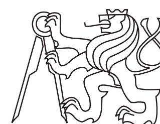
Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze

Fakulta informačních technologií

Katedra softwarového inženýrství



Diplomová práce

Analytické webové služby pro portál Spolupráce s průmyslem

Jiří Maroušek

Vedoucí práce: Ing. Pavel Kordík, Ph.D.

21. června 2015

Prohlášení

Prohlašuji, že jsem tuto práci vytvořil samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některý zákonů (autorský zákon), nemám závažný důvod proti užití tohoto školního díla a k užití uděluji svolení.

V Praze dne 21. června 2015 ....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2015 Jiří Maroušek. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Jiří Maroušek. *Analytické webové služby pro portál Spolupráce s průmyslem: Diplomová práce*. Praha: ČVUT v Praze, Fakulta informačních technologií, 2015.

Abstract

The work deals with the implementation of analytical methods using text mining algorithms, which are designed to facilitate and automate the user activity of faculty Portal collaboration with industry. The second part is the design and implementation of web services that give you access to the outputs of analytical methods. Text thesis deals with the theoretical part used text-mining algorithms, analyzing external data usable for analytical methods, documents the implementation of both parts and labor finally focuses the debate over the quality of results.

Keywords Portal Collaboration with industry, analytical methods, text mining, LSA, web services.

Abstrakt

Práce se zabývá implementací analytických metod využivajících algoritmů text miningu, které mají za úkol usnadnit a zautomatizovat činnost uživatelů fakultního portálu Spolupráce s průmyslem. Druhou částí práce je návrh a implementace webových služeb, které zpřístupňují výstupy analytických metod. Text práce se zabývá teoretickou částí použitých text miningových algoritmů, analýzou externích dat využitelných pro analytické metody, dokumentuje implementaci obou částí práce a na závěr se věnuje diskuzi nad kvalitou výsledků.

Klíčová slova Portál Spolupráce s průmyslem, analytické metody, text mining, LSA, webové služby.

Obsah

Úvod **15**
Portál Spolupráce s průmyslem 15
Cíl práce 15
Struktura textu práce 15

1 Podobnost dokumentů **17**
1.1 Reprezentace dokumentu 17
1.2 Předzpracování dokumentu 19
1.3 Určení podobnosti dokumentů 20
1.4 Latentní sémantická analýza 21

2 Extrakce tagů **27**
2.1 Automatická extrakce tagů 27
2.2 Zvolený přístup 28

3 Analýza využitelných dat **33**
3.1 Základní entity 33
3.2 KOSapi 33
3.3 DBpedia 35
3.4 Portál Spolupráce s průmyslem 37

4 Použité technologie **39**
4.1 JAMA 39
4.2 KDG Commons - MappedFileBuffer 40
4.3 PostgreSQL 41
4.4 Spring Framework 43
4.5 Apache Maven 43
4.6 Apache Tomcat 43

5 Analytické metody **45**
5.1 Úvod 45
5.2 Pojmy 45
5.3 Zpracování dokumentu 46
5.4 Vytvoření Term-by-Document matice 47

5.5 Výpočet LSA 48
5.6 Výpočet relevance dokumentů 50
5.7 Automatická extrakce tagů 51

6 Webové služby **57**
6.1 Úvod 57
6.2 Controllery 57
6.3 Služby relevance 58
6.4 Výpočetní služby 61
6.5 Výjimky 65
6.6 Logování 66

7 Diskuze **69**
7.1 Technika hodnocení výsledků 69
7.2 Automatická extrakce tagů 69
7.3 Výsledky měření 71
7.4 Zhodnocení výsledků 71
7.5 Srovnání 72

Závěr **73**

Literatura **75**

A Seznam použitých zkratek **77**

B Instalace a spuštení **79**
B.1 Nastavení konfiguračního souboru 79
B.2 Databáze 80
B.3 Build aplikace 80
B.4 Spuštení aplikace 80

C Konfigurační soubor **83**

D SQL skript pro vytvoření objektů v databázi **87**

E Obsah přiloženého CD **91**

Seznam obrázků

4.1 EER Diagram databáze 42

Seznam tabulek

2.1 POS tagging - kategorie 29
2.2 POS tagging - slovní druhy 30

7.1 Kontingenční tabulka výsledků automatické extrakce a manuálního přiřazení klíčových slov 70
7.2 Naměřené výsledky 71
7.3 Porovnání modelů automatické extrakce klíčových slov 72

C.1 Konfigurační soubor 83
C.2 Konfigurační soubor 84
C.3 Konfigurační soubor 85

Úvod

Portál Spolupráce s průmyslem

Portál Spolupráce s průmyslem (SSP) funguje na principu spolupráce průmyslových partnerů (podniků) s univerzitním pracovištěm, zejména pak s jeho studenty. Většina studentů již v průběhu studia nějakým způsobem pracuje a získává praxi. Nicméně v převážné většině případů se tato praxe nijak nepromítne do výuky ve škole, i když by v mnoha případech mohla být uznána do prakticky orientovaných předmětů. Nemluvě o závěrečných prací, které se dají realizovat ve spolupráci s podniky.[2]

Cíl práce

Portál tak nabízí možnost podnikům představit studentům zajímavé projekty, které studentům mohou nabídnout jak finanční odměnu, tak možnost uplatnění projektu v rámci studijního předmětu. Důležitým aspektem celého procesu je určení vhodného studenta na daný projekt, tak aby měl v dané problematice určité znalosti. Přesně tento bod procesu je hlavní náplní práce, tj. určit na základě analytických metod vhodné obsazení pro jednotlivé projekty.

Struktura textu práce

V úvodních dvou kapitolách si teoreticky rozebereme techniky použité pro analytické metody. Stručně se podíváme na možná řešení problému a podrobněji si rozebereme námi vybrané řešení. Ve třetí kapitole provedeme analýzu externích dat, která budou figurovat jako vstup pro naše analytické metody. V následujících třech kapitolách se podíváme na použité technologie, implementaci jednotlivých částí práce a v závěrečné kapitole budeme diskutovat nad dosaženými výsledky.

KAPITOLA 1

Podobnost dokumentů

1.1 Reprezentace dokumentu

První důležitou věcí při určování podobnosti (relevance) dokumentů je stanovení formy reprezentace dokumentu. Pro většinu metod se ukázalo jako nejlepší řešení využití reprezentace pomocí vektoru, který má tolik složek kolik je možných termů. Term je slovo v základním tvaru, tj. po použití určité metody pro získání tohoto tvaru (stemování, lemmatizace).

Délka vektoru dokumentu je určena podle počtu všech termů v kolekci dokumentů. Z toho vyplývá, že pravděpodobně bude obsahovat velký počet nul, protože většina termů se v daném dokumentu nebude vyskytovat. Abychom mohli porovnávat dva vektory, musí mít stejnou velikost. Váha jednotlivých složek (termů) je stanovena na základě kombinace lokální a globální metodiky vážení, které jsou uvedeny v sekci 1.1.1, resp. 1.1.2.

Tato reprezentace má své výhody a nevýhody. Bezesporu největší výhodou je invariance vůči pořadí termů a celková jednoduchost převedení dokumentu do vektorové reprezentace. Z nevýhod musíme určitě zmínit, že nevyužíváme struktury dokumentu (např. termy obsažené v nadpisech jsou důležitější, než termy vyskytující se v odstavcích). Tato nevýhoda se dá řešit úpravou váhy daného termu. Další nevýhodou je, že nijak nezachytíme víceslovné fráze. Tuto nevýhodu můžeme odstranit kódováním frází, nebo použitím N-gramů. Poslední nevýhodou je velká dimenze vektorů, která se řeší většinou v předzpracování dokumentů. Určité techniky jí řeší až ve svém průběhu a mnohdy jsou na ní založeny (jako např. LSA). [7]

1.1.1 Lokální metodiky

Lokální metodiky ohodnotí výskyt daného termu v rámci jednoho dokumentu a získáme tak relativní frekvenci termu v dokumentu.

1.1.1.1 Binární

$$l_{ij} = 1, \text{ pokud term } i \text{ existuje v dokumentu } j, \text{ jinak } 0 \tag{1.1}$$

1.1.1.2 Term Frequency (TF)

$$l_{ij} = tf_{ij} \tag{1.2}$$

Kde tf_{ij} je počet výskytů termu i v dokumentu j.

1.1.1.3 Log

$$l_{ij} = Log(1 + tf_{ij}) \tag{1.3}$$

Kde tf_{ij} je počet výskytů termu i v dokumentu j.

1.1.1.4 Augnorm

$$l_{ij} = \frac{(\frac{tf_{ij}}{max_i(tf_{ij})}) + 1}{2} \tag{1.4}$$

Kde tf_{ij} je počet výskytů termu i v dokumentu j.

1.1.2 Globální metodiky

Globální metodiky popisují relativní frekvenci termu napříč celou kolekcí dokumentů.

1.1.2.1 Binární

$$g_i = 1 \tag{1.5}$$

1.1.2.2 Normal

$$g_i = \frac{1}{\sqrt{\sum_j tf_{ij}^2}} \tag{1.6}$$

Kde tf_{ij} je počet výskytů termu i v dokumentu j.

1.1.2.3 GfIdf

$$g_i = \frac{gf_i}{df_i} \tag{1.7}$$

Kde gf_i je počet výskytů termu i v celé kolekci dokumentů a df_i je počet dokumentů, ve kterých se term i vyskytuje.

1.1.2.4 Idf

$$g_i = \log_2 \frac{n+1}{1+df_i} \tag{1.8}$$

Kde df_i je počet dokumentů, ve kterých se term i vyskytuje a n je počet dokumentů v kolekci.

1.1.2.5 Entropy

$$g_i = 1 + \sum_j \frac{p_{ij} \log p_{ij}}{\log n}, \text{ kde } p_{ij} = \frac{tf_{ij}}{gf_i} \tag{1.9}$$

Kde gf_i je počet výskytů termu i v celé kolekci dokumentů, tf_{ij} je počet výskytů termu i v dokumentu j a n je počet dokumentů v kolekci.

1.2 Předzpracování dokumentu

Tento proces je závislý na zvolené reprezentaci dokumentu. Pokud se zde budeme bavit o vektorové reprezentaci dokumentu, pak je hlavním úkolem předzpracování dat snížení dimenze výsledného vektoru. V podstatě se jedná o snížení počtu termů v dokumentu a tudíž v celé kolekci dokumentů. Jak toho dosáhneme?

Nejdříve můžeme odstranit všechny interpunkční znaménka. V druhém kroku můžeme odstranit tzv. *stop words*, což jsou slova, která nám neříkají nic o významu dané věty nebo souvětí. A v neposlední řadě využijeme technik stemování nebo lemmatizace, což jsou techniky, které nám z daného slova vytvoří jeho základní tvar. U stemování je to tzv. stem, což pro představu je něco jako kořen slova, nicméně stemování je většinou založeno na „odtržení“ koncovek, tudíž se o kořen slova tak, jak ho známe z mluvnice, nejedná. Lemmatizace je o něco sofistikovanější, ale je zapotřebí korpusu dokumentů, ze kterého se má možnost lemmatizátor učit. Lemmatizátor vrací tzv. lemma. Pro ukázku srovnání těchto dvou technik využijeme slovo „barvě“. V případě stemování získáme stem „barv“, ten samý získáme např. i ze slova „barva“,

ale nikoli ze slova „barevný“. Naopak v případě lemmatizace získáme ze stejného původního slova „barvě“ lemma ve tvaru „barva“, které získáme i ze slova „barevný“.

Z uvedené ukázky vidíme, že při použití, ať už jedné nebo druhé techniky, dojde ke snížení dimenze. U lemmatizace pak dosáhneme větší dimenzionální redukce výsledného vektoru.

1.3 Určení podobnosti dokumentů

Určení podobnosti dvou vektorů (dokumentů) je čistě matematickou záležitostí. My si pouze určíme jakou podobnostní míru budeme používat. Pro dva dokumenty $d_1 = \{d_{11}, d_{12}, ...d_{1n}\}$ a $d_2 = \{d_{21}, d_{22}, ...d_{2n}\}$ si uvedeme několik možností.

1.3.1 Kosinová míra podobnosti

$$\mathrm{sim}_C(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\| \, \|d_2\|} \tag{1.10}$$

1.3.2 Míra symetrického překrytí

$$\mathrm{sim}_S(d_1, d_2) = \frac{\sum_{i=1}^{n} \min(d_1 i, d_2 i)}{\min(\sum_{i=1}^{n} d_1 i, \sum_{i=1}^{n} d_2 i)} \tag{1.11}$$

1.3.3 Diceho míra podobnosti

$$\mathrm{sim}_D(d_1, d_2) = \frac{2 \, \|d_1 \cap d_2\|}{\|d_1\| + \|d_2\|} \tag{1.12}$$

1.3.4 Jacardova míra podobnosti

$$\mathrm{sim}_J(d_1, d_2) = \frac{\|d_1 \cap d_2\|}{\|d_1 \cup d_2\|} \tag{1.13}$$

1.4 Latentní sémantická analýza

V předchozích sekcích jsme si ukázali základní principy pro určování podobnosti dokumentů. Nicméně existují další techniky, který tento proces zdokonalují, pro získání lepších výsledků. Latentní sémantická analýza (LSA) je technika určená pro analýzu vztahů mezi dokumenty a termy, které dokumenty obsahují, při čemž využívá jejich vektorovou reprezentaci. Vycházíme z předpokladu, že slova z podobné oblasti, tudíž slova spolu související, se vyskytují ve stejném dokumentu. Principu, který je popsán v předchozí větě, se říká: latentní sémantické vazby. Občas se také můžeme setkat s označením Latentní sémantické indexování (Latent Semantic Indexing - LSI).

1.4.1 Jak LSA funguje?

Původně bylo LSA vyvinuto pro získání několika relevantních dokumentů podle zadaného dotazu z velké kolekce dokumentů. Do té doby byl tento problém řešen pomocí vyhledávání klíčových slov v dokumentu, popř. vylepšenou metodou vážených klíčových slov a v neposlední řadě také vektorovým přístupem založeným na četnosti slov v dokumentu (metoda, kterou jsme si nastínili na začátku této kapitoly). LSA tento přístup obohacuje o využití tzv. Singular Value Decomposition (SVD). Do detailu si tento proces vysvětlíme později, ale v podstatě se jedná o nástroj lineární algebry, který přeorientuje a seřadí dimenze ve vektorovém prostoru dokumentů a zachytí nám skryté významové vazby mezi dokumenty.

Po využití SVD jsou jednotlivé dimenze ve vektorovém prostoru seřazeny od té nejvýznamnější po tu nejméně významnou. Díky této vlastnosti můžeme méně významné dimenze zanedbat, dojde tak k redukci dimenze. Tato redukce nám zajistí zrychlení výpočtu. Důležitějším faktem je, že touto dimenzionální redukcí přejdeme do tzv. „sémantického prostoru“. Většinou nám stačí např. pouze několik stovek dimenzí z původních tisíců nebo deseti tisíců.

1.4.1.1 Kolekce dokumentů

Pokud do samotné techniky LSA nebudeme počítat předzpracování dokumentů, ve kterém opět dochází k popsaným technikám v sekci 1.2, tak prvním krokem je vytvoření kolekce dokumentů. Tímto dokumentem bývá typicky jedna kniha, článek, kapitola, sekce nebo odstavec. Kolekci dokumentů pak poskládáme ze všech dokumentů, které budeme brát v potaz při hledání relevantních dokumentů.

1.4.1.2 Term-by-Document metiace

Druhým krokem je vytvoření matice výskytů tzv. *Term-By-Document*, kterou si označíme A. Její matematický zápis můžeme vidět ve vzorci 1.14.

$$t_i^T \rightarrow \overset{\displaystyle d_j \atop \downarrow}{\begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix}} \tag{1.14}$$

Jeden řádek v této matici popisuje relaci daného termu ke všem dokumentům, tj. řádek reprezentuje term.

$$t_i^T = \begin{bmatrix} a_{i,1} & \dots & a_{i,n} \end{bmatrix} \tag{1.15}$$

Naopak sloupec popisuje výskyt jednotlivých termů v daném dokumentu, tj. sloupec nám reprezentuje dokument.

$$d_j = \begin{bmatrix} a_{1,j} \\ \vdots \\ a_{m,j} \end{bmatrix} \tag{1.16}$$

Jinými slovy můžeme říci, že buňka i, j nám popisuje výskyt termu i v dokumentu j. Je samozřejmě nutné podotknout, že každý term je unikátní, v *Term-by-Document* matici se nevyskytují jejich duplicity. To samé platí i pro dokumenty, ale to snad ani není třeba zdůrazňovat.

Na každou buňku z matice je použita určitá metodika, která ohodnotí výskyt daného termu v daném dokumentu a výskyt daného termu napříč celou kolekcí dokumentu. Jedná se o lokální metriku (v rámci jednoho daného dokumentu) a o globální metriku (napříč celou kolekcí dokumentů). Tyto metodiky jsou popsány v kapitole 1.1.1 a 1.1.2. Většina studií ukázala, že pro většinu kolekcí dokumentů nejlépe funguje *LogEntropy*, tj. každá buňka z matice A je vypočtena pomocí vzorce 1.18.

$$g_i = 1 + \sum_j \frac{p_{ij} \log p_{ij}}{\log n}\text{ , kde } p_{ij} = \frac{tf_{ij}}{gf_i} \tag{1.17}$$

$$a_{ij} = g_i \log(tf_{ij} + 1) \tag{1.18}$$

1.4.1.3 Singular Value Decomposition

Dalším krokem je samotný výpočet SVD. Nejdříve je nutné si několik věcí ujasnit. Skalární součin $t_i^T t_p$ nám určuje korelaci mezi termy napříč dokumenty. Pokud vynásobíme matice AA^T, dostaneme matici, která bude obsahovat všechny tyto skalární součiny. Jinými slovy buňka i, p obsahuje skalární

součin $t_i^T t_p$, stejně tak jako buňka p, i obsahuje skalární součin $t_p^T t_i$. Obdobně bude výsledná matice po maticovém násobení matic $A^T A$ obsahovat skalární součiny všech vektorů dokumentů, které udávají jejich korelaci napříč termy. Tentokrát si tento fakt uvedeme matematickým zápisem: $d_j^T d_q = d_q^T d_j$.[3]

Z teorie lineární algebry víme, že existuje dekompozice matice A, po jejím provedení získáme dvě ortogonální matice U, V a jednu diagonální matici S.[9]

$$A = USV^T \tag{1.19}$$

$$\begin{aligned} AA^T &= (USV^T)(USV^T)^T \\ &= (USV^T)(V^{T^T} S^T U^T) \\ &= USV^T V S^T U^T \\ &= USS^T U^T \end{aligned} \tag{1.20}$$

$$\begin{aligned} A^T A &= (USV^T)^T (USV^T) \\ &= (V^{T^T} S^T U^T)(USV^T) \\ &= VS^T U^T USV^T \\ &= VS^T SV^T \end{aligned} \tag{1.21}$$

Protože SS^T a $S^T S$ jsou diagonální, matice U musí obsahovat vlastní vektory AA^T, zatímco V musí obsahovat vlastní vektory $A^T A$.

$$\begin{array}{ccccccc} & U & & S & & \begin{array}{c} V^T \\ (\hat{d}_j) \\ \downarrow \end{array} \\ (\hat{t}_i^T) \rightarrow & \left[\left[\begin{array}{c} \\ u_1 \\ \\ \end{array} \right] \ldots \left[\begin{array}{c} \\ u_l \\ \\ \end{array} \right] \right] & \cdot & \left[\begin{array}{ccc} \sigma_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_l \end{array} \right] & \cdot & \left[\begin{array}{c} \left[\quad v_1 \quad \right] \\ \vdots \\ \left[\quad v_l \quad \right] \end{array} \right] \end{array} \tag{1.22}$$

Hodnoty $\sigma_1 ... \sigma_l$ se nazývají singulární hodnoty a $u_1 ... u_l$ resp. $v_1 ... v_l$ jsou levé a pravé singulární vektory. Na předchozím zápisu SVD je důležité si povšimnout, že pouze matice U v sobě uchovává informace o vektorech t_i. Řádkové vektory z matice U pro další využití nazveme $\hat{t}_i$. To samé co bylo řečeno o matici U a vektorech t_i můžeme říci o matici V^T a vektorech dokumentů d_j, opět tyto sloupcové vektory z matice V^T nazveme $\hat{d}_j$.

1.4.1.4 Dimenzionální redukce

Posledním krokem samotného výpočtu je dimenzionální redukce na základě parametru k. Podle tohoto parametru jsou všechny tři matice dimenzionálně zredukovány. Samotné velikosti parametru k a jeho určení se věnuje sekce 1.4.1.5.

1.4.1.5 Určení parametru dimenzionální redukce

Parametr k, který nám určuje dimenzionální redukci pro všechny tři, matice se typicky pohybuje v rozmezí mezi čísly 100 a 300. Pokud zvolíme vhodnou kombinaci lokální a globální metodiky ohodnocení výskytu, můžeme k určení parametru k zvolit Frobeinovu normu, také někdy nazývanou Euklidovská. Frobeinovu normu vypočítáme pomocí následujícího vzorečku 1.23.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\|a_{ij}\|^2} = \sqrt{trace(A*A)} = \sqrt{\sum_{i=1}^{min(m,n)} \sigma_i^2} \tag{1.23}$$

I přes tuto dimenzionální redukci jsou zachovány důležité sémantické informace, zatímco dojde k redukci šumu a odstranění dalších nežádoucích faktorů, které se mohly vyskytovat v originálním vektorovém prostoru matice A. Nejdůležitějším faktem dimenzionální redukce je ten, že po její provedení můžeme pracovat v tzv. „sémantickém prostoru“.

1.4.1.6 Shrnutí

Výsledkem výpočtu LSA jsou tři matice. Dvě z těchto tří matic definují dva rozdílné vektorové prostory, které jsou rovněž rozdílné od vektorového prostoru definovaného originální *Term-by-Document* maticí. Třetí matice, která obsahuje singulární hodnoty, slouží k převodu vektorů z jednoho prostoru do druhého. Všechny matice mají redukovanou dimenzi na základě parametru k.

1.4.2 Dotazování v LSA

Jak bylo uvedeno výše, pohybujeme se v aproximovaném prostoru s nižší dimenzí, než měl původní vektorový prostor. Toto je důležité si uvědomit, abychom správně pracovali ať už s vektory dokumentů nebo termů. Tuto aproximaci můžeme napsat následujícím způsobem.

$$A_k = U_k S_k V_k^T \tag{1.24}$$

Nyní se můžeme podívat na možnosti dotazování, které nám LSA nabízí:

- Porovnání dokumentu j a dotazu q (to samé platí pro dva dokumenty z kolekce dokumentů).
- Porovnání termu i a dotazu p (opět můžeme porovnávat dva termy, které se vyskytují v kolekci dokumentů).
- Vektory termů a dokumentů můžeme clustrovat za použití clustrovacích algoritmů (např. k-means algoritmu).

Jelikož v této práci využíváme právě první zmiňované možnosti, podíváme se na ni podrobněji. Nejdříve je nutné říci, co vlastně budeme porovnávat. V tomto případě jsou to vektory $S_k\hat{d}_j$ a $S_k\hat{q}$. Abychom mohli vykonat toto porovnání, musíme nejdříve převést původní vektor dokumentu a vektor dotazu do nízkodimenzionálního prostoru. To provedeme pomocí následujících vzorců.

$$\hat{d}_j = S_k^{-1} U_k^T d_j \tag{1.25}$$

$$\hat{q} = S_k^{-1} U_k^T q \tag{1.26}$$

Nejdříve je nutné vektor dotazu sestavit. Termy, které se vyskytují v dotazu, ale nikoli v kolekci dokumentů, nám neposkytnou žádnou informaci, takže je můžeme zanedbat a ve vektoru je nebereme v potaz. Naopak musíme vektor „rozšířit“ na správnou dimenzi. Zde máme na mysli termy, které se vyskytují v kolekci dokumentů, ale nikoli v dotazu.

1.4.2.1 Výpočet relevance

Posledním krokem je samotný výpočet relevence na základě podobnostní míry. V našem případě jsme zvolili kosinovou míru podobnosti. Ve vzorečku 1.27 vidíme, jak vypočítáme relevanci každého dokumentu z kolekce vzhledem k dotazu.

$$\mathrm{sim}_C(S_k\hat{d}_j, S_k\hat{q}) = \frac{S_k\hat{d}_j \cdot S_k\hat{q}}{\left\|S_k\hat{d}_j\right\| \left\|S_k\hat{q}\right\|} = \frac{\sum_{i=1}^{n} S_k\hat{d}_j \times S_k\hat{q}}{\sqrt{\sum_{i=1}^{n} (S_k\hat{d}_j)^2}\sqrt{\sum_{i=1}^{n} (S_k\hat{q})^2}} \tag{1.27}$$

KAPITOLA 2

Extrakce tagů

2.1 Automatická extrakce tagů

Automatická extrakce tagů, nebo chcete-li klíčových slov, je úkol, kdy se snažíme z daného dokumentu získat množinu slov, popř. frází, které nám nejlépe popisují daný dokument. Takto získaná slova můžeme dále využít pro spoustu data miningových úloh, jako jsou např. automatická indexace, automatické clusterování, automatické filtrování a spoustu dalších. Z výše uvedeného vyplývá, že můžeme na extrakci tagů nahlížet jako na jeden z klíčových kroků zpracování textových dokumentů.

V poslední větě jsme u extrakce tagů nezmínili důležité slovo: automatická. Jinými slovy bychom mohli dokumenty opatřit klíčovými slovy sami, u některých dokumentů a systémů se tomu tak i děje. Nicméně tato manuální extrakce tagů většinou připadá v úvahu pouze, pokud jsme sami autoři daného dokumentu, v opačném případě bychom se s daným dokumentem museli důkladně seznámit a teprve potom bychom byli schopni ho opatřit klíčovými slovy. Což dozajista zabere spoustu času, proto se většina systémů snaží člověku usnadnit práci a nabídnout mu právě automatickou extrakci tagů.

K automatické extrakci klíčových slov můžeme přistupovat několika způsoby. V podstatě se jedná o čtyři kategorie: statistický přístup, lingvistický přístup, přístup strojového učení a poslední skupinu tvoří přístupy, které nezapadají ani do jednoho zmíněného. [4]

2.1.1 Statistický přístup

Tento přístup je jednoduchý a bezesporu je jeho výhodou, že není zapotřebí trénovacích dat. Principem tohoto přístupu je, že ke stanovení klíčových slov použijeme statistickou informaci jednotlivých slov obsažených v dokumentu. Statistickou informací může být např. N-Gram, což je v podstatě sled po sobě

jdoucích slov, frekvence slova, *TF-IDF* nebo také jakákoliv lokální nebo globální metodika zmíněná v sekcí 1.1.1 a 1.1.2. Samozřejmě můžeme jednotlivé zdroje statistické informace kombinovat a zkombinovat tak např. N-gramy proměnné velikosti s *TF-IDF*.

2.1.2 Přístup strojového učení

V tomto případě se jedná o klasické využití strojového učení a to tak, že máme určitou množinu dokumentů, které jsou již opatřeny klíčovými slovy. Z této množiny dokumentů se vytvoří model, říkáme že se „naučí“, proto také strojové učení. Tento model je posléze aplikován na dokumenty, které mají být automaticky opatřeny tagy. V souvislosti s klíčovými slovy jsou nejčastěji používanými modely Bayesovské sítě a podpůrné vektory (SVM).

2.1.3 Lingvistický přístup

Tento přístup je založen čistě na lingvistických rysech slov, slovních spojení, vět, souvětí a větších textových celků. Pod tímto spojením si můžeme představit lexikální analýzu, která nám ze vstupní posloupnosti znaků vytvoří tzv. lexémy neboli lexikální jednotky, syntaktickou analýzu, díky které získáme gramatickou strukturu vůči předem dané formální gramatice, morfologickou analýzu, která se naopak zabývá nejmenšími významonosnými či význam modifikujícími výrazy tzv. morfémy, sémantickou analýzu, která se zabývá významem jazykových výrazů, ať už zmíněných morfémů, slov, frází, vět či souvětí. A v neposlední řadě je to analýza diskurzu, také známa jako textová lingvistika, která se zabývá principy spojování vět a souvětí ve větší textové celky.

2.1.4 Ostatní přístupy

Jak už bylo zmíněno, do této kategorie spadají všechny ostatní přístupy, které striktně nezapadají ani do jedné z výše uvedených kategorií. Většinou se jedná o přístupy, které kombinují dva nebo hned všechny tři přístupy. Také sem spadají automatické extrakce klíčových slov, které využívají nějaký heuristický přístup. V tomto případě pak využíváme např. pozici slova, délku slova, rozložení slov a další.

2.2 Zvolený přístup

V této sekci si představíme přístup automatické extrakce klíčových slov, který je použit v naší aplikaci. Jedná se o kombinaci heuristické metody se všemi třemi způsoby: strojového učení, statistického a lingvistického přístupu. Celý proces si představíme v jednotlivých krocích.

2.2.1 POS tagging

POS tagging (part-of-speech tagging) je proces značkování slov na základě definice a kontextu, neboli vztahu se souvisejícími slovy ve větě, frázi či odstavci. Značkuje se v několika kategoriích, počet kategorií je odvislý od daného jazyka, tou nejjednodušší je právě slovní druh, z jehož názvu je tento proces odvozen. Tabulka 2.1 ukazuje jednotlivé kategorie pro český jazyk [6].

Pozice	**Označení**	**Popis**
1	POS	Slovní druh
2	SUBPOS	Upřesněný slovní druh
3	GENDER	Rod
4	NUMBER	Číslo
5	CASE	Pád
6	POSSGENDER	Rod vlastníka
7	POSSNUMBER	Číslo vlastníka
8	PERSON	Osoba
9	TENSE	Čas
10	GRADE	Stupňování
11	NEGATION	Negace
12	VOICE	Slovesný rod
13	RESERVE1	Nepoužívané, rezerva
14	RESERVE2	Nepoužívané, rezerva
15	VAR	Speciální použití

Tabulka 2.1: POS tagging - kategorie

Nás převážně zajímá hned první kategorie a to je slovní druh. Tabulka 2.2 ukazuje jednotlivé slovní druhy a jejich zkratky [6]. Ještě krátce k ostatním kategoriím. Druhá kategorie je nejobsáhlejší a můžeme v ní najít např. demonstrativní zájmena (ten, onen...), ale také vlastní hodnoty pro slova „krát“ nebo „kolik“. Těchto jednotlivých hodnot je téměř osmdesát. Rod není až tak jednoduchou skupinou, jak by se na první pohled mohlo zdát, protože rozlišujeme celkem jedenáct různých hodnot (mužský neživý, ženský, střední, ženský nebo střední...). Číslo kromě klasického množného a jednotného ještě nabízí hodnotu podvojného čísla. Pád je oproti předchozím kategoriím jasný, zde se nabízí sedm hodnot pro pády od prvního po sedmý, plus hodnota pro jakýkoliv pád. Osoba je opět jasná, první, druhá nebo třetí. Kategorie čas obsahuje hodnoty: minulý, přítomný, budoucí a ještě hodnota pro minulý nebo přítomný (pokud nejsme schopni čas rozlišit). Stupňování nabývá tří hodnot končící superlativem. A poslední skupinou je slovesný rod: činný nebo trpný.

Hodnota	Popis
A	Přídavné jméno
C	Číslice
D	Příslovce
I	Citoslovce
J	Spojka
N	Podstatné jméno
P	Zájmeno
V	Sloveso
R	Předložka
T	Částice
X	Neurčeno, nezařaditelné
Z	Punkce

Tabulka 2.2: POS tagging - slovní druhy

2.2.2 Ohodnocení slova (skóre)

Po fázi značkování slov, která spadala do lingvistického přístupu, ale zároveň strojového učení, protože pro značkování je vytvořen model z korpusu dokumentů určených pro naučení modelu, přejdeme k heuristické a statistické části. Statistický přístup zde vychází z použití metodiky ohodnocení váhy slova *TF-IDF*, která je popsaná v sekci 1.1.1, resp. 1.1.2. Naopak z heuristického pohledu je zde využita délka slova označená *Len*.

$$Len = \frac{\log_2(\text{Délka aktuálního slova})}{\log_2(\text{Maximální délka slova v dokumentu})} \tag{2.1}$$

A první výskyt daného slova, který označíme jako *Dep*.

$$Dep = \frac{1}{\log_2(\text{Pozice prvního výskytu slova} + 1)} \tag{2.2}$$

Celkové skóre, které označíme jako *Score*, je stanoveno následujícím vztahem.

$$Score = TF\text{-}IDF * Len * Dep \tag{2.3}$$

2.2.3 Ohodnocení posloupnosti

Každé slovo má nyní stanoveno svoje skóre. V následujícím kroku vytvoříme posloupnosti slov o délce n. Délka n je proměnná a nabývá hodnot jedna, dva nebo tři. Tyto posloupnosti jsou tvořeny po sobě jdoucími slovy. Skóre celé posloupnosti, pokud n je vyšší než jedna, je stanoveno následujícím vzorcem 2.4, kde $Score_i$ jsou *Score* jednotlivých slov v posloupnosti a n je délka posloupnosti. Pokud je n rovno jedné, ohodnocení celé posloupnosti je rovno ohodnocení daného slova.

$$Score_{seq} = \frac{\max(Score_i) + n \cdot \min(Score_i)}{n+1} \tag{2.4}$$

Proč je použit daný vzorec? Snažíme se zohlednit posloupnosti, které v sobě obsahují slova, která mají vysoké *Score*. To nám zařizuje minimální hodnota *Score* slova, které se v posloupnosti objevuje. Jinými slovy pokud je v posloupnosti slovo, které má malé *Score*, celá posloupnost bude mít relativně nízké ohodnocení. Minimální hodnotu *Score* násobíme n, abychom „zvýhodnily“, u kterých se *Score* jednotlivých slov tolik neliší. Nemůžeme jednoduše sečíst hodnoty *Score* jednotlivých slov v posloupnosti, protože pak by vždy byla posloupnost délky větší než jedna ohodnocena větším *Score* než jednotlivá slova dané posloupnosti.

V dalším kroku projdeme všechny posloupnosti a vyřadíme všechny, kde se nevyskytuje podstatné jméno, protože opět nás nezajímají tagy jako např. „databázový“, „relační“ apod. i když mohou mít vysoké skóre, ale zajímají nás tagy jako „databázový server“ nebo „relační databáze“, které mohou mít relativně nižší skóre než předchozí dva příklady. Zde dojde k využití lingvistického přístupu. Jako další vylepšení se ukázalo zohledňování posloupností, kde se vyskytuje podstatné jméno v prvním pádu.

Ještě před seřazením tagů od nejvyššího skóre po nejnižší dochází k zohlednění tagů, které se již v systému portálu Spolupráce s průmyslem vyskytují. Každý tag, který se již v systému objevuje, je vynásoben parametrem, který je možno měnit přes konfigurační soubor. Samozřejmě je jasné, že aby docházelo k zvýhodnění, je nutné, aby byl parametr větší než jedna. A naopak pokud ho nastavíme jako roven jedné, pak je tento krok v podstatě vyřazen. Nyní dojde k seřazení tagů podle jejich skóre (relevance) od největšího skóre po nejnižší.

2.2.4 Odstranění duplikátů tagů

Pod odstraněním duplikátů nemáme na mysli pouze odstranění naprosto stejných tagů, což se samozřejmě může stát, protože daná posloupnost slov se může vyskytovat v textu dokumentu několikrát. Nás v tomto případě zajímá, zda je tag ještě relevantní, co se předchozích tagů týče, tj. tagů s větším *Score*. Jde o to, abychom např. nevraceli dva tagy „analýza relační databáze“ a „relační databáze“, kdy oba tagy budou mít relativně podobné *Score*.

Odstranění duplikátů závisí na několika kritériích, podle kterých se rozhodujeme, zda daný tag odstraníme, nebo ho budeme považovat za relevantní. Pokud ani jedno slovo z daného tagu ještě není obsaženo v seznamu relevantních tagů, daný tag považujeme za relevantní (tato situace určitě nastane u prvního tagu, protože tagy jsou seřazeny podle relevance). Určitě chceme vždy mezi relevantní tagy zařadit tag, který se již v portálu Spolupráce s průmyslem vyskytuje. Třetí kritérium se vztahuje pouze na tagy, které obsahují více slov. Zjistíme zda tag může nějakým svým slovem obohatit relevantní tagy (alespoň jedno slovo v tagu se nevyskytuje v seznamu relevantních tagů). V případě, že ano, existuje další rozhodovací proces, na jehož základě rozhodneme, zda daný tag budeme považovat za relevantní.

Zajímají nás tagy, které v sobě obsahují jednotlivá slova s podobným *Score* (díky tomu zanedbáme veškeré dvojice a trojice, ve kterých figurují spojky, zájmena a další). K tomu využijeme minimální a maximální hodnotu *Score* slov v tagu, pokud je absolutní hodnota jejich rozdílu menší než minimální hodnota *Score* slova z posloupnosti, budeme daný tag považovat za relevantní. Sledujeme také pozici jednotlivých slov v tagu, pokud se dané pozice již objevují mezi relevantními tagy, pravděpodobně jsme již obdobný tag s vyšší relevancí do relevantních tagů zařadili. Může nastat situace, kdy bychom odstranili tag, který se objevuje v textu několikrát, a to na základě opatření s absolutní hodnotou rozdílu minimálního a maximálního *Score*. V takovém případě zjišťujeme, zda daný tag netvoří, alespoň jedno procento z celkové délky textu (vyskytuje se přibližně v každé desáté větě). Pokud ano, tag zařadíme mezi relevantní tagy. Po odstranění duplikátů získáme výsledný seznam relevantních tagů seřazených podle relevance.

KAPITOLA 3

Analýza využitelných dat

3.1 Základní entity

Analytické metody jsou postaveny na pěti základních entitách, kterými jsou: kurz (předmět), student, expert, skill a tag. Z těchto základních entit budeme při analýze vycházet.

3.2 KOSapi

Jako jeden z hlavních zdrojů externích dat bylo vybráno KOSapi, což je aplikační rozhraní v podobě RESTful webových služeb, které zprostředkovává přístup k vybrané části dat v databázi KOS. Jinými slovy můžeme se dostat např. ke všem předmětům, co se na FIT ČVUT vyučují, nebo ke všem studentům studujícím na fakultě. Navíc je velkou výhodou, že je možné použít pro filtrování výsledků RSQL výraz.

Hned pro začátek je nutné uvést, že jednotlivé filtrační podmínky a např. vlastnosti kurzu, které budou brány v potaz, se mohou nastavovat v konfiguračním souboru. Jelikož vycházíme z XML struktury, která je určená rozhraním KOSapi, není zde úplná volnost, ale např. při přidání nového atributu u kurzu v KOSapi, není nutné zasahovat do implementace, stačí upravit konfigurační soubor.

3.2.1 OAuth

KOSapi umožňuje přístup pro studenty pod jejich školním heslem a uživatelským jménem, což je ale samozřejmě v našem případě nežádoucí, protože přístup k datům z KOSapi provádí samotná aplikace. Proto byla využita možnost, že KOSapi podporuje OAuth 2.0, což je moderní autorizační protokol,

jehož hlavní výhodou je poskytnutí přístupu k datům určité služby bez nutnosti vyzrazení klientových přihlašovacích údajů do dané služby.

V OAuth 2.0 jsou definovány tyto role: [1]

- resource owner (uživatel) – vlastník chráněného zdroje (dat), entita schopná přidělit nebo odepřít přístup ke chráněnému zdroji (typicky se jedná o koncového uživatele),
- resource server (služba) – poskytovatel a hostitel chráněného zdroje, entita schopná obsluhovat požadavky (obsahující access token) ke chráněnému zdroji (typicky se jedná o serverovou službu vystavující API),
- client (klient) – aplikace, která přistupuje ke chráněnému zdroji na resource serveru s oprávněními resource ownera (uživatele),
- authorization server (autorizační server) – server, který klientovi vydává access token v případě jeho úspěšné autentizace od resource ownera (uživatele) a získání autorizace (autorizační server může být přímo součástí resource server nebo oddělený).

V případě naší aplikace se využije tzv. *grant Client Credentials*, kdy se v podstatě aplikace autentizuje sama za sebe. Celý proces je velmi jednoduchý, na adresu *Authorization Endpoint* daného autorizačního serveru jsou poslány hodnoty *Client ID* a *Client Secret* s tím, že je nutné serveru říci, že nám jde právě o zmiňovaný *grant Client Credentials*. Server nám vrátí *Access token*, který je použit pro přihlášení do služby KOSapi, která musí pouze ověřit, zda je daný přístupový token validní, tj. byl vydán autorizačním serverem.

3.2.2 Kurzy

KOSapi nabízí získání seznamu všech předmětů, které navíc můžeme filtrovat na základě katedry (podle dokumentace KOSapi se tato vlastnost nazývá středisko). Výpis kurzu obsahuje spoustu informací jako např. povolený počet zapsání, jazyk výuky, způsob zakončení, počet kreditů a spoustu dalších. Jelikož se naše analytické služby zaměřují na text, jsou pro nás klíčové vlastnosti, které mají určitou informační hodnotu.

Ze všech vlastností co KOSapi o předmětu nabízí, byly vybrány tyto vlastnosti kurzu: název, popis, osnova přednášek, literatura, cíle předmětu, požadavky, osnova cvičení a klíčová slova. Z těchto jednotlivých vlastností je vytvořen textový obsah dokumentu pro daný kurz. Jelikož při určování relevance dokumentů nebereme v potaz strukturu dokumentu ani pořadí jednotlivých termů, jsou hodnoty jednotlivých vybraných vlastností jednoduše zřetězeny.

3.2.3 Studenti

I u studentů je využita možnost filtrování záznamů, tentokrát na základě fakulty. U studentů nás převážně zajímají úspěšně absolvované kurzy. Jelikož už máme dokumenty jednotlivých kurzů, tak je snadné ze všech dokumentů absolvovaných kurzů poskládat dokument studenta tím, že tyto dokumenty jednoduše zřetězíme.

Jak bylo řečeno v sekci 1.4 očekáváme, že text dokumentu spolu souvisí. Zde by se mohlo zdát, že spolu texty jednotlivých předmětů nemusí souviset vůbec, zvláště jestli budeme porovnávat předměty humanitního a technického zaměření. Nicméně pro nás je touto vazbou mezi jednotlivými předměty právě student, který předměty vystudoval. Tím pádem při výpočtu LSA a přechodu do sémantického prostoru se studenti, kteří absolvovali podobné předměty, budou shlukovat.

Je samozřejmě jasné, že předměty, které jsou celofakultní, absolvovali všichni studenti. Na druhou stranu pomocí globální metodiky 1.1.2 docílíme toho, že termy z těchto předmětů nebudou zdaleka tak důležité na rozdíl od termů, které se vyskytují v předmětech, co absolvovalo pouze několik desítek studentů. A právě absolvováním těchto předmětů se jednotliví studenti odliší.

3.2.4 Experti

Jako experti byli vybrání všichni učitelé, kteří spadají pod katedru, která je podřízena FIT ČVUT, a konkrétně nás zajímají předměty, které vyučující přednáší, cvičí, zkouší nebo garantuje, a to ať už v tomto semestru nebo v jakémkoliv jiném. Co se vytvoření dokumentu a významu tohoto dokumentu týče, platí to samé co v případě studentů.

3.3 DBpedia

Další entitou, o které se budeme bavit, jsou skilly. Jelikož skilly jsou čistě interní věc portálu Spolupráce s průmyslem, KOSapi nám v tomto případě nijak nepomůže. Skilly jsou v systému portálu vedeny pouze pod unikátním identifikátorem a názvem. Nicméně název, převážně jedno či dvouslovný, je pro jakoukoliv textovou analytickou metodu nedostačující, proto je nutné získat tyto data z jiného, externího zdroje.

Bylo by možné opatřit každý skill anotací manuálně. V souvislosti s extrakcí tagů, dáváme přednost zautomatizování veškeré činnosti. A to nejen z pohledu usnadnění práce, ale především protože se skilly mohou kdykoliv v budoucnu měnit. Jako zdroj anotace k jednotlivým skillům byly vybrány záznamy z české DBpedie.

Česká DBpedia poskytuje SPARQL endpoint, na který můžeme posílat SPARQL dotazy, které se provedou nad datasety DBpedie. Nejdřív je důležité říci, že každý záznam na DBpedii obsahuje rozsáhlé a mnohdy rozdílné

vlastnosti, ale vždy obsahuje *rdfs:label* pro název daného záznamu, *dbpedia-owl:abstract* pro popis záznamu a *dbpedia-owl:wikiPageID* jako unikátní identifikátor záznamu. První dvě vlastnosti ze zmiňovaných použijeme pro textovou anotaci skillu a třetí jako unikátní identifikátor.

Protože skillů není v současné době v portálu Spolupráce s průmyslem velké množství, zvolili jsme metodu, že každému skillu vyhledáme na DBpedii několik záznamů. Toto má hned několik výhod. Zvýšíme počet dokumentů, nad kterými se budeme dotazovat, a tím pádem budeme moci přejít do sémantického prostoru (pokud by byla dimenze dokumentů nízká, nemohli bychom provést dimenzionální redukci a nepřešli bychom tak do sémantického prostoru).

Druhá výhoda vyplývá z toho, že se jedná o automatický proces. Může nastat situace, kdy nalezený záznam bude odpovídat všem stanoveným kritériím (vzhledem k názvu skillu), přesto s daným skillem nebude mít nic společného a může být úplně z jiné oblasti. Díky tomu, že je z jiné oblasti, bude mít při dotazovaní velmi nízkou relevanci. Kdybychom pro každý skill hledali pouze jeden záznam, mohlo by se stát, že bychom získali právě tento záznam, který s daným skillem nemá nic společného.

3.3.1 Vyhledávání záznamů

Vyhledávání skillů probíhá ve čtyřech úrovních, nejdříve se podíváme zda existuje na DBpedii záznam, jehož název přesně odpovídá názvu skillu u nás v systému. V druhé úrovni se podíváme, zda existuje záznam, kde se název skillu objevuje v názvu záznamu a zároveň v popisu záznamu se vyskytuje posloupnost slov, která odpovídá názvu skillu u nás v systému. Při hledání skillu na třetí úrovni musí popis záznamu na DBpedii obsahovat všechna slova z názvu skillu, ale nemusí být v posloupnosti slov za sebou. V poslední nejvolnější úrovni stačí, když je obsaženo alespoň jedno slovo z názvu skillu v popisu na DBpedii.

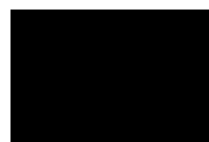
V každé úrovni hledání je nastavený limit vrácených záznamů z DBpedie a zároveň je stanoven minimální počet záznamů pro jeden skill, který se snažíme na DBpedii najít. Jinými slovy pokud dojde na druhé úrovni k nalezení minimálního stanového počtu, hledání přerušíme. Tímto zabráníme, abychom hledali méně související záznamy pro skilly, ke kterým jsme na DBpedii již našli více související záznamy.

Co z celého procesu hledání záznamů na DBpedii získáme? Je to seznam záznamů s jejich popisem a názvem, kdy každý tento záznam připadá určitému skillu u nás v systému. Nevytváříme tak každému skillu dokument tím způsobem, že bychom zřetězili jednotlivé záznamy nalezené na DBpedii, a to z důvodů které byly uvedeny v předchozí sekci (tj. větší počet dokumentů a možné nalezení nesouvisejícího záznamu), ale každý záznam z DBpedie tvoří samostatný dokument.

3.4 Portál Spolupráce s průmyslem

Poslední entita, která nám zůstala, jsou tagy. Zde není zapotřebí vytvářet kolekci dokumentů, protože pro automatickou extrakci tagů využijeme kolekci dokumentů jiné entity. Vycházíme ze samotného textu dotazovaného dokumentu. Nicméně pro určité zlepšení automatické extrakce bereme v potaz tagy, které se již v portálu Spolupráce s průmyslem vyskytují. Tudíž se pro nás stane portál zdrojem seznamu používaných tagů. A stejně tak z portálu potřebujeme získávat seznam aktuálních skillů, abychom mohli provést nalezení jejich anotací na DBpedii.

KAPITOLA 4



Použité technologie

4.1 JAMA

JAMA je balíček základní lineární algebry pro Javu, který poskytuje třídy pro konstrukci a manipulaci s maticemi. Je navržen tak, aby poskytl dostatečné prostředky pro běžné problémy lineární algebry. Pro naše účely byl vybrán tento balíček především, protože má v sobě implementovanou *Singular Value Decomposition*.

Nicméně v průběhu práce na této diplomové práci se objevilo několik problémů z hlediska používání tohoto balíčku. Převážně to bylo z pohledu práce s pamětí. JAMA každou matici reprezentuje jako dvourozměrné pole datového typu double. Navíc každá třída, která se stará o jednotlivé dekompozice (Choleského dekompozice, SVD, LU dekompozice, QR dekompozice), má v sobě jako privátní proměnnou minimálně jednu matici, se kterou daná dekompozice pracuje. Dalším problémem z hlediska paměti jsou *gettery*, které vždy vracejí kopii vypočtené matice. Což je z hlediska paměti velice neúsporné, zvláště když si uvědomíme, že se o správu paměti v Javě stará *Grabage Collector*, který může provést úklid až za nějakou dobu. Do té doby nám tyto vnitřní nebo nakopírované matice zabírají paměť, i když je nadále nepotřebujeme.

Kvůli výše uvedeným důvodům bylo nutné knihovnu upravit tak, že každé jednotlivé třídě z balíčku JAMA byla implementována metoda, která se postará o odstranění vnitřních matic dané třídy (jednotlivých dekompozic a samotné třídy, která reprezentuje matici jako takovou) z paměti. Samozřejmě objekt jako takový maže stále Garbage Collector, což nám ale již nečiní potíže s pamětí, protože nyní už obsahuje maximálně několik údajů o velikosti vnitřních matic, nebo několik málo pomocných proměnných. Když přistoupíme na toto řešení, musíme si dávat pozorm, kdy metodu na vyprázdnění paměti voláme, abychom potom nepřistupovali k maticím, které jsou již z paměti odstraněny. Je nutné postupovat při mazání obezřetně, na druhou stranu

nejlepší je smazat matice z paměti v okamžiku, kdy víme, že nadále danou matici nebudeme potřebovat, abychom místo v paměti uvolnili co nejdříve.

S touto úpravou a správnou prací s pamětí bylo již možné pracovat s maticemi o dimenze řádově vyšší než předtím. Nicméně při násobení dvou matic musíme přistupovat k oběma maticím a navíc potřebujeme třetí matici, do které ukládáme výsledky násobení. Z toho nám vyplývá, že při násobení máme v paměti minimálně tři matice, což se může ukázat jako problém, pokud jsou tyto matice příliš velké (stačí matice o velikosti v řádech tisíců sloupců a deseti tisíců řádků).

4.2 KDG Commons - MappedFileBuffer

Proto ani tato úprava nebyla dostatečná a vnitřní reprezentace matice v třídě *Matrix* bylo nutné předělat na *MappedFileBuffer*, kdy se do paměti ukládá pouze mapování daného souboru, a nikoli hodnoty jednotlivých buněk matice, ty jsou uloženy v souboru. K tomuto účelu byla využita část balíčku z KDG Commons a to právě *MappedFileBuffer*. Což je balíček, který obaluje paměťové mapování souborů, přičemž zachovává sémantiku *ByteBufferu*. Na rozdíl od klasického *byte bufferu* se ke všemu přistupuje pomocí absolutního indexu, což se nám z hlediska matic hodí. Jen je nutné stanovit, jak budeme matice do souboru ukládat (samozřejmě stejným způsobem je budeme i ze souboru načítat). V našem případě jsme zvolili toto schéma reprezentace jednotlivých buněk matice:

$$index = (i * n * doubleBytes) + (j * doubleBytes); \qquad (4.1)$$

Kde nám *index* udává místo, kde je daná buňka uložena v souboru, i a j je pozice buňky, n je počet sloupců matice a *doubleBytes* je velikost jednoho datového typu double v bytech. Z toho nám vychází, že matice je v souboru uložena po řádcích.

S touto úpravou bylo již možné provádět násobení velkých matic, které mají rozměry v řádech tisíců resp. deseti tisíců. Nicméně ani zde se implementace neobešla bez problémů, protože při vytváření velkých objektů *MappedFileBuffer* dochází v Javě k chybám. Tudíž bylo zapotřebí nastavit maximální velikost těchto souborů a matici tak rozdělit do menších souborů. S takto rozdělenými soubory pracuje pouze třída *Matrix*, která nám matici reprezentuje. Nicméně pokud ukládáme matici pro pozdější zpracování, tj. v případě výpočtu LSA, ukládáme ji již do normálního souboru prostým kopírováním obsahu jednotlivých mapovaných souborů. Pokud matici v budoucnu potřebujeme, ze zdrojového souboru ji načteme a opět provedeme rozdělení do menších mapovaných souborů.

Z výše uvedeného je jasné, že nám *index* označuje absolutní umístění buňky v rámci abstraktního souboru, který si můžeme představit, tak že zastřešuje

menší rozdělené soubory. Pro získání hodnoty buňky potřebujeme *fileIndex*, který určuje pořadí souboru, ve kterém se buňka vyskytuje, a *indexInFile*, který určuje již absolutní pozici v daném souboru.

$$fileIndex = \frac{index}{maxFileSize} \tag{4.2}$$

$$indexInFile = index \bmod maxFileSize \tag{4.3}$$

Samozřejmě pokud dosáhneme určitého zlepšení v jedné oblasti, zde se jednalo o práci s pamětí, není vyloučeno, že dojde ke zhoršení v jiné oblasti. A k tomu došlo i v tomto případě. Konkrétně se jedná o zpomalení rychlosti veškerých výpočetních operací s maticemi. Když se nad tím zamyslíme, zjistíme, že nám to prakticky nevadí. Nejdéle trvá operace výpočtu LSA. Tento výpočet se ale provádí asynchronně a výpočet samotné relevance dokumentu je pouze vynásobení vektoru dotazu s jednou maticí a porovnání s vektory ostatních dokumentů z kolekce, což již netrvá nijak závratně dlouho. Každopádně z tohoto důvodu byla doimplementována do balíčku JAMA metoda násobení matice a vektoru, který je reprezentován jako jednorozměrné pole datového typu *double*.

4.3 PostgreSQL

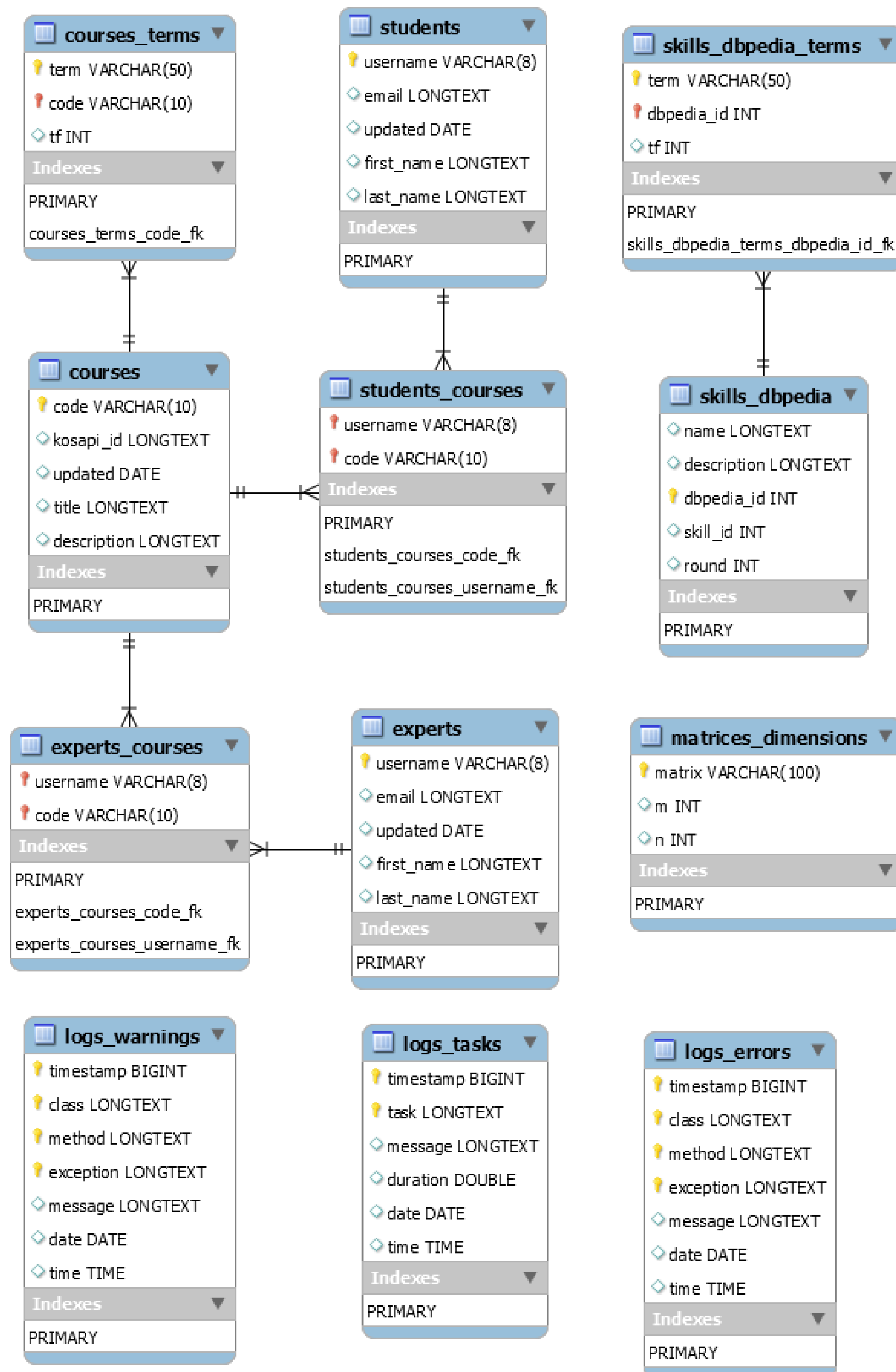
Pro ukládání dat byla vybrána objektově-relační databáze PostgreSQL, která implementuje většinu ze standartu SQL:2011 [8], dodržuje ACID a pro zabránění uváznutí používá tzv. MVCC (multiversion concurrency control). Jedná se o multiplatformní software běžící na mnoha operačních systémech včetně Linuxu, FreeBSD, Solaris a Microsoft Windows.

Strukturu databáze můžeme vidět na obrázku 4.1, kde jsou zachyceny vztahy mezi jednotlivými entitami z databáze.

Pro data získaná z KOSapi existují jednotlivé entity (*courses*, *students*, *experts*). Propojovací entity *students_courses* a *experts_courses* nám vyjadřují jaké předměty student absolvoval resp. jaké předměty vyučující přednáší, cvičí, zkouší nebo garantuje. A entita *courses_terms* nám říká, jaké termy a s jakou frekvenci obsahují jednotlivé předměty.

Skilly, jejichž data se získávají z DBpedie mají obdobnou strukturu jako kurzy, tj. existuje hlavní entita *skills_dbpedia* pro jednotlivé záznamy získané z DBpedie postupem uvedeným v sekci 3.3.1 a entita *skills_dbpedia_terms*, která obsahuje jednotlivé termy, opět s jejich frekvencí výskytu v daném záznamu z DBpedie.

Třetí a poslední skupinou jsou entity, které se v systému využívají pro logování (*logs_warnings*, *logs_tasks* a *logs_errors*) a entita *matrices_dimensions*, která slouží pro ukládání rozměrů matic. Matice ukládáme do souborů zkopírováním obsahů menších souborů, do kterých je matice mapována, pokud s



Obrázek 4.1: EER Diagram databáze

ní aktuálně pracujeme. Pro jejich uložení a pozdější načtení potřebujeme znát jejich rozměry, proto si je ukládáme do databáze.

Jednotlivé dokumenty jsou do databáze ukládány již po stemování, tj. získáme jednotlivé termy dokumentu spolu s jejich frekvencí. Tohoto můžeme využít, protože pracujeme s vektorovou reprezentací dokumentů, kde každá složka představuje daný term. Díky tomu tokenizujeme a stemujeme dokument pouze jednou, a to při aktualizaci databáze.

4.4 Spring Framework

Spring Framework je aplikační framework pro Javu. Jádro frameworku se dá použít pro jakoukoliv aplikaci psanou v Javě, pro nás je ale důležité, že obsahuje rozšíření pro psaní a vývoj webových Java EE aplikací. Jedná se o open source framework, který se stal od svého vzniku v roce 2002 mezi vývojáři velmi populární.

Jádro Spring Frameworku je postaveno na využití návrhového vzoru *Inversion of Control* a označujeme ho jako *IoC kontejner*. Výhodou tohoto návrhového vzoru je, že přesouvá odpovědnost za vytváření a provázání objektů na samotný framework, jinými slovy nemusí se o to starat aplikace. Objekty pak získáváme prostřednictvím tzv. *Dependency Injection*, která se stará o samotný způsob vložení objektů. Takto „vložené“ objekty nazýváme anglickým slovem *Bean*. Vytvoření těchto *Bean* může probíhat na základě konfiguračního souboru ve formátu XML, který obsahuje definice jednotlivých *Bean*, nebo na základě anotací přímo v kódu, kdy není žádný konfigurační soubor zapotřebí. Toto nastavení pomocí anotací využijeme v našem případě.

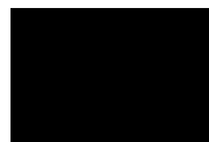
4.5 Apache Maven

Apache Maven je nástroj pro správu, řízení a automatizaci buildů aplikací, převážně používaný pro projekty psané v Javě. Maven se zabývá dvěma hlavními aspekty vytváření softwaru: za prvé jak bude software vybuildován a za druhé popisuje jeho závislosti.

Projekt je popsán pomocí *Project Object Model* souboru (zkráceně *POM* souboru) s XML strukturou. Právě v tomto souboru jsou definovány závislosti na externích knihovnách, dále popisuje proces buildování softwaru, můžeme v něm definovat konstanty, které pak využívají jednotlivé pluginy. Soubor je uložen v kořenovém adresáři projektu pod názvem *pom.xml*.

4.6 Apache Tomcat

Apache Tomcat je open source webový server a servlet kontejner. Je založen na jazyce Java a implementuje řadu Java EE specifik jako jsou *Java Servlets*, *Java JavaServer Pages* (*JSP*), *Java EL* a *WebSockets*.

KAPITOLA 5 

Analytické metody

5.1 Úvod

V této kapitole se podíváme na implementační záležitosti analytických metod. Pod tímto pojmem rozumíme metody založené na algoritmech data a text miningu, které jsou teoreticky popsány v prvních dvou kapitolách 1 a 2.

5.2 Pojmy

Nejdříve je nutné si vysvětlit pojmy, které budeme v souvislosti s analytickými metodami používat. Všechny tyto pojmy jsou reprezentovány svojí vlastní třídou, která má určité vnitřní vlastnosti a metody.

5.2.1 Term

Term je základní tvar slova, který získáme po stemování. Pro nás je důležitá hodnota termu, ale také frekvence, s jakou se objevuje v dokumentu (nebo v kolekci), hloubka (tj. první výskyt daného termu od počátku dokumentu) a váha termu, což je hodnota, kterou získáme po použití lokální 1.1.1 a globální metodiky 1.1.2 vážení termů.

Objekt termu můžeme použít pro reprezentaci termu uvnitř dokumentu, nebo jako reprezentaci termu v rámci celé kolekce dokumentů. Vždy se jedná o ten samý term, jen hodnota frekvence se jednou vztahuje k dokumentu a podruhé ke kolekci, tj. jednou frekvence vyjadřuje, kolikrát se term vyskytuje v dokumentu, a v druhém případě vyjadřuje, kolik dokumentů tento term obsahuje.

5.2.2 Dokument

Dokument je jedním z hlavních stavebních kamenů, na kterém analytické metody stavějí. Dokument na počátku celého procesu má pouze svůj název a textový obsah. My ho opatříme dalšími vlastnostmi: mapou termů a mapou tzv. značek, kterou získáme po *POS taggingu*, počtem slov, maximální frekvencí termu (tj. nejvyšší hodnota frekvence termu ze seznamu termů), maximální délkou (tj. hodnota délky nejdelšího slova z dokumentu) a na konci celého procesu mu stanovíme relevanci k dotazovanému dokumentu.

Jak bylo řečeno v sekci 4.3, v databázi máme uloženy jednotlivé termy pro jednotlivé dokumenty, tím pádem po většinou nás ani nebude původní textový obsah dokumentu zajímat, protože pracujeme rovnou s termy .

5.2.3 Kolekce dokumentů

V kolekci dokumentů jsou uloženy všechny dokumenty (samozřejmě pro každou entitu máme vlastní kolekci) ve formě seznamu. Společně s dokumenty jsou zde uloženy v mapě termy kolekce. V tomto případě je využit stejný objekt, který reprezentuje term v rámci dokumentu, jen frekvence zde vyjadřuje počet dokumentů, ve kterých se term objevuje, jinými slovy frekvenci výskytu napříč kolekcí dokumentů.

Objekt, který kolekci dokumentů reprezentuje, má v sobě navíc uloženou *Term-by-Document* matici a vektor dotazu, lépe řečeno dotazovaného dokumentu. Každá entita má svojí vlastní kolekci, to převážně kvůli tomu, že dokumenty pro jednotlivé entity jsou v databázi reprezentovány jiným způsobem. Pro kurzy to je přímo tabulka *courses_terms*, pro studenty a experty získáme termy ze stejné tabulky tj. *courses_terms*, ale tím že ze spojovací tabulky *students_courses* (*experts_courses*) zjistíme, jaké předměty student absolvoval, resp. které předměty expert cvičil, přednášel, vyučoval nebo garantoval a pro skilly je to opět přímo tabulka *skills_dbpedia_terms*. Abychom docílili toho, že každá kolekce bude svoje dokumenty načítat jiným způsobem, jsou objekty, reprezentující kolekce jednotlivých entit, podtřídami abstraktní třídy reprezentující kolekci dokumentů. Tato abstraktní třída implementuje většinu metod, které jsou pro všechny kolekce společné, a jednotlivé podtřídy pouze implementují právě odlišné metody pro načítání dokumentů z databáze.

5.3 Zpracování dokumentu

Původně je dokument opatřen pouze názvem a textovým obsahem. Naším prvním úkolem je z tohoto textového obsahu získat seznam termů, který dokument obsahuje.

5.3.1 Tokenizace

První, co v procesu získávání termů provedeme, je tzv. tokenizace, což znamená, že textový obsah dokumentu rozdělíme podle daného separátoru na tokeny. Separátorem není myšlena pouze mezera, ale veškerá interpunkční znaménka (tečka, čárka, středník, dvojtečka, vykřičník, otazník, pomlčka, uvozovky, závorky a další). Tím dojde nejen k získání jednotlivých tokenů, ale také k odstranění interpunkčních znamének.

Z takto získaných tokenů odstraníme tzv. *stop words*, což jsou slova, při jejichž vypuštění neztratíme význam věty nebo souvětí. Jedná se o slova, která nejsou nositeli významu ve větě, což jsou například spojky, předložky, zájmena, ale i třeba slova jako ano a ne. Už zde v tokenizaci se snažíme snižovat dimenzi výsledných matic. Stejně jako zmíněné *stop words* odstraníme i numerické hodnoty opět ze stejného důvodu, nejsou nositeli významu ve větě a dojde tak ke snížení dimenze. Poslední krok v tokenizaci je spíše zajímavostí, jedná se o odstranění „prázdných“ tokenů. To jsou tokeny, které vznikly při rozdělování řetězce na jednotlivé tokeny, ale přitom neobsahují žádnou hodnotu. Tokeny jsou uloženy do seznamu, který je využit pro stemování.

5.3.2 Stemování

Stemování je proces, při kterém je z daného slova získán jeho základní tvar. Pro účely této aplikace byl využit open source český stemmer, který vytvořil *Dolamic Ljiljana* z *University of Neuchatel*. Tento stemmer odstraňuje ze slov koncovky. Toto odstranění koncovek funguje na principu délky slova a hodnoty koncovky. Jinými slovy podle délky slova si najdeme koncovky, a pokud slovo některou z uvedených koncovek obsahuje, tak ji odstraníme. Odstraněny jsou koncovky, které vyjadřují pád, přivlastnění, zdrobněliny slova a koncovky, které vznikají odvozením slova (např. koncovka „-obinec“).

Každý token je ostemován a uložen do mapy termů. Při tomto ukládání je počítána frekvence termu a jeho hloubka v dokumentu, tyto hodnoty jsou používány ať už v automatické extrakci tagů, nebo při výtváření *Term-by-Document* matice. Jelikož používáme pro reprezentaci tagů v dokumentu mapu, jednoduše zjistíme, zda již daný tag je v mapě, pokud ano, zvýšíme jeho frekvenci o jedničku, a pokud není, uložíme ho do mapy s frekvencí jedna.

5.4 Vytvoření Term-by-Document matice

Term-by-Document matice, neboli matice výskytů, která je teoreticky detailně popsaná v sekci 1.4.1.2, je základním kamenem pro výpočet LSA. Každá buňka z této matice obsahuje váhu termu v daném dokumentu vypočtenou na základě lokální 1.1.1 a globální 1.1.2 metodiky vážení termu.

Jak tuto matici získáme? Nejdříve je nutné spočítat váhu každého termu v rámci dokumentu. V našem případě používáme metodiku *LogEntropy*, pro připomenutí si uvedeme vzorec 5.2 pro výpočet váhy pro jednu buňku matice.

$$g_i = 1 + \sum_j \frac{p_{ij} \log p_{ij}}{\log n}, \text{ kde } p_{ij} = \frac{tf_{ij}}{gf_i} \tag{5.1}$$

$$a_{ij} = g_i \log(tf_{ij} + 1) \tag{5.2}$$

Kde tf_{ij} je počet výskytů termu i v dokumentu j, gf_i je počet výskytů termu i v celé kolekci dokumentů a n je počet dokumentů v kolekci.

Co to pro nás znamená? Lokální příspěvek $\log(tf_{ij} + 1)$ získáme z vlastnosti frekvence objektu daného termu, kterou jsme vypočítali při stemování a ukládání stemů do databáze. Co se gf_i týče, tak zde musíme projít celou kolekci dokumentů a sečíst hodnoty frekvence termu ve všech dokumentech z kolekce, kde se daný term vyskytuje, čímž získáme celkový počet výskytů termu ve všech dokumentech.

Nyní už zbývá opět projít všechny dokumenty, vypočítat hodnotu p_{ij} a dále vypočítat hodnotu zlomku $\frac{p_{ij} \log p_{ij}}{\log n}$ pro každý dokument a tuto hodnotu načítat od nuly skrz všechny dokumenty. Tímto procesem jsme získali globální část vzorce a stačí už jen lokální příspěvek vynásobit s globálním příspěvkem.

Tímto procesem získáme váhu každého termu vztaženou k dokumentu, kde se vyskytuje. Díky naší vnitřní reprezentaci hodnot matice pomocí *MappedFileBuffer*, matici *Term-by-Document* inicializujeme jako prázdnou, tj. všude budou nuly datového typu *double*. A postupným projitím všech dokumentů a termů v nich nastavíme jednotlivé její buňky. Tím, že procházíme pouze termy, které dokument obsahuje, zdaleka nemusíme procházet všechny buňky *Term-by-Document* matice, jelikož *Term-by-Document* matice je velmi řídká, neboli většina dokumentů obsahuje pouze zlomek z celkového počtu termů.

5.5 Výpočet LSA

Nebudeme zde znovu podrobně vypisovat, co to LSA je, jak se počítá a jaké algebraické vzorce u toho využijeme, to vše se dočteme v sekci 1.4. Zde se budeme soustředit na implementační záležitosti této techniky. Ještě je nutné zdůraznit, že výpočet LSA se bude využívat pouze pro entity: kurz, student, expert a skill. Jinými slovy pro automatickou extrakci tagů samozřejmě není nutné LSA počítat.

Při výpočtu LSA vycházíme z *Term-by-Document* matice. Z této matice také vypočítáme nejprve parametr *k*, který nám určuje dimenzionální redukci výsledných matic, které získáme po výpočtu SVD. Jelikož používáme *LogEntropy* pro určení váhy jednotlivých buněk *Term-by-Document* matice, můžeme

použít Frobeinovu normu. Pro pořádek si znovu ukážeme vzorec 5.3, podle kterého jí vypočítáme.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\|a_{ij}\|^2} = \sqrt{trace(A * A)} = \sqrt{\sum_{i=1}^{min(m,n)} \sigma_i^2} \tag{5.3}$$

Z výše uvedeného vzorce je možné vidět, že máme na výběr, jakým způsobem tuto hodnotu získáme. Vzhledem k tomu, co tu již bylo zmíněno, je násobení matic paměťově velice zatěžující. Nejvýhodnější je využít sumy kvadrátů váhy všech buněk v matici. Tím získáme parametr k, který určuje dimenzionální redukci.

Jelikož se stále jedná o automatický výpočet, jsou v konfiguračním souboru nastaveny meze této dimenzionální redukce. Jinými slovy máme možnost nastavit minimální a maximální dimenzionální redukci. Nikde není řečeno, že nemůže nastat situace, kdy nám parametr k vyjde vyšší, než je původní dimenze *Term-by-Document* matice. Tím pádem by k žádné redukci nedošlo, proto je zde uvedena právě hodnota minimální dimenzionální redukce, abychom přešli do sémantického prostoru. A stejně tak může nastat situace, kdy hodnota parametru k vyjde v řádech jednotek, nebo desítek, což opět není žádoucí, protože pokud bychom zredukovali matice na dimenzi v řádech jednotek, získáme malý počet konceptů, do kterých by mohly spadat jednotlivé dokumenty. Z tohoto důvodu je stanovena maximální dimenzionální redukce.

5.5.1 Výpočet SVD

Následujícím krokem je výpočet SVD. Jak již bylo zmíněno, balíček JAMA má v sobě výpočet SVD implementovaný. Jelikož nás zajímá pouze relevance dokumentů, budou nám z SVD stačit pouze matice S a U. Protože pro výpočet *Concept-by-Document* matice potřebujeme inverzní matici matice S, musíme zajistit, abychom tuto matici mohli zkonstruovat, jinými slovy musíme zajistit, aby matice nebyla singulární, tj. aby její determinant nebyl rovný nule a inverzní matice šla zkonstruovat. Zde využijeme toho, že už z definice LSA dochází k dimenzionální redukci. V podstatě provádíme dimenzionální redukci v nekonečné smyčce, kde v každém kroku provedeme snížení dimenze matice o jedna, dokud není determinant rozdílný od nuly. Tím získáme konečnou hodnotu parametru k, na jehož základě provedeme dimenzionální redukci i matice U.

5.5.2 Výpočet Concept-by-Document matice

Následuje výpočet *Concept-by-Document* matice, kterou vypočteme ze vztahu $S^{-1}U^T$. Tuto matici používáme pro dotazování na relevanci dokumentu podle následujícího vztahu $S_k\hat{d}_j$ a $S_k\hat{q}$. Pro připomenutí si uvedeme ještě vzorec 5.4,

kde je vidět převedení vektoru dokumentu do nízko dimenzionálního prostoru. Na uvedeném vzorci vidíme, že jeho část je právě naše *Concept-by-Document* matice.

$$\hat{d}_j = S_k^{-1} U_k^T d_j \tag{5.4}$$

Jelikož tento převod potřebujeme pokaždé, když počítáme relevanci dokumentů, není důvod, proč si tento vektor pro každý dokument z kolekce nepředpočítat. To samé, co jsme si řekli o převodu vektoru do nízko dimenzionálního prostoru, platí i o přípravě vektoru pro dotazování relevance $S_k\hat{d}_j$. Dokonce si můžeme uložit až výsledný vektor, který vznikne po převedení do nízko dimenzionálního prostoru a následné přípravě k porovnání podobnosti dokumentů.

5.5.3 Výpočet Term-by-Document low-dimensional space matice

V rámci paměťové úspory uložíme tyto vektory do podoby matice, která má rozměry: počet řádku matice *Concept-by-Document* $\times$ počet dokumentů v kolekci. Děláme to hlavně z důvodu, že v paměti je mapován pouze jeden *MappedFileBuffer*, což je paměťově výhodnější, než kdybychom každý dokument opatřili ať už vektorem reprezentovaným polem datového typu double, nebo vektorem, který bychom uložili do objektu datového typu *Matrix*. V druhém případě by také nedošlo k žádné úspoře, sice bychom jednotlivé vektory měli uloženy v souborech, ale samotné mapování těchto souborů by v paměti zabralo více místa než mapování jednoho většího souboru, ve kterém bude uložena výsledná matice. Díky tomuto předpočítání ušetříme čas při počítání relevance dokumentů. Výslednou matici nazveme *Term-by-Document low-dimensional space* matice.

5.6 Výpočet relevance dokumentů

Jak už bylo řečeno, budeme porovnávat vektory $S_k\hat{d}_j$ a $S_k\hat{q}$. Toto porovnání proběhne pro všechny dokumenty v kolekci a dotazovaný dokument. Pro dokumenty z kolekce máme již hodnotu $S_k\hat{d}_j$ vypočtenou v matici *Term-by-Document low-dimensional space*, a to konkrétně v j-tém sloupci této matice. Tudíž musíme provést převedení do nízko dimenzionálního prostoru pouze pro dotazovaný dokument, což provedeme pomocí následujícího vzorečku.

$$\hat{q} = S_k^{-1} U_k^T q \tag{5.5}$$

A následně tento vektor $\hat{q}$ vynásobíme maticí S_k. Výsledný vektor si předpočítáme před samotným projitím celé kolekce dokumentů a výpočtem rele-

vance jednotlivých dokumentů, abychom ho nemuseli počítat neustále dokola při porovnání dotazovaného dokumentu s jednotlivými dokumenty z kolekce. V tuto chvíli naopak pro urychlení výpočtu používáme pro tento vektor reprezentaci pomocí pole datového typu *double*.

5.6.1 Kosínová podobnost

Pro výpočet relevance byla zvolena Kosinová podobnost, což je míra podobnosti dvou vektorů, která se získá výpočtem kosinu úhlu těchto vektorů. Jinými slovy tuto podobnost vypočteme jako skalární součin těchto vektorů, vydělený součinem jejich velikostí, jak je uvedeno ve vzorci 5.6. Porovnáváme vektory $S_k\hat{d}_j$ a $S_k\hat{q}$. Vektory $S_k\hat{d}_j$ jsou sloupce matice *Term-by-Document low-dimensional space* a vektor $S_k\hat{q}$ je náš výsledný vektor dotazovaného dokumentu. Vektor d ve vzorci představuje finální vektor dokumentu z kolekce dokumentů a vektor q je finální vektor dotazovaného dokumentu.

$$sim_C = \frac{d \cdot q}{\|d\| \, \|q\|} = \frac{\sum_{i=1}^{n} d_i \times q_i}{\sqrt{\sum_{i=1}^{n} (d_i)^2} \sqrt{\sum_{i=1}^{n} (q_i)^2}} \tag{5.6}$$

Tato vypočtená míra kosinové podobnosti je naše relevance j-tého dokumentu vzhledem k dotazovanému dokumentu. Tímto krokem je ukončena analytická práce na relevanci dokumentů, o seřazení a reprezentaci výsledku se již stará část aplikace, která je popsána v kapitole 6.

5.7 Automatická extrakce tagů

Pro automatickou extrakci tagů budeme potřebovat hodnotu *TF-IDF* pro dané slovo. Mohlo by se zdát, že je nutné vypočítat celou *Term-by-Document* matici stejně, jak je uvedeno v sekci 5.4, jen za použití jiné metodiky pro vážení termů a tudíž podle jiného vzorce 5.7. Nicméně to není zapotřebí, jelikož potřebujeme pouze hodnotu *TF-IDF* pro slova, která se vyskytují v dotazovaném dokumentu. Veškeré informace nutné pro tento výpočet máme uložené v objektu, který reprezentuje kolekci dokumentů.

$$TF\text{-}IDF = tf_{ij} * \log_2 \frac{n+1}{df_i + 1} \tag{5.7}$$

Kde tf_{ij} je počet výskytů termu i v dokumentu j, df_i je počet dokumentů, ve kterých se term i vyskytuje, a n je počet dokumentů v kolekci.

5.7.1 POS tagging

K opatření jednotlivých slov značkami byl využit *RDRPOSTagger* od autorů: Dat Quoc Nguyen, Dai Quoc Nguyen, Dang Duc Pham, a Son Bao Pham. Pro tento tagger je připravený naučený model pro český jazyk na základě korpusu *Prague Dependency Treebank 2.5*. Tagger v sobě má jednoduchý tokenizer, který vstupní sekvenci slov rozdělí na jednotlivé tagy a tyto tagy jsou na základě modelu opatřeny značkou. Nicméně tento tokenizer neodstraňuje tzv. *stop words*, tudíž nám ve výsledku zůstávají veškeré spojky, předložky, zájmena a další. Tento fakt se nám nicméně pro automatickou extrakci hodí, protože bereme v potaz i slovní spojení, která právě spojky a předložky obsahují. Celý následující proces vychází z těchto tagů, které označkuje *RDRPOSTagger*.

5.7.2 Výpočet skóre tagu

Skóre pro jednotlivé tagy vypočteme podle vzorce 5.8, kde *Len* je metrika na zohlednění délky termu a *Dep* je pozice prvního výskytu daného termu. Více o teorii stanovení *Score* tagu se dočteme v sekci 2.2.2. *POS tagging* nám vrací seznam tagů a my zde mluvíme o termech. To z toho důvodu, že jak *TF-IDF*, tak hodnoty *Len* a *Dep* máme stanoveny pro termy. Proto je nutné provést na tagu stemování a na základě tohoto stemu získáme objekt termu, který obsahuje potřebné hodnoty.

$$Score = TF\text{-}IDF * Len * Dep \tag{5.8}$$

Tokenizer *RDRPOSTaggeru* neodstraňuje *stop words*, díky tomu může nastat situace, že daný term neexistuje (jedná se pravděpodobně o spojky, zájmena atd.). Jinými slovy jde o slova, která nemají stanovenou *TF-IDF*, *Len* ani *Dep*. Jejich *Score* určíme na základě vzorce 5.9.

$$Score = Len * Dep \tag{5.9}$$

Hodnotu *Len* vypočteme podle vzorce 5.10. Na vzorci je důležité si všimnout dvou věcí. Pokud je délka slova rovna jedné, pak je hodnota *Len* rovná nule a tudíž i hodnota *Score* pro daný tag. Na rozdíl od termů nemáme ve jmenovateli dvojkový logaritmus, to z toho důvodu, abychom tyto tagy „znevýhodnili“. Hodnotu *Dep* pak vypočítáme jako podíl čísla jedna a aktuální pozice daného slova.

$$Len = \frac{\log_2(\text{Délka slova})}{\text{Maximální délka termu v dokumentu}} \tag{5.10}$$

Toto stanovení skóre tagu provedeme pro každý tag, který jsme získali z *POS taggingu*.

5.7.3 Vytvoření N-gramů

Zatím jsme ve stádiu, kdybychom mohli tagy opatřené skórem seřadit dle relevance a získali bychom automaticky extrahované tagy, ale pouze délky jedna. Proto nyní vytvoříme seznam N-gramů, který bude obsahovat posloupnosti původních tagů o délce jedna, dva a tři. U délky jedna, se jedná prakticky jen o překopírování původních tagů do nového seznamu. Co se délky dva a tři týče, vytvoříme posloupnosti sousedních tagů o délce daného N-gramu.

Při vytváření N-gramu nastavíme objektu, který daný N-gram reprezentuje, množinu pozic. Tím máme na mysli pozice jednotlivých tagů, které nám vrátil *RDRPOSTagger*. Těchto pozic budeme později využívat při odstraňování duplicit v seznamu výsledných tagů.

5.7.4 Výpočet relevance tagu

Těmto N-gramům, které se pro nás stávají novými tagy, stanovíme relevanci podle vzorce 5.11, kde n je délka N-gramu, $Score_i$ je ohodnocení jednotlivých tagů v N-gramu a $Score_{seq}$ je relevance daného tagu. Proč vzorec pro výpočet relevance má daný tvar se dočtete v sekci 2.2.3.

$$Score_{seq} = \frac{\max(Score_i) + n \cdot \min(Score_i)}{n + 1} \tag{5.11}$$

Jedná se o nalezení dvou hodnot (minima a maxima z N-gramu), na základě kterých vypočteme relevanci. Ve stejném cyklu, kde stanovujeme ohodnocení celému tagu, zjišťujeme, zda se v daném tagu vyskytuje podstatné jméno. Protože známe strukturu značky, bude nám k tomu stačit regulární výraz. Pokud daný tag podstatné jméno neobsahuje, ze seznamu tagů ho odstraníme. Nicméně žádný naučený model není dokonalý. Proto máme možnost stanovit výjimky pro slova, která nejsou *POStaggerem* označeny jako podstatná jména, přestože my víme, že podstatná jména jsou. Tyto výjimky jsou uloženy v souboru, ze kterého získáme mapu těchto podstatných jmen, a následně kontrolujeme zda se dané slovo v mapě slov vyskytuje.

Zároveň zvýhodníme tagy, které obsahují podstatné jméno v prvním pádu. Opět ze struktury značky víme, na jakém místě se vyskytuje hodnota pádu, stačí nám získat *substring* a ten převést na číselnou hodnotu. Samotné zvýhodnění funguje na principu znevýhodnění tagů, které podstatné jméno v prvním pádu neobsahují. Takovéto tagy vynásobíme hodnotou nižší než jedna, tím dojde k jejich znevýhodnění.

Portál Spolupráce s průmyslem si eviduje všechny použité tagy pro jednotlivá zadání, toho můžeme využít pro poslední úpravu relevance tagů. Vynásobíme relevanci tagu konstantou, kterou je možné měnit v konfiguračním souboru na základě toho, zda se již daný tag v portálu Spolupráce s průmyslem vyskytuje. Pokud danou konstantu nastavíme na hodnotu rovnu jedné, pak tento krok budeme ignorovat.

5.7.5 Odstranění duplikátů tagů

V sekci 2.2.4 je popsáno, proč a na základě čeho „duplikáty“ odstraňujeme. Co se týče samotné implementace, budeme potřebovat tři množiny a jeden seznam pro vrácené relevantní tagy. První množina bude reprezentovat pozici (*positions*), druhá bude reprezentovat jednotlivé stemy ze slov tagů (*duplicates*) a třetí bude reprezentovat celé tagy (*duplicatesSeq*).

Nejdříve je nutné spočítat výskyty jednotlivých tagů. To uděláme pomocí mapy, kam ukládáme textové reprezentace jednotlivých tagů a počet jejich výskytů. Výskyty počítáme průchodem seznamu tagů. Při druhém průchodu seznamu tagů nastavíme hodnotu výskytů, kterou získáme z mapy výskytů.

Jednotlivé tagy máme seřazeny ve vstupním seznamu podle jejich relevance. V cyklu z každého tagu získáme jednotlivé jeho termy. Při stemování rovnou zjišťujeme, zda se v množině daný stem již vyskytuje. Na základě tohoto zjištění následuje rozhodovací fáze, která má tři stavy:

- Žádný ze stemů zatím v množině *duplicates* neexistoval a zároveň tag neexistuje v množině *duplicatesSeq*.
- Tag existuje v portálu Spolupráce s průmyslem a zároveň tag neexistuje v množině *duplicatesSeq*.
- Nebyly splněny předchozí podmínky, navíc délka N-gramu je větší než jedna a zároveň tag neexistuje v množině *duplicatesSeq*.

V prvních dvou případech víme, že se jedná o relevantní tagy a přidáme je do seznamu výsledných relevantních tagů. Zároveň do množiny *duplicates* přidáme jednotlivé stemy slov z tagu, textovou hodnotu celého tagu vložíme do množiny *duplicatesSeq* a z objektu reprezentující tag získáme jeho množinu pozic, kterou sloučíme s množinou *positions*.

Ve třetím případě, kdy některý ze stemů (nebo všechny) byl již obsažen v množině *duplicates*, musíme rozhodnout, zda daný tag budeme považovat za relevantní. Jak bylo uvedeno v sekci 2.2.4, určíme tak na základě absolutní hodnoty rozdílu maximální a minimální hodnoty *Score* tagu z N-gramu. Protože víme, že maximální hodnota je vždy větší nebo stejná než minimální hodnota *Score*, stačí nám odečíst maximální hodnotu od minimální.

Pokud je výsledná hodnota po odečtení nižší než minimální hodnota *Score* tagu z N-gramu, budeme tento tag brát jako relevantní. Posledním kontrolním bodem je zjištění, zda se nějaká pozice z daného tagu neobjevuje v množině *positions*. Pokud ano, tak jsme již obdobný tag s větší hodnotou relevance do seznamu relevantních tagů zařadili. Pro lepší ilustraci můžeme porovnávat tagy, které začínají na stejné pozici a jeden je délky dva a druhý délky tři. Ten s délkou dva má větší relevanci a již ve výsledném seznamu je, ten o délce tři má relevanci nižší, ale vyjadřuje něco velmi podobného, proto ho do relevantních tagů dávat nebudeme.

V předchozím odstavci jsme si řekli, jak postupujeme, pokud na základě absolutní hodnoty bereme tag jako relevantní. Pokud tomu tak není, ještě zohledníme počet výskytů daného tagu (posloupnosti slov). Tím dosáhneme toho, že i tagy, které mají velmi rozdílné maximální a minimální *Score*, budou považovány za relevantní, pokud se objevují ve více jak jednom procentu z původního seznamu tagů.

Tímto krokem je i zde ukončena analytická práce na automatické extrakci tagů z dokumentu. O reprezentaci výsledku se již stará část aplikace, která je popsána v kapitole 6.

KAPITOLA 6

Webové služby

6.1 Úvod

Jak bylo řečeno v sekci 3.1, v systému evidujeme pět základní entity: kurz, student, expert, skill a tag. Webové služby jsou navrženy tak, že každá tato entita má vystavenou svojí službu pro vrácení seznamu dokumentů dané entity seřazených dle relevance k přijatému zadání (dotazovanému dokumentu). Tyto služby budeme nazývat jako *služby relevance*, a zároveň každá z entit má druhou službu, která slouží k aktualizaci databáze a výpočtu LSA (až na službu obsluhující tagy), které budeme nazývat *výpočetní služby*.

6.2 Controllery

Každá entita má svůj vlastní *controller*, který slouží pro přijímání požadavků. Jedná se o třídu, která je anotována jako *RestController*. Pomocí anotací je v jednotlivých *controllerech* nastaveno i mapovaní, jak celého *controlleru*, tak jednotlivých metod. Spring Framework se stará o vytváření a provázání objektů, my pouze určujeme, zda dané objekty budou typu *singleton* nebo *prototype*. Jinými slovy zda bude jejich instance existovat pouze jednou, nebo se vytvoří pokaždé, když budeme po Spring Framewroku požadovat přístup k dané *Beaně*. Prvního způsobu budeme využívat u všech managerů pro přístup do databáze, nebo např. pro jednotlivé modely, které se starají o zpracování přijatého požadavku. Naopak *prototype* využijeme např. pro objekty reprezentující kolekci dokumentů nebo objekty provádějící výpočet LSA.

Proč budeme využívat právě *prototype*? Je to převážně z důvodu konkurenčního zápisu, který by mohl nastat při výpočtu LSA. Jinými slovy, pokud nevyužijeme *prototype*, tak na jednom místě budeme upravovat něco, s čím pracujeme na místě jiném, což by v nejhorším případě vyvolalo chybu konku-

renčního zápisu, v tom lepším bychom pouze získali výsledek, který neodpovídá skutečnosti, což je ale úplně stejně nežádoucí.

6.3 Služby relevance

Služby relevance jsou vždy mapovány na relativní adresu */[entita]* s tím, že za entitu dosadíme vždy danou entitu, konkrétně pak jedno z následujících: *courses*, *skills*, *students*, *experts* nebo *tags*. Relativní adresa pro získání kurzů seřazených podle relevance bude vypadat */courses*.

Tyto služby očekávají HTTP metodu *POST*, která ve svém těle nese informace o konkrétním zadání v JSON podobě, ke kterému bude vztažen výsledek seřazených dokumentů dle relevance. Při výchozím nastavení služba vrací všechny dokumenty dané entity z kolekce dokumentů. Nicméně můžeme využít dvou parametrů: *limit* a *offset*, které nám určují počet vypsaných dokumentů a od kterého dokumentu bude výpis začínat. Pokud chceme získat prvních deset nejrelevantnějších kurzů, adresa i s parametry bude vypadat následovně: */courses?limit=10*. Použití druhého parametru *offset* si budeme demonstrovat na příkladu vypsání pěti kurzů seřazených dle relevance začínajícího od šestého nejrelevantnějšího kurzu, tj. prvních pět kurzů „zahodíme“. Relativní adresa v tomto případě vypadá následovně: */courses?limit=5&offset=5*.

6.3.1 Model pro výpočet relevance dokumentů

Z přijatého zadání vytvoříme dokument, který společně s kolekcí dokumentů dané entity a objektem, který se stará o výpočet LSA, předáme modelu pro výpočet relevance dokumentů. V tomto modelu má každá entita implementovánu svojí metodu pro výpočet relevance dokumentů, která nám určí umístění matic, které budeme načítat pro výpočet relevance. Výpočet relevance se u jednotlivých entit liší pouze maticemi a kolekcí dokumentů. Díky tomu můžeme samotnou metodu, která se stará o výpočet relevance, implementovat pouze jednou pro všechny entity s tím, že umístění matic a kolekci dokumentů obdrží jako parametry.

V této metodě dojde k načtení kolekce dokumentů z databáze. K tomu dojde vždy pouze v případě, že objekt reprezentující kolekci dokumentů je nastaven na „reload“. Do stavu „reload“ je kolekce dokumentů nastavena pouze, došlo-li k aktualizaci databáze a novému výpočtu LSA. V opačném případě jsou již dokumenty načteny a můžeme tento krok přeskočit. Z předchozí metody známe umístění matic vypočtených LSA, ty musíme nyní načíst jak je uvedeno v sekci 4.1, a rozdělit do menších pracovních souborů. Dále je nutné provést stemování dokumentu zadání, po jehož provedení můžeme sestavit vektor dotazu.

6.3.1.1 Porovnávání dokumentů

K porovnání můžeme přistupovat dvěma způsoby. Ten první je, že dotazovaný dokument je sám považován za dokument z kolekce. Tudíž bychom ho měli začlenit do kolekce dokumentů, provést výpočet *Term-by-Document* matice a následně celý výpočet LSA. Výpočet LSA může trvat i několik desítek minut, tudíž by tento postup byl velice neefektivní. Proto zvolíme druhý přístup, kdy dotazovaný dokument je považován za dotaz nad kolekcí dokumentů a tudíž ho do kolekce jako takové nezařadíme. Tím odpadají výše zmíněné kroky. Vektor dotazovaného dokumentu je sestaven z termů a dokumentů z kolekce, protože u každého objektu, který reprezentuje term, resp. dokument, máme uloženy všechny potřebné informace pro sestavení tohoto vektoru (frekvence termu v dokumentu nebo v kolekci).

Nyní když máme vše připraveno může dojít k samotnému výpočtu relevance, který je popsán v sekci 5.6. Poté následuje důležitý krok uklizení paměti, tj. odstraníme z paměti všechny matice, s kterými jsme pracovali, k jejich opětovnému načtení do paměti dojde, až je budeme opět potřebovat. To znamená při dalším dotazu na relevanci dokumentů.

Posledním krokem je samotné seřazení dokumentů podle relevance a vytvoření seznamu, který obsahuje objekty typu *Result*. Tento seznam je předám modelu, který se stará o vytvoření odpovědi v konečné formě, neboli kdy dojde k uplatnění parametrů *limit* a *offset*. Zde jde jednoduše o získání podseznamu z poskytnutého seznamu. Musíme ohlídat, abychom nevyžadovali prvek, který je mimo rozsah pole, tím by došlo k vyvolání výjimky. Pokud má parametr *offset* větší hodnotu než je délka seznamu dokumentů v kolekci, vrátíme prázdný seznam. Vše co je uvedeno v této sekci platí pro entity: kurz, student, expert a skill.

6.3.2 Model pro automatickou extrakci tagů

Stejně jako tomu bylo u ostatních entit, i u entity tag, opět vytvoříme z přijatého zadání dokument. Tento dokument je předán modelu pro automatickou extrakci tagů, následně je ostemován a je na něm proveden *POS tagging*. Nyní načteme kolekci dokumentů z databáze, opět k načtení dojde pouze, pokud je objekt reprezentující kolekci databáze nastaven na „reload“.

Dalším krokem je už samotná automatická extrakce tagů, jejíž implementace je popsána v sekci 5.7. Výsledkem tohoto procesu je seznam tagů, kde každý tag je opatřen relevancí. Tento seznam seřadíme podle relevance a předáme modelu, který se stará o konečnou podobu odpovědi, tj. na výsledný seznam se aplikují hodnoty parametrů *limit* a *offset*, jak bylo popsáno v sekci 6.3.

6.3.3 Odpověď

Služba odesílá odpověď ve formátu JSON, ať už se jedná o chybové hlášení, nebo o úspěšně zpracovaný požadavek. Chybové hlášení nastává pouze v případě, že ještě není pro danou entitu spočtena LSA. V tomto případě je návratový status roven *500*.

V případě úspěšné odpovědi je vrácen JSON objekt, který má dvě vlastnosti: *status* a *result*, kde první z výše uvedených odpovídá HTTP návratovému kódu, v tomto případě *200 OK* s tím, že *status* obsahuje pouze číselnou hodnotu návratového kódu. *Result* v sobě obsahuje JSON pole JSON objektů, které reprezentují jednotlivé záznamy dané entity seřazené podle relevance. Tyto objekty mají vždy dvě vlastnosti *name* a *relevance*, kde vlastnost *name* obsahuje vždy identifikátor daného dokumentu. V případě kurzů je to kód kurzu, u studentů a expertů je to uživatelské jméno, u skillů je to ID, pod kterým jsou vedeny v portálu Spolupráce s průmyslem, a u tagů je to přímo hodnota daného tagu. Druhá vlastnost *relevance* obsahuje vypočtenou relevanci. Ukázku odpovědi můžete vidět na ukázce odpovědi 6.1.

```
1  {
2      "status": 200,
3      "result": [
4          {
5              "relevance": 0.562698078308097,
6              "name": "BI-GIT"
7          },
8          {
9              "relevance": 0.45727816659955867,
10             "name": "BI-ADW"
11         },
12         {
13             "relevance": 0.4513183545163601,
14             "name": "MI-SYB"
15         },
16         {
17             "relevance": 0.45058882253711935,
18             "name": "MI-EDW"
19         },
20         {
21             "relevance": 0.4459636755828946,
22             "name": "PI-KP"
23         }
24     ]
25 }
```

Listing 6.1: Ukázka odpovědi úspěšně provedeného požadavku

6.4 Výpočetní služby

Výpočetní služby jsou vždy mapovány na relativní adresu */[entita]/update*, opět za entitu dosadíme jedno z následujících: *courses*, *skills*, *students*, *experts* nebo *tags*. Relativní adresa pro aktualizaci kurzů a výpočet jejich LSA bude */courses/update*.

6.4.1 Odpověď

Tyto služby jako odpověď opět vrací JSON objekt, který má dvě vlastnosti *status* a *message*, kde první odpovídá HTTP návratovému kódu a druhý je zpráva v textové podobě. Z toho, jak jsou služby implementovány, jsou možné pouze dva návratové HTTP kódy. Jsou to kódy *200 OK*, pokud byl požadavek úspěšně přijat (ukázka 6.2), a *409 Conflict* (ukázka 6.3). Kdy k jakému navrátovému kódu dochází si povíme v sekci 6.4.2.

```
1 {
2     "status": 200,
3     "message": "Request was accepted and passed to
         processing"
4 }
```

Listing 6.2: Ukázka odpovědi úspěšně přijatého požadavku

```
1 {
2     "status": 409,
3     "message": "Updating is already in process"
4 }
```

Listing 6.3: Ukázka odpovědi, kdy již výpočet na dané entitě probíhá

6.4.2 Model pro aktualizaci entit

Už to zde v rámci tohoto textu bylo několikrát zmíněno, výpočet LSA a aktualizace databáze z externích zdrojů trvá vzhledem k ostatním procesům několika násobně déle, přičemž se můžeme dostat na dobu několika desítek minut. Je proto nežádoucí, aby klient čekal na odpověď, zda se vše povedlo,

po celou dobu aktualizace a výpočtu LSA. Proto je zde využita možnost, kterou nabízí Spring Framework, a to asynchronní metody. Asynchronní metody jsou takové, které Spring Framework spustí, ale klient pokračuje ve vykonávání kódu, aniž by čekal na ukončení asynchronní metody.

6.4.3 Asynchronní metody

Z pohledu klienta je tato asynchronicita dobrá věc, ale podívejme se na to z hlediska serveru a vůbec výpočetních prostředků. Pokud by byly tyto metody spouštěny opakovaně, došlo by brzy k vyčerpání místa v paměti. Je to ale vůbec zapotřebí? Jinými slovy se ptáme, jak často je nutné počítat znovu LSA? Odpovědí je, že stačí když dojde k aktualizaci externích zdrojů, tudíž určitě není třeba, abychom souběžně počítaly LSA stejné entity více než jednou. V podstatě se tomu snažíme zabránit.

6.4.4 Využití principu mutexů

Ten, kdo programoval v jazyku *C*, je určitě obeznámen s tzv. *mutexy*, které zabraňovaly v konkurenční práci se sdílenou pamětí. My využijeme podobného principu, jinými slovy si řekneme, že daná asynchronní metoda je v podstatě sdílenou pamětí a že chceme zamezit prací s ní více než jednomu klientovi naráz. Proto i my zavedeme *mutexy* pro jednotlivé entity, které aktivujeme, když vstoupíme do dané asynchronní metody, a pokud je daný *mutex* aktivní, nikdo další nemůže asynchronní metodu spustit. Zde se konečně dostáváme k možným odpovědím. Pokud je *mutex* neaktivní a naším zavoláním asynchronní metody se stane aktivním, dostaneme odpověď *200 OK* a spustí se výpočet LSA. Pokud je daný *mutex* v době našeho zavolání aktivní, dostaneme odpověď *409 Conflict* s hláškou, že výpočet již probíhá. O uvolnění *mutexu* se stará daná asynchronní metoda, která po skončení aktualizace a výpočtu LSA *mutex* uvolní.

Pokud je *mutex* neaktivní, klientovi je odeslána odpověď, že byl spuštěn výpočet a samotný výpočet a aktualizace byla delegována na model, který se o zmíněné věci stará. V tomto modelu má každá entita vždy svojí metodu, kde dojde k určení cest k souborům, kam se později uloží vypočtené matice, a dojde k aktualizaci databáze z externích zdrojů. Více o těchto externích zdrojích se dočtete v kapitole 3.

6.4.4.1 Parametr run

V tuto chvíli je ještě nutné zmínit parametr *run*, který je typu *boolean*. Tento parametr nám říká, zda máme asynchronní metodu a výpočet v ní spustit. Je to z toho důvodu, pokud bychom se chtěli zeptat, zda již je daná asynchronní metoda dokončena. V tu chvíli by bylo další spouštění aktualizace databáze a výpočtu LSA nežádoucí. Výchozí hodnota parametru *run* je *true*, což znamená, že pokud pošleme požadavek bez parametru *run*, tak dojde ke

spuštění asynchronní metody. Pokud je asynchronní metoda již dopočítána, odpověď v případě parametru *run* nastavenému na *false* vypadá podle ukázky 6.4. Pokud výpočet probíhá, obdržíme stejnou odpověď ať je parametr *run* roven *true*, nebo *false*.

```
{
    "status": 200,
    "message": "Request can be accepted and passed to
        processing"
}
```

Listing 6.4: Ukázka odpovědi úspěšně přijatého požadavku

6.4.5 Aktualizace databáze

Pro aktualizaci každé entity je připraven manager, který se o aktualizaci postará. V případě entit: kurz, student a expert, se budeme za tímto účelem připojovat ke KOSapi. Pro tento účel bylo nutné implementovat jednoduchou OAuth autorizaci. Jedná se o třídu, která zašle HTTP požadavek na URL adresu autorizačního serveru. V hlavičce *Authorization* odešle *client ID* a *client secret* naší aplikace, která jsou zakódována kódováním *Base64*. Odpovědí na tento požadavek ze strany autorizačního serveru je JSON objekt, který v sobě obsahuje vlastnost *acces_token*. Tento *acces_token* je pak posílán v jednotlivých požadavcích, které jsou směrovány na KOSapi, opět v hlavičce *Authorization* jako tzv. *Bearer*. KOSapi se už jen ujistí, zda tento token byl vydán autorizačním serverem.

6.4.5.1 Externí data z KOSapi

Samotné získávání dat z KOSapi je v podstatě parsování jednotlivých XML souborů. V případě kurzů nejdříve získáme seznam všech kurzů. V tomto kroku nás zajímá především kód kurzu a datum poslední úpravy kurzu v KOSu. V druhém kroku porovnáme kurzy, které máme v databázi s kurzy, které jsme získali z KOSapi. Pokud kurz v databázi není, nebo pokud datum poslední aktualizace v databázi je starší než datum poslední úpravy v KOSu, dojde k aktualizaci kurzu. Pod tím je myšleno dotázání se KOSapi na detail kurzu. V tomto detailu obdržíme informace, ze kterých je sestaven textový obsah dokumentu kurzu, který je následně ostemován, a jeho termy jsou uloženy do databáze i s jejich frekvencí. Samozřejmostí je úprava data poslední aktualizace daného kurzu v naší databázi.

Obdobný proces provedeme i v případě studentů a expertů pouze s tím rozdílem, že nepotřebujeme získat textový obsah dokumentu, protože ten je získáván až při načítání expertů a studentů z databáze na základě dokumentů

jednotlivých kurzů. Nejdříve získáme seznam všech studentů, resp. expertů s jejich uživatelským jménem a datem poslední úpravy v KOSu. Opět porovnáme data poslední aktualizace v databázi s datem poslední úpravy v KOSu a vzhledem k tomu provedeme úpravu úspěšně absolvovaných kurzů u studentů, resp. předměty, které expert cvičí, přednáší nebo garantuje.

6.4.5.2 Externí data z DBpedie

Dokumenty pro skilly jsou získány z DBpedie. Manager, který se o aktualizaci skillů stará, obdrží seznam aktuálních skillů v portálu Spolupráce s průmyslem. Pro každý skill pošleme na SPARQL endpoint DBpedie několik dotazů. Jejich počet se může pro každý skill lišit. Záznamy hledáme několika úrovňově s tím, že máme stanoven minimální počet nalezených záznamů. Pokud tohoto počtu dosáhneme v nižší úrovní hledání, již nepokračujeme. Porovnáváme vždy název skillu uvedený v portálu Spolupráce s průmyslem a název záznamu na DBpedii (*rdfs:label*) a popis záznamu (*dbpedia-owl:abstract*).

Jednotlivé úrovně vyhledávání záznamů:

- Hledáme přesnou shodu názvu skillu s názvem záznamu v DBpedii.
- Hledáme shodu názvu skillu s názvem záznamu v DBpedii a přesný výskyt názvu skillu v popisu záznamu v DBpedii.
- Hledáme v popisu záznamu na DBpedii všechny části názvu skillu.
- Hledáme v popisu záznamu na DBpedii alespoň nějaké části názvu skillu.

Ve výše uvedeném seznamu vidíme jednotlivé úrovně hledání záznamů na DBpedii. Ještě je nutné zmínit další sadu úrovní. V každé úrovni z této sady jsou vždy prohledávány všechny úrovně uvedené v seznamu. Přičemž v první úrovni s názvem skillu nic neděláme, v druhé úrovni odstraníme speciální charaktery (např. interpunkci, plus, mínus atd.) a v poslední úrovni odstraníme speciální charaktery a provedeme stemování všech jednotlivých částí názvu skillu.

Výsledkem je seznam záznamů z DBpedie, kde u každého si pamatujeme, jakému skillu z portálu Spolupráce s průmyslem náleží. Z tohoto seznamu záznamů vytvoříme seznam dokumentů. Každý dokument ze seznamu ostemujeme a jeho termy uložíme do databáze. Na rozdíl od zdrojů z KOSapi zde vždy nejdříve smažeme předchozí záznamy z databáze a vždy dojde k celkové aktualizace databáze. To z toho důvodu, že nevíme zda se nezměnil skill v portálu Spolupráce s průmyslem (nikde si nevedeme předchozí verzi seznamu skillů). Navíc může dojít k nalezení vhodnějších záznamů na DBpedii. Navíc pokud bychom rovnou všechny záznamy z databáze nevymazali, ani bychom neušetřili nijak výrazně čas, protože zároveň s každým SPARQL dotazem se vrací rovnou celý popis záznamu z DBpedii, na rozdíl od dat z KOSapi, kdy

se na detail daného záznamu ptáme až následně po porovnání dat aktualizace v KOSu a u nás v databázi.

6.4.6 Výpočet LSA

Po této aktualizaci databáze je spuštěna metoda, která se stará o výpočet LSA, opět pro všechny entity je tato metoda stejná, liší se pouze kolekcí dokumentů, kterou získáme z databáze, a místem, kam se budou ukládat výsledné matice.

V této metodě dojde k načtení dokumentů z databáze a vypočtení *Term-by-Document* matice. Nakonec dojde k samotnému výpočtu LSA, implementace tohoto procesu je popsána v sekci 5.5, a uložení vypočtených matic do souborů. Opět je nutné si po sobě uklidit, proto jsou z paměti odstraněny všechny matice, které již nyní nebudeme potřebovat (jelikož jsou pro pozdější použití uloženy v souborech). Ještě musíme nastavit kolekci dokumentů do stavu „reload“, aby se při dalším dotazu na službu relevance znovu načetla z databáze a odpovídala tak nově vypočteným maticím LSA. Na závěr asynchronní metoda uvolní *mutex*, aby v budoucnu mohlo dojít k jejímu opětovnému spuštění.

Výpočet LSA se provádí u entit kurz, student, expert a skill. Co se entity tag týče, tak zde není nutný žádný výpočet, protože veškeré informace, které potřebujeme, získáme buď přímo z dotazovaného dokumentu (hloubku výskytu jednotlivých slov, délku slova, značky *POS taggingu*), nebo z objektu, který reprezentuje kolekci dokumentů (informace pro výpočet TF-IDF). Proto pouze nastavíme kolekci dokumentů do stavu „reload“, abychom při příštím dotazu získali z databáze aktuální dokumenty.

6.5 Výjimky

Veškerá práce se soubory, databází, XML nebo JSON, maticemi, *MappedFileBuffery* vždy vyhazuje výjimku *AppException*. Neboli všechny výjimky jiného typu jsou odchytávány v *try* a *catch* blocích hned při první možnosti, a zde jsou předělány na výjimku typu *AppException*. U výjimek tohoto typu máme možnost stanovit název třídy a metody, kde výjimka nastala, typ výjimky (pokud se jednalo o odchycenou výjimku jiného typu, uvedeme právě typ odchycené výjimky) a zprávu.

Výjimky typu *AppException* „probublávají“ až do *controllerů*, kde se o jejich odchycení stará *try a catch* blok. V případě odchycení výjimky dojde k odeslání odpovědi, kde je daná výjimka převedena do JSON formátu.

6.5.1 ExceptionAdvice

Co ale s ostatními výjimkami (např. nulový ukazatel)? Nebo s výjimkami, které nastanou v asynchronních metodách 6.4.3? O odchycení takovýchto výjimek se stará *ExceptionAdvice*. *ExceptionAdvice* je anotována jako *Controlle-*

rAdvice, což jinými slovy znamená, že se nám odchytávání výjimek v této třídě bude aplikovat na všechny *controllery*. Jednotlivé metody jsou pro změnu anotovány pomocí *ExceptionHandler*, čímž Springu řekneme, že se jedná právě o metodu pro odchytávání výjimek.

Každá z těchto metod je určena pro jiný typ výjimky. Proto zde určitě budeme mít jednu pro náš typ *AppException*. Velkou výhodou je, že můžeme odchytávání nastavit i na objekty třídy *Exception* nebo *Throwable*, díky čemuž pokryjeme veškeré výjimky, které nám mohou nastat.

6.6 Logování

V aplikaci logujeme tři věci: chyby, varování a úspěšně dokončené úlohy. Veškeré logy se ukládají do databáze, kde každá skupina logů má svojí vlastní tabulku, do které ji ukládáme.

6.6.1 Chyby

Za chybu považujeme jakoukoliv výjimku, která probublala až do *controlleru*, nebo kterou jsme odchytili v *ExceptionAdvice*. Tyto výjimky znamenají, že pravděpodobně nedošlo k očekávanému výsledku např. nebyla dopočítána LSA, nepovedl se zápis do souboru, nebo do databáze apod.

Při logování chyb do databáze uložíme *timestamp*, převážně z důvodu unikátního klíče pro daný záznam v databázi, třídu a metodu, kde daná chyba vznikla, typ výjimky, který danou chybu zapříčinil, zprávu, datum a čas chyby.

6.6.2 Varování

Varování jsou takové výjimky, které nejsou klíčová pro celkový dopad výsledku požadavku. Za varování můžeme považovat např. pokud nám KOSapi vrátí chybu, díky tomu sice přijdeme o detail nějakého kurzu, nebo např. detail jiné entity, ale pro celkový výsledek to není až tak podstatný problém (pokud nejde o opakovaný výpadek, kdy dojde k vyvolání výjimky, která je vyhodnocena jako chyba). Navíc to může být stav, který se na KOSapi nemusí delší dobu změnit, proto nemá cenu kvůli externí chybě blokovat aktualizaci ostatních záznamů.

Druhou variantou, kdy dojde varování je chyba, která by zapříčinila ukončení výpočtu, ale my se s touto situací vypořádáme. Přesto chceme o takové situaci zanechat záznam, abychom věděli, kolikrát taková situace nastala. Typickým příkladem tohoto varování je singulární matice při výpočtu LSA.

Do databáze ukládáme naprosto stejné informace jako v případě chyby, protože se většinou jedná právě o výjimku, která byla odchycena v *try* a *catch* bloku. Jelikož, ale tuto výjimku řešíme např. úpravou matice, nevyhazujeme výjimku typu *AppException*, která by „probublala“ až do *controlleru*.

6.6.3 Úspěšně dokončené úlohy

Do této kategorie spadají veškeré aktualizace databáze z externích zdrojů a výpočty LSA. V tomto případě do databáze ukládáme opět *timestamp* z důvodu unikátního klíče, název úlohy, která byla úspěšně dokončena, zprávu, dobu trvání úlohy, čas a datum, kdy byla úloha dokončena.

Co se aktualizací databáze týče, ve zprávě uvedeme, kolik záznamů bylo celkově získáno, a v případě zdrojů z KOSapi, uvedeme, kolik záznamů bylo nově vloženo a kolik aktualizováno. U výpočtu matic pro LSA nás převážně zajímá doba, po kterou výpočet probíhal.

KAPITOLA 7

Diskuze

7.1 Technika hodnocení výsledků

Pro zhodnocení výsledků získaných z našich analytických metod jsme použili běžné techniky k hodnocení účinnosti algoritmů, které se využívají v odvětví dobývání znalostí (Information retrieval - IR). Jelikož je občas obtížné najít český název pro danou techniku měření, jsou názvy technik ponechány v anglickém znění.

7.1.1 Precision

$$precision = \frac{\|\{\text{relevantní dokumenty}\} \cap \{\text{načtené dokumenty}\}\|}{\|\{\text{načtené dokumenty}\}\|} \tag{7.1}$$

7.1.2 Recall

$$recall = \frac{\|\{\text{relevantní dokumenty}\} \cap \{\text{načtené dokumenty}\}\|}{\|\{\text{relevantní dokumenty}\}\|} \tag{7.2}$$

7.1.3 F-measure

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{7.3}$$

7.2 Automatická extrakce tagů

Pro vyhodnocení klíčových slov si trochu vzorce upravíme podle kontingenční tabulky 7.1. *Precision* a *recall* pak vypočítáme podle vzorce 7.4, resp. 7.5. Po-

čet tagů, kde se automatická extrakce shoduje s manuálním přiřazením, nám vyjadřuje a. Špatně přiřazené tagy nám vyjadřuje b a tagy, které automatická extrakce vynechala, nám vyjadřuje c.

$$precision = \frac{a}{a+b} \tag{7.4}$$

$$recall = \frac{a}{a+c} \tag{7.5}$$

	Klíčová slova manuálně přiřazená	Neklíčová slova manuálně přiřazená
Klíčová slova automaticky extrahovaná	a	b
Neklíčová slova automaticky extrahovaná	c	d

Tabulka 7.1: Kontingenční tabulka výsledků automatické extrakce a manuálního přiřazení klíčových slov

7.2.1 Zdrojové dokumenty

Jako zdrojové dokumenty byly vybrány články z českého serveru *www.zdrojak.cz*, především protože jsou opatřeny větším množstvím tagů a jsou zaměřeny na informační technologie. Proto, jak naše metoda extrakce klíčových slov funguje, je nutné zmínit, že ne všechna klíčová slova, kterými byly články opatřeny, se v textu článku vyskytovala. Z toho důvodu bylo měření provedeno jak se všemi tagy, kterými byly články opatřeny, tak s upravenými tagy. Pod tímto označením si můžeme představit tagy, které můžeme pomocí naší automatické extrakce opravdu získat (tj. musí se v textu, alespoň jednou vyskytovat).

7.3 Výsledky měření

Měření bylo provedeno na 100 článcích rozdílných délek a byly vybrány články z různých kategorií. Měření bylo provedeno jak s originálními tagy, tak s upravenými tagy. Všem článkům jsme postupně automaticky přiřazovali tři, čtyři, pět a šest tagů. V tabulce 7.2 můžete vidět naměřené hodnoty pro jednotlivé počty tagů jak pro originální tagy, tak pro upravené tagy. Z naměřených hodnot pro jednotlivé články jsme provedli aritmetický průměr pro získání výsledné hodnoty.

Automatická extrakce	**P**	**R**	$\mathbf{F_1}$
6 - originální tagy	0,3267	0,5638	0,3984
5 - originální tagy	0,3820	0,5471	0,4323
4 - originální tagy	0,4475	0,5164	0,4600
3 - originální tagy	0,5300	0,4670	0,4753
6 - upravené tagy	0,3267	0,7029	0,4294
5 - upravené tagy	0,3820	0,6812	0,4699
4 - upravené tagy	0,4475	0,6466	0,5066
3 - upravené tagy	0,5300	0,5825	0,5307

Tabulka 7.2: Naměřené výsledky

7.4 Zhodnocení výsledků

Z naměřených výsledků 7.2 jednoznačně vidíme, že čím méně tagů automaticky přiřazujeme, tím lepších výsledků dosáhneme. To je způsobeno dvěma aspekty. Průměrný počet manuálně přiřazených tagů na článek je roven 3,7. Díky tomu je jasné, že čím více tagů bude automatická extrakce přiřazovat, tím více bude klesat hodnota *precision*. Můžeme z toho odvodit i další věc, že pokud byl tag pomocí automatické extrakce správně přiřazen, objevoval se vždy velmi vysoko (tj. do třetího místa podle relevance).

Pokud se nejdříve budeme bavit o originálně přiřazených klíčových slovech, tak pouze u pěti článků z celkového počtu se automatické extrakci nepodařilo přiřadit žádný správný tag, bavíme-li se o počtu šesti automaticky extrahovaných štítků. Tento počet stoupl na šest článků, pokud jsme přiřazovali vždy pouze tři tagy. Což jenom potvrzuje výše uvedené, že v případě nalezení správného tagu, byl tento tag vyhodnocen s vysokou relevancí. Úplně stejné výsledky jsme naměřili v případě upravených štítků (tj. pět článků pro šest automaticky extrahovaných klíčových slov a šest článků pro tři tagy).

Zajímá nás také, u kolika článků se nám podařilo přiřadit všechna klíčová slova, tj. hodnota *recall* se rovná jedné. Pro originální tagy a šest automaticky přiřazených tagů se nám tak podařilo u 15 článcích. Při snížení počtu

automaticky extrahovaných tagů na tři se tak stalo u 9 článků. Nejlepších výsledků jsme dosáhli u upravených tagů a šesti přiřazených klíčových slovech, kdy hned 40 článků obdrželo všechny svoje klíčová slova. Při snížení počtu na tři tagy, jsme pak všechny tagy přiřadili u 24 článků.

7.5 Srovnání

Abychom zde stále neuváděli pouze nějaká čísla, uvedeme si porovnání různých modelů automatické extrakce, které uvádí [5]. Měření bylo provedeno na 600 akademických pracích v oblasti ekonomie s průměrným počtem klíčových slov rovným 7,83 na dokument. Srovnání můžete vidět v tabule C.3. Kde modely *BaseLine1* a *BaseLine2* jsou základní heuristické modely, ze kterých studie vycházela. *MLR* je metoda založená na lineární regresi. *Logit* je model využívající logistické regrese. *SVM* (*Support Vector Machines*) je metoda založená na strojovém učení a *CRF* (*Conditional Random Fields*) je pravděpodobnostní model pro segmentaci a značkování sekvencí dat.

Model	**P**	**R**	$\mathbf{F}_1$
BaseLine1	0,2343	0,4508	0,3083
BaseLine2	0,2778	0,5287	0,3656
MLR	0,3174	0,5233	0,3951
Logit	0,3248	0,5388	0,4067
SVM	0,8017	0,3327	0,4653
CRF	0,6637	0,4196	0,5125

Tabulka 7.3: Porovnání modelů automatické extrakce klíčových slov

Samozřejmě nemůžeme porovnávat výsledky našeho měření s výsledky měření jiných modelů na jiných datech. Nicméně pro ilustraci můžeme vidět, jakých výsledků dosahují jiné modely.

Závěr

Cílem práce bylo vytvořit analytické webové služby, které pomohou usnadnit práci uživatelům portálu Spolupráce s průmyslem. Analytické metody jsou koncipovány tak, aby byly nezávislé na vstupních dokumentech. Jinými slovy je možné aplikaci rozšířit např. na další externí zdroje pro jednotlivé entity. To jak moc tato práce pomůže uživatelům portálu Spolupráce s průmyslem, ukáže až její samotné používání. Výsledkem není pak jenom aplikace, ale také nabyté zkušenosti z vývoje této aplikace a získané vědomosti v dané oblasti.

Literatura

[1] OAuth 2.0. Dostupné z WWW: https://rozvoj.fit.cvut.cz/Main/oauth2

[2] Portál spolupráce s průmyslem. Dostupné z WWW: https://wiki.cvut.cz/confluence/pages/viewpage.action?pageId=14057588

[3] *Handbook of Latent Semantic Analysis*. Psychology Pr, 2014, ISBN 1138004197.

[4] Berry Michael W, K. J.: *Text mining: applications and theory*. John Wiley and Sons, 2010.

[5] Chengzhi ZHANG, Y. L. D. W. Y. L. B. W., Huilin WANG: Automatic Keyword Extraction from Documents Using Conditional Random Fields. 2008. Dostupné z WWW: http://eprints.rclis.org/12305/1/Automatic_Keyword_Extraction_from_Documents_Using_Conditional_Random_Fields.pdf

[6] Hajic, J.: Positional Tags: Quick Reference (Czech Morphology). 2000. Dostupné z WWW: https://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/Doc/hmptagqr.html

[7] M, W. S.: *Text mining: predictive methods for analyzing unstructured information*. Springer, 2005.

[8] postgresql.org: SQL Conformance. Dostupné z WWW: http://www.postgresql.org/docs/current/static/features.html

[9] Zdeněk Dostál, V. V.: *Lineární algebra*. Technická univerzita Ostrava, Západočeská univerzita v Plzni, 2012.

PŘÍLOHA A

Seznam použitých zkratek

ACID Atomicity, Consistency, Isolation, Durability

ČVUT České vysoké učení technické

FIT Fakulta informačních technologií

HTTP Hypertext Transfer Protocol

IDF Inverse Document Frequency

IR Information Retrieval

JAMA A Java Matrix Package

Java EE Java Enterprise Edition

Java EL Java Expression Language

JSON JavaScript Object Notation

JSP Java Server Pages

LSA Latentní sémantická analýza, Latent Semantic Analysis

LSI Latentní sémantické indexování, Latent Semantic Indexing

MVCC Multiversion Concurrency Control

POS Project Object Model

POS Part-of-speech

RSQL RESTful Service Query Language

SQL Structured Query Language

SSP portál Spolupráce s průmyslem

SPARQL SPARQL Protocol and RDF Query Language

SVD Singular Value Decomposition

SVM Support Vector Machine

TF Term Frequency

TF-IDF Term Frequency - Inverse Document Frequency

URL Uniform Resource Locator

XML Extensible Markup Language

PŘÍLOHA B

Instalace a spuštení

B.1 Nastavení konfiguračního souboru

Před samotnou instalací a spuštěním aplikace je nutné provést její konfiguraci. Ke konfiguraci slouží konfigurační soubor (*sspwebservices/project/src/-main/resources/application.properties*). Velká část konfigurace je přednastavena, nicméně musíme nastavit připojení k databázi B.1, kde konfigurační proměnné *postgresql.user* nastavíme hodnotu uživatelského jména uživatele, který bude přistupovat do databáze, a do proměnné *postgresql.password* jeho heslo. Do proměnné *postgresql.url* ukládáme adresu připojení k databázi ve formátu *jdbc:subprotocol:subname*.

```
postgresql.password=
postgresql.url=
postgresql.user=
```

Listing B.1: Nastavení databáze v souboru application.properties

Dále je nutné nastavit absolutní cestu do pracovního adresáře aplikace B.2 do konfigurační proměnné *filesystem.basepath*.

```
filesystem.basepath=
```

Listing B.2: Nastavení pracovního adresáře v souboru application.properties

Jako poslední je nutné nastavit hodnoty *Client ID* a *Client secret* pro přídstup do KOSapi pomocí OAuth B.3.

```
oauth.clientid=
oauth.clientsecret=
```

Listing B.3: Nastavení OAuth v souboru application.properties

Ostatní konfigurační proměnné jsou přednastavené a jejich vyvsětlivky naleznete v příloze C.

B.2 Databáze

Na stroji, kde aplikace poběží, je nutné mít běžící PostgreSQL databázi. Pro vytvoření struktury databáze slouží SQL skript (*database/database.sql*). Po spuštění tohoto skriptu v dotazovacím nástroji databáze jsou vytvořeny potřebné tabulky pro běh aplikace.

B.3 Build aplikace

O vybuildování aplikace se stará Apache Maven. Ve stromové struktuře adresářů se přesuneme do adresáře *sspwebservices/project* a zde spustíme příkaz:

```
mvn package
```

Nebo následující příkaz pro operační systém Windows (zde je nutné mít v proměnné prostředí PATH uloženou cestu k binárním souborům Apache Maven):

```
mvn.cmd package
```

Po tomto kroku se nám vytvoří adresář *sspwebservices/project/target*, který obsahuje zabalený JAR soubor s aplikací a serverem Apache Tomcat.

B.4 Spuštení aplikace

Pro spuštění aplikace je nutné spustit JAR soubor, který v sobě má zabalený server Apache Tomcat spolu s WAR souborem naší aplikace. Stačí vejít do adresáře *sspwebservices/project/target* a zde spustit následující příkaz:

```
java -jar sspwebservices-1.0.jar
```

Tím to příkazem se spustí naše aplikace běžící na adrese *http://localhost:8080*. Port i adresu můžeme změnit v konfiguračním souboru *sspwebservices/project/src/main/resources/application.properties*, nicméně je nutné znovu provést build aplikace popsán v sekci B.3. Také musíme mít na paměti, že po spuštění aplikace je port, pod kterým aplikace běží, zabrán a k jeho uvolnění dojde až při ukončení procesu.

Pro správný běh aplikace je nutné nakopírovat obsah adresáře *src* do pracovního adresáře aplikace, jehož nastavení je popsáno v B.1.

PŘÍLOHA C

Konfigurační soubor

server.port	Port, na kterém aplikace poběží.
server.address	Adresa, na které aplikace běží (zakomentováno).
postgresql.user	Uživatelské jméno pro přístup do databáze.
postgresql.password	Heslo pro přístup do databáze.
postgresql.url	Adresa připojení k databázi ve formátu *jdbc:subprotocol:subname*.
filesystem.basepath	Absolutní cesta k pracovnímu adresáři.
filesystem.coursematrixbase	Název matic, které pracují s kurzy.
filesystem.expertsmatrixbase	Název matic, které pracují s experty.
filesystem.studentsmatrixbase	Název matic, které pracují s studenty.
filesystem.skillsmatrixbase	Název matic, které pracují s skilly.
filesystem.lsamatrixbase	Název matic, které pracují s LSA.
filesystem.termextractionmatrixbase	Název matic, které pracují s automatickou extrakcí termů.
application.lsaweightingmethodclass	Třida použitá pro vážení termů při výpočtu LSA.

Tabulka C.1: Konfigurační soubor

application.termextractionweightingmethodclass	Třida použitá pro vážení termů při automatické extrakci tagů.
application.lsaminreduction	Minimální dimenzionální redukce LSA (parametr k).
application.lsamaxreduction	Maximální dimenzionální redukce LSA (parametr k).
application.tagconsideration	Konstanta, kterou násobíme relevanci tagů, které se objevují v portálu pro Spolupráci s průmyslem.
oauth.clientid	*Client ID* aplikace použito pro OAuth přístup do KOSapi.
oauth.clientsecret	*Client secret* aplikace použito pro OAuth přístup do KOSapi.
oauth.url	URL autorizačního serveru pro OAuth přístup do KOSapi.
oauth.parameters	Tělo požadavku odeslákého na adresu autorizačního serveru.
kosapi.url	URL KOSapi.
kosapi.coursesurl	Relativní adresa pro výpis seznamu kurzů.
kosapi.coursesparams	Parametry pro výpis seznamu kurzů.
kosapi.coursedetailparams	Parametry pro detailní výpis kurzu.
kosapi.coursesnodes	Položky detailu kurzu použité pro setsavení dokumentu.

Tabulka C.2: Konfigurační soubor

kosapi.studentsurl	Relativní adresa pro výpis seznamu studentů.
kosapi.studentsparams	Parametry pro výpis seznamu studentů.
kosapi.studentdetailurl	Relativní adresa detailu studenta.
kosapi.studentdetailparams	Parametry pro detailní výpis studenta.
kosapi.expertsurl	Relativní adresa pro výpis seznamu expertů.
kosapi.expertsparams	Parametry pro výpis seznamu expertů.
kosapi.expertdetailurl	Relativní adresa detailu experta.
kosapi.expertdetailparams	Parametry pro detailní výpis experta.
ssp.skillsurl	URL adresa pro získání seznamu skillů z portálu Spolupráce s průmyslem.
ssp.tagsurl	URL adresa pro získání seznamu tagů z portálu Spolupráce s průmyslem.
dbpedia.url	URL SPARQL endpointu české DBpedie.

Tabulka C.3: Konfigurační soubor

PŘÍLOHA D

SQL skript pro vytvoření objektů v databázi

```
CREATE TABLE courses (
  code varchar(10),
  kosapi_id text,
  updated date,
  title text,
  description text,
  CONSTRAINT courses_pk PRIMARY KEY (code)
)
WITH (
  OIDS=FALSE
);

CREATE TABLE courses_terms (
  term varchar(50),
  code varchar(10),
  tf integer,
  CONSTRAINT courses_terms_pk PRIMARY KEY (term, code),
  CONSTRAINT courses_terms_code_fk FOREIGN KEY (code)
      REFERENCES courses (code) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION

)
WITH (
  OIDS=FALSE
);
```

```
CREATE TABLE students (
    username varchar(8),
    email text,
    updated date,
    first_name text,
    last_name text,
    CONSTRAINT students_pk PRIMARY KEY (username)
)
WITH (
    OIDS=FALSE
);

CREATE TABLE students_courses (
    username varchar(8),
    code varchar(10),
    CONSTRAINT students_courses_pk PRIMARY KEY (username, code),
    CONSTRAINT students_courses_code_fk FOREIGN KEY (code)
      REFERENCES courses (code) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT students_courses_username_fk FOREIGN KEY (username)
      REFERENCES students (username) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
    OIDS=FALSE
);


CREATE TABLE matrices_dimensions (
    matrix varchar(100),
    m integer,
    n integer,
    CONSTRAINT matrices_dimensions_pk PRIMARY KEY (matrix)
)
WITH (
    OIDS=FALSE
);

CREATE TABLE experts (
    username varchar(8),
    email text,
    updated date,
    first_name text,
    last_name text,
```

```
    CONSTRAINT experts_pk PRIMARY KEY (username)
)
WITH (
    OIDS=FALSE
);

CREATE TABLE experts_courses (
    username varchar(8),
    code varchar(10),
    CONSTRAINT experts_courses_pk PRIMARY KEY (username, code),
    CONSTRAINT experts_courses_code_fk FOREIGN KEY (code)
      REFERENCES courses (code) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT experts_courses_username_fk FOREIGN KEY (username)
      REFERENCES experts (username) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
    OIDS=FALSE
);

CREATE TABLE skills_dbpedia (
  name text,
  description text,
  dbpedia_id integer,
  skill_id integer,
  round integer,
  CONSTRAINT skills_dbpedia_pk PRIMARY KEY (dbpedia_id)
)
WITH (
  OIDS=FALSE
);

CREATE TABLE skills_dbpedia_terms (
  term varchar(50),
  dbpedia_id integer,
  tf integer,
  CONSTRAINT skills_dbpedia_terms_pk PRIMARY KEY (term, dbpedia_id),
  CONSTRAINT skills_dbpedia_terms_dbpedia_id_fk FOREIGN KEY (dbpedia_id)
      REFERENCES skills_dbpedia (dbpedia_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
```

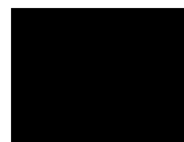
```
);


CREATE TABLE logs_warnings (
    timestamp bigint,
    class text,
    method text,
    exception text,
    message text,
    date date,
    time time,
    CONSTRAINT logs_warnings_pk PRIMARY KEY (timestamp, class, method,
)
WITH (
  OIDS=FALSE
);

CREATE TABLE logs_tasks (
    timestamp bigint,
    task text,
    message text,
    duration double precision,
    date date,
    time time,
    CONSTRAINT logs_tasks_pk PRIMARY KEY (timestamp, task)
)
WITH (
  OIDS=FALSE
);

CREATE TABLE logs_errors (
    timestamp bigint,
    class text,
    method text,
    exception text,
    message text,
    date date,
    time time,
    CONSTRAINT logs_errors_pk PRIMARY KEY (timestamp, class, method, e
)
WITH (
  OIDS=FALSE
);
```

PŘÍLOHA E



Obsah přiloženého CD

- database .... adresář obsahující skript pro vytvoření databáze
- measurement ... adresář obsahující jednotlivé články využité pro měření
- src .... adresář obsahující soubory nutné pro běh aplikace
- sspwebservices .... adresář projektu aplikace
- thesis .... adresář obsahující práci ve formátu LaTeX
- text .... text práce
  - DP_Marousek_Jiri_2015.pdf .... text práce ve formátu PDF