadefinujeme pomocnou funkci stem^{-1}_{max}:

$$\text{stem}^{-1}_{max}(s) = \left\{ x \in P_s \mid \text{sf}(x) = \max_{y \in P_s} \text{sf}(y) \right\}.$$

Tato funkce nám vrátí množinu všech slov, která se zobrazují na stem s a jejichž počet výskytů v sadě dokumentů je shodný, ale maximální mezi všemi slovy z P_s. Funkce stem^{-1} může z této množiny vrátit libovolný prvek. Aby byla funkce jednoznačně definována, vrátí takové slovo, které je nejmenší vzhledem k lexikografickému uspořádání:

$$\text{stem}^{-1}(s) = w \quad \Leftrightarrow \quad w = \min_{x \in \text{stem}^{-1}_{max}(s)} x.$$

Pokud nyní použijeme funkci stem^{-1} například na stem „mnoh“, měla by vrátit slovo „množina“, protože to je slovo, které má stem „mnoh“ a pravděpodobně se v dokumentech vyskytuje nejčastěji.

2.6. Odpovídání na dotazy

Hlavním účelem vyhledávače je samozřejmě odpovídání na dotazy. Uživatel vloží do rozhraní vyhledávače svůj dotaz, který je dále zpracován vyhledávačem, který vrátí nějaký seznam výsledků.

2.6.1. Syntax dotazu

Dotazem může být libovolný text, přičemž může obsahovat jisté řídicí příkazy, kterým říkáme operátory. CLaSeek podporuje tři operátory: AND, OR a NOT. Musí být psány velkými písmeny, takže pokud chce uživatel použít daná slova jako obyčejný text, stačí použít jejich variantu s malými písmeny.

Pokud nevložíme do textu žádný operátor, použije se implicitně mezi každým slovem AND operátor. Spojíme-li dvě slova operátorem OR, budou se hledat dokumenty, které obsahují alespoň jedno z těchto slov. Operátor NOT vylučuje ty dokumenty, které obsahují slovo, které se nachází za NOT.

Tyto operátory můžeme různě kombinovat a s pomocí závorek můžeme tvořit složitější dotazy, například „(spočetné OR nespočetné) množiny NOT (komplexní OR přirozená)“.

2.6.2. Vyhledání odpovídajících dokumentů

Ve chvíli, kdy máme syntaktický strom, se můžeme pustit do hledání dokumentů. K tomu využijeme vybudovaný index, který nám umožňuje snadno zjistit, v jakých dokumentech se vyskytuje dané slovo. Pro účely popisu mechanismu si definujeme funkci R, která bere na vstupu dotaz (syntaktický strom) Q a na výstupu vrací množinu dokumentů, které odpovídají zadanému dotazu. Tato funkce se bude chovat odlišně v závislosti na tom, jak vypadá dotaz Q.

Dále označme $\mathbb{D}$ množinu všech dokumentů a α a β nechť označují nějaké dotazy.

```
1: function RANK(d, S)
2:     score ← sc_{d,S}
3:     for all s ∈ S do
4:         if s ∈ title(d)  ∨  s ∈ url(d) then
5:             score ← score · 3
6:         end if
7:         if s ∈ description(d) then
8:             score ← score · 2
9:         end if
10:    end for
11:    return score
12: end function
```

Obrázek 1. Algoritmus rank

- Je-li $Q = s$, kde s je nějaký stem, pak funkce R vrátí množinu dokumentů, které obsahují daný stem, což vyčteme z indexu. Formálně to můžeme zapsat jako:

$$\mathrm{R}_s = \{d \in \mathbb{D} \mid s \in d\}\,.$$

- Je-li dotaz ve tvaru Q = „α AND β“, pak $\mathrm{R}_Q = \mathrm{R}_\alpha \cap \mathrm{R}_\beta$.
- Je-li dotaz ve tvaru Q = „α OR β“, pak $\mathrm{R}_Q = \mathrm{R}_\alpha \cup \mathrm{R}_\beta$.
- Je-li dotaz ve tvaru Q = „NOT α“, pak $\mathrm{R}_Q = \mathbb{D} \setminus \mathrm{R}_\alpha$.

Postupnou aplikací těchto pravidel dostaneme množinu výsledných dokumentů R. V dalším kroku tyto dokumenty seřadíme podle relevance.

2.6.3. Seřazení dokumentů

V současné chvíli máme množinu dokumentů R_Q. Aby byl vyhledávač smysluplný, měl by tyto dokumenty seřadit podle toho, jak relevantní dané dokumenty vzhledem k položenému dotazu jsou. To je obecně nelehký úkol. Ve vyhledávači je pak použit standardní tf-idf algoritmus popsaný v části 2.4.2.

Abychom mohli ohodnotit jednotlivé dokumenty, potřebujeme znát slova, vůči kterým máme dokumenty ohodnocovat. Odstraníme tak z dotazu všechny operátory a dostaneme množinu všech slov S. Vůči těmto slovům budeme dokumenty hodnotit.

K tomu už využijeme tf-idf algoritmus – pro každé slovo z množiny S a pro každý dokument z množiny R_Q spočítáme jeho tf-idf skóre; poté tato skóre sečteme a dokumenty seřadíme sestupně podle tohoto skóre. Skóre $\mathrm{sc}_{d,S}$ dokumentu d tak udává vzorec:

$$\mathrm{sc}_{d,S} = \sum_{s \in S} \text{tf-idf}_{s,d}\,.$$

Toto základní skóre ještě dále upravíme podle toho, zda se klíčová slova z dotazu nevyskytují v důležitých částech dokumentu, konkrétně jde o název, adresu a popisek stránky. Přesný postup ukazuje funkce rank, viz obrázek 1., která nejprve vypočte hodnotu $\mathrm{sc}_{d,S}$ a poté aplikuje pravidla o ztrojnásobení a zdvojnásobení. Funkce rank bere na vstupu dva parametry: dokument d, pro který počítáme skóre a slova S z dotazu Q.

Nyní seřadíme dokumenty R_Q do n-tice $\langle d_1, d_2, \ldots, d_n \rangle$, kde

$$n = |\mathrm{R}_Q| \quad \text{a} \quad \bigcup_{i=1}^{n} d_i = \mathrm{R}_Q$$

tak, aby platilo $\mathrm{rank}_{d_1,S} \geq \mathrm{rank}_{d_2,S} \geq \ldots \geq \mathrm{rank}_{d_n,S}$. Tato seřazená n-tice je výstupem algoritmu vyhledávání.

3. Hledání souvisejících dokumentů

V této kapitole bude popsána druhá část vyhledávače, která se stará o nalezení souvisejících dotazů. Nejprve uveďme stručný úvod do formální konceptuální analýzy.

3.1. Formální zavedení FCA