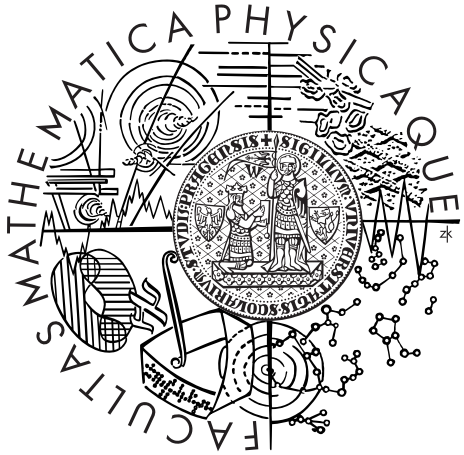
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Ondřej Odcházel

Automatické doporučování ilustračních snímků

Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: RNDr. Pavel Pecina, Ph.D.
Studijní program: Informatika
Studijní obor: Matematická lingvistika

Praha 2014

Děkuji svému vedoucímu, RNDr. Pavlovi Pecinovi, Ph.D. za pomoc a cenné připomínky v průběhu celého vývoje projektu. Také bych rád poděkoval své rodině a přítelkyni, hlavně za velkou trpělivost.

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Automatické doporučování ilustračních snímků

Autor: Bc. Ondřej Odcházel

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: RNDr. Pavel Pecina, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt:

Klíčová slova: vyhledávání obrazových informací

Title: Automatic suggestion of illustrative images

Author: Bc. Ondřej Odcházel

Department: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Pavel Pecina, Ph.D., Institute of Formal and Applied Linguistics

Abstract:

Keywords: information retrieval, image retrieval

Obsah

1. Úvod

Cílem diplomové práce je implementovat kompletní webovou aplikaci pro doporučování a vyhledávání ilustračních obrázků v textu. Vytořit takovou aplikaci přináší mnoho rozličných úkolů a problémů. Tato kapitola se bude snažit tyto problémy načrtnout. Další kapitola se bude jednotlivými problémy zabývat podrobně.

1.1 Práce s daty

Zadaná data obsahují 20 milionů anotací obrázků. Základním úkolem je být schopen takové množství dat vůbec nahrát do databáze a být schopný obsloužit mnoho požadavků za minutu. Bude zmíněn současný stav vývoje databázového software pro práci s velkými daty zejména s ohledem na snadnost hledání a škálovatelnost.

1.2 Extrakce klíčových slov

Extrakce klíčových slov je důležitý podobor NLP. V práci budou rozebrány algoritmy pro extrakci klíčových slov. Bude kladen zejména důraz na rychlost a nenáročnost na zdroje. Z uživatelských testovaní společnosti Google vychází, že rychlost načtení stránky je jedním z klíčových vlastností pro spokojenost uživatele. Klíčová slova budou mít v aplikaci dvě využití. Pokud uživatel zadá pouze text článku, extrahovaná klíčová slova se použijí na vyhledávání relevantních obrázků. Prvních několik klíčových slov bude navíc použita jako nápověda uživateli, ten pak může tato klíčová slova využít k exaktnímu omezení množiny klíčových slov.

1.3 Překlad do češtiny

Popisky klíčových slov jsou v angličtině. Tato práce řeší překlad množiny klíčových slov do češtiny. Kromě překladu je pro hledání také nutno implementovat algoritmus na stemming. Celá aplikace je navržena tak, aby případný další jazyk mohl být přidán co nejjednodušeji.

1.4 Detekce jazyka

Jednou z drobností, kterou ocení uživatel aplikace je detekce jazyka. Uživatel bude mít možnost zadat jazyk vstupního článku exaktně, ale aplikace bude také jazyk vstupního textu sama detekovat. Budou prozkoumány možnosti detekce jazyka. Opět se nejedná o nějakou klíčovou funkci aplikace. Výstup detekce bude moci být uživatelem měněn (podobně jako funguje Google Translate[1]), důraz bude tedy kladen na rychlost a jednoduchost.

[1] https://translate.google.com/

1.5 Webová aplikace

Všechny předchozí komponenty se spojí v jedné webové aplikaci. Webový vývoj zažívá bouřlivý rozvoj. Na backendu jsou nové zejména způsoby práce s velkým množstvím dat v distribuovaném prostředí. Ve frontendové části probíhá rozvoj pomocí implementace nových technologií, známých pod hlavičkou HTML5, do moderních prohlížečů. Práce bude rozebírat všechny možnosti tvorby moderních webových aplikací.

1.6 Testování

Aplikace bude otestována na několika úrovních. Extrakce klíčových slov bude otestována pomocí korpusu článků a klíčových slov. Bude vytvořena komplexní webová aplikace pro testování doporučených obrázků. Tato aplikace bude vydělena ze samotné webové aplikace a bude používána i nezávisle.

2. Poskytnutá data

Tato kapitola popisuje data, která slouží jako vstup projektu.

2.1 Profimedia dataset

Společnost Profimedia poskytla pro výzkumné účely korpus více než dvaceti milionů ilustračních obrázků spolu s jejich textovým popisem.

Textové popisky byly očištěny[7] a poskytnuty pro tento projekt ve formě souboru `profi-text-cleaned.csv`. Soubor je ve formátu CSV a obsahuje 20 014 394 řádků. Každý řádek obsahuje 4 složky:

locator
: Identifikátor obrázku v databázi Profimedia. Desetimístný řetězec číslic.

title
: Název obrázku v anglickém jazyce.

description
: Pro všechny řádky souboru prázdná položka.

keywords
: Mezerami oddělená klíčová slova obrázku.

Ukázka 2.1 Příklad obsahuje příklad jednoho řádku souboru `profi-text-cleaned.csv`.

```
"0000000980","hradec kings holy ghost cathedral","",
  "outdoors nobody urban scenes architecture houses
  towers czech czech republic europe buildings build
  history historical churches church fronts holy
  ghost cathedral spirit ceska republika cathedrals
  sv hradec kralove"^M
```

Ukázka 2.1: Řádek souboru profi-text-cleaned.csv

Na příkladu jsou vidět některé problémy, které data z datasetu Profimedie mají. Některá klíčová slova, jako například „czech republic“, k sobě patří a tvoří frázi. V souboru ovšem tyto víceslovné fráze nejsou vyznačené. Některým slovům chybí diakritika, například „ceska“. Některé fráze vznikly nejspíše strojovým překladem z cizího jazyka do angličtiny. To je vidět na frázi „hradec kings“, která zřejmě původně byla názvem českého města „Hradec Králové“. Všechna slova v souboru obsahují pouze malá písmena, což například znesnadňuje detekci názvů.

Všechny popsané nedostatky dat negativně ovlivňují možnosti pro kvalitní vyhledávání v poskytnutých datech. Jedním z cílů práce je co nejvíce těchto nedostatků opravit.

2.2 Profimedia dataset s detekovanými frázemi

Některé problémy s datasetem Profimedie byly odstraněny v rámci bakalářské práce Bc. Jana Botorka[5]. Jedním z výsledků této práce je soubor `keyword-clean-phrase-expo` Ukázka 2.2 obsahuje příklad jednoho řádku tohoto souboru.

```
"0000000980";"Hradec kings holy ghost
   cathedral";"outdoors,nobody,urban
   scenes,architecture,houses,towers,czech,czech
   republic,Česká republika,Europe,
   buildings,build,history,historical,
   churches,church,fronts,cathedrals sv,holy ghost
   cathedral,spirit,Hradec Králové"
```

Ukázka 2.2: Řádek souboru keyword-clean-phrase-export.csv

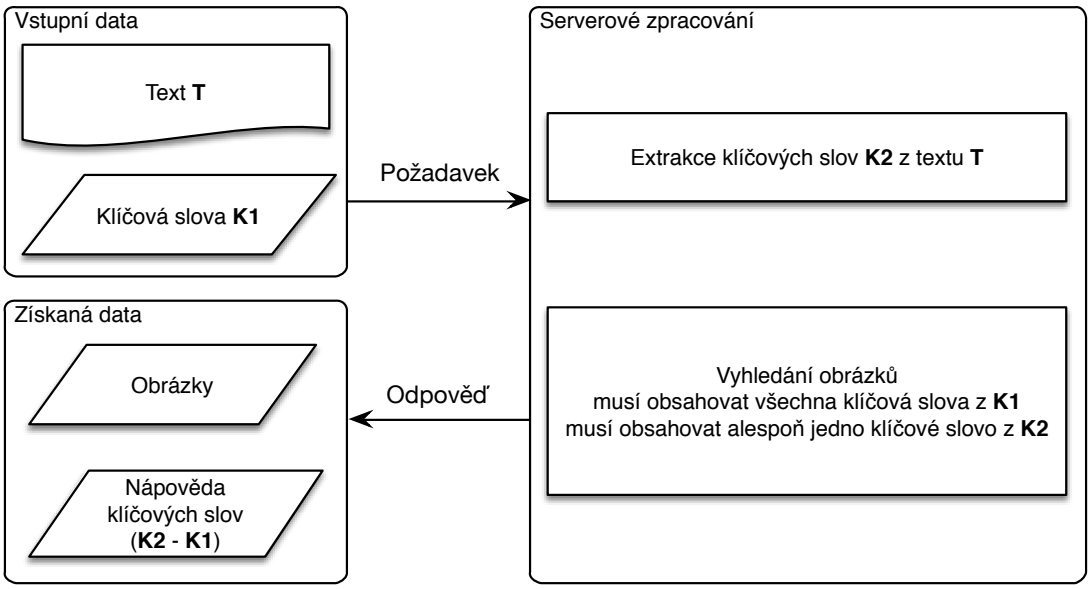
Soubor obsahuje lépe zpracovaná data z datasetu Profimedia. Nejdůležitější změnou pro tuto práci je detekce klíčových frází. Jednotlivé fráze jsou od sebe odděleny čárkou. Detekce probíhala pomocí databáze WordNet[25] a Wikipedie. Ne vždy se detekce povedla, takže v některých řádcích nejsou žádné fráze detekovány. Detekce frází je důležitá zejména pro překlad dat do jiných jazyků.

2.3 Vektor podobnosti obrázků

Kromě textových popisků jsou součástí datasetu Profimedie i samotné obrázky. Jedním z cílů práce je umožnit nad databází obrázků vyhledávání na základě vizuální podobnosti. K tomuto účelu byl poskytnut soubor `profi-neuralnet-20M.data.gz`, který je komprimovaný formátem ZIP a má velikost 129 GB.

Soubor obsahuje pro každý obrázek z datasetu Profimedie vektor 4096 reálných čísel. Pomocí vektorové vzdálenosti lze určit, jak jsou si podobné dva obrázky mezi sebou.

3. Vyhledávání relevantních obrázků



Obrázek 3.1: Základní tok dat mezi klientem a serverem.

Aplikace poskytuje uživateli dvě možnosti vyhledávání. Základním uživatelským scénářem je vložit do rozhraní text. Aplikace by v takovém případě měla poskytnout relevantní obrázky k danému textu. V dalším uživatelském scénáři je přímý požadavek na klíčová slova obrázku. Uživatel by měl mít možnost zadat přímo klíčová slova, které metadata k nalezeným obrázkům musí obsahovat. Oba scénáře by mělo navíc být možné propojit pomocí nápovědy klíčových slov – uživatel zadá do rozhraní text, dostane výsledné obrázky a nápovědu klíčových slov, kterými může množinu nalezených obrázků více omezit. Celý proces hledání vhodných obrázků popisuje diagram 3.1.

3.1 Extrakce klíčových slov

Extrakce klíčových slov je velmi důležitou složkou celého vyhledávání. Článek [23] shrnuje základní techniky extrakce klíčových slov z textu. Algoritmy na extrakci klíčových slov lze v zásadě rozdělit do dvou kategorií - „s korpusem“ a „bez korpusu“. Metody pracující bez korpusu jsou zajímavé a mohou dosahovat podobných výsledků jako metody s korpusem. My však máme k dispozici dataset Profimedie, takže o metody pracující bez korpusu se tato práce dále nezajímá.

3.2 TF-IDF

TF-IDF je jeden ze základních vyhledávacích algoritmů. Algoritmus využívá korpusu dokumentů D a dvou složek TF a IDF, lze ho vyjádřit jako rovnost

$$TFIDF(t, d, n, N) = TF(t, d) \times IDF(n, N). \tag{3.1}$$

Složka TF znamená $TERM\ FREQUENCY$ a pokud t je slovo a $d \in D$ je dokument, je TF

$$TF(t, d) = \sum_{slovo \in d} \begin{array}{ll} 1 & \text{pokud } slovo = t \\ 0 & \text{jinak} \end{array}. \tag{3.2}$$

Jedná se tedy o frekvenci slova v dokumentu.

Složka IDF, tedy $INVERSE\ DOCUMENT\ FREQUENCY$ vyjadřuje, jak moc daný termín popisuje dokument. Pokud je N počet všech dokumentů v D, tedy $N = |D|$ a n je počet dokumentů, ve kterých se vyskytuje slovo t, je IDF tohoto slova

$$IDF(n, N) = \log\left(\frac{N}{n}\right). \tag{3.3}$$

Čím je tedy slovo v korpusu častější, tím více se s logaritmem snižuje jeho informační hodnota. Slova, která jsou velmi běžná, většinou klíčovými slovy nejsou.

Výsledný vzorec pak jde shrnout jako

$$TFIDF(t, d, n, N) = \left(\sum_{slovo \in d} \begin{array}{ll} 1 & \text{pokud } slovo = t \\ 0 & \text{jinak} \end{array}\right) \times \log\left(\frac{N - n}{n}\right). \tag{3.4}$$

3.3 TF-IDF pro extrakci klíčových slov

Algortimus TF-IDF můžeme použít pro extrakci klíčových slov z textu. Všechna znaky vstupního textu převedeme na malá písmena a text rozdělíme na slova. Odstraníme slova ze stop-slovníku, který obsahuje nejběžnější slova pro každý z podporovaných jazyků[1]. Dále nás nezajímá diakritika a různé speciální znaky, které může text obsahovat. Slova převedeme na stemy. O převodu slov na stemy se podrobněji píše v Kapitole 5.

Algoritmus TF-IDF pak použijeme na každé slovo vstupního textu a získáme tak skóre jeho významnosti v textu. Slova s nejvyšším skóre pak označíme za klíčová. Algoritmus musíme upravit pro naše účely. Zaprvé nás zajímají pouze taková slova, která existují v datasetu Profimedie. Slova, která se v korpusu nenachází, dostanou skóre 0. Část TF v algoritmu znamená četnost slova v textu.

Složitější je situace s IDF. Nabízí se použít frekvenci slov z datasetu Profimedie. Ukázalo se, že klíčová slova obrázků z datasetu nejsou pro tento účel vhodná. Klíčová slova totiž obsahují mnoho názvů a obecně méně běžných slov. Naopak obsahují velmi málo běžných slov. To pak způsobuje, že algoritmus na tomto datasetu přiřazuje vysoké skóre běžným slovům. Tato slova ale typicky nejsou vhodnými klíčovými slovy daného textu. Je tedy potřeba použít pro vzorec DF jiný korpus. V naší aplikaci jsme použili korpus Wikipedie, jejíž data jsou volně dostupná pro mnoho jazyků pod otevřenou licencí.

[1]Seznam slov ve stopslovnících je převzat z knihovny Lucene.

Pokud je w slovo vstupního textu, $Freq(w)$ je četnost slova ve vstupním textu a $Wiki(w)$ je četnost slova v korpusu Wikipedie, můžeme každému slovu vstupního slova přířadit $Score$:

$$Score(w) = \begin{cases} Freq(w) \times (\frac{C}{Wiki(w)}) & w \text{ je v datasetu Profimedie} \\ 0 & w \text{ není v datasetu Profimedie} \end{cases} \tag{3.5}$$

C je experimentálně zjištěná konstanta. V našem případě je její hodnota 10 000 000.

Nyní tedy máme skóre udávající význam slova pro každé slovo vstupního textu. Slova s nejvyšším skóre vrátíme uživateli jako nápovědu pro explicitní požadavek obrázků s klíčovými slovy.

3.4 Vyhledávání obrázků

Samotné vyhledávání obrázků má dva druhy vstupních dat. Prvním typem je text článku, který uživatel vloží do uživatelského rozhraní. Máme tedy k dispozici řetězec s textem článku. Dále může uživatel zadat explicitní klíčová slova, které má hledaný obrázek obsahovat. Tato vstupní data získáváme jako pole řetězců. Úkolem algoritmu na vyhledávání obrázků je vrátit uživateli všechny relevantní obrázky v pořadí podle relevance.

Text si nejprve zpracujeme pomocí algoritmu uvedeném v sekci 3.3. Získáme skóre významnosti pro všechna slova ve vstupním textu. Pro vyhledávání použijeme pouze několik slov s nejvyšším skóre.

Nyní můžeme jako množinu relevantních obrázků označit obrázky, které ve svých klíčových slovech mají všechna uživatelem zadaná explicitní klíčová slova a alespoň jedno klíčové slovo získané extrakcí klíčových slov z textu.

Množina relevantních obrázků může být velká a obsahovat obrázky, které jsou relevantní jen velmi málo. Je proto důležité množinu relevantních obrázků správně seřadit. Máme množinu K_{text} klíčových slov získaných z textu. Každé $w \in K_{text}$ má skóre $Score(w)$. Relevantní obrázek má množinu klíčových slov K_{img}. Relevanci obrázku vůči uživatelskému dotazu můžeme ohodnotit funkcí $Rank$:

$$Rank(K_{text}, K_{img}) = \sum_{w \in K_{text}} \begin{cases} Score(w) \times \frac{1}{|K_{img}|} & w \in K_{img} \\ 0 & w \notin K_{img} \end{cases} \tag{3.6}$$

Vzorec bere v úvahu počet klíčových slov, které obrázek obsahuje. Snižuje relevanci obrázků, které obsahují mnoho klíčových slov a zvýhodňuje tím ve vyhledávání obrázky, které mají menší počet přesnějších klíčových slov.

4. Překlad obrázkových popisků

Popisky obrázků v datasetu Profimedie jsou v angličtině. Jedním z cílů aplikace je poskytnout kromě angličtiny vyhledávání i v jiném jazyce. Zaměřujeme se na češtinu, ale podobné úvahy a postupy platí většinou i pro jiné jazyky. Jsou v podstatě dvě možnosti, jak implementovat vyhledávání v češtině. Můžeme buď překládat do angličtiny hledané texty a klíčová slova, nebo předpřeložit dataset Profimedie.

My jsme se rozhodli pro druhou možnost. Musíme tedy přeložit všechny texty v datasetu Profimedie. Vzhledem k tomu, že obrázků je více než 20 milionů, nepřipadá lidský překlad v úvahu z časových i finančních důvodů. Jedinou reálnou možností je použít překlad strojový.

4.1 Strojový překlad

Strojový překlad jako obor zažívá velký rozvoj. Potřeba překladu stále celosvětově prudce stoupá. To jednak zvyšuje poptávku po kvalitním strojovém překladu a dále to strojovému překladu dává k dispozici velké množství lidských překladů, které jsou pro kvalitní strojový překlad nezbytné. Lze brát v úvahu i pravidlové překladové systémy, které ke své práci tolik dat většinou nepotřebují. Obecnější a pro většinu jazykových párů nejlepší řešení však nabízí frázový statistický strojový překlad.

4.2 Frázový statistický strojový překlad

Frázový statistický strojový překlad[20] ke své práci potřebuje databázi lidsky přeložených frází. Fráze jsou několikaslovné kusy přeloženého textu. Typicky se získávají z paralelního korpusu textů ve zdrojovém a cílovém jazyce. Soubor s frázovými daty obsahuje položku s textem fráze ve zdrojovém a cílovém jazyce spolu s pravděpodobností, že je takový překlad fráze správný.

Druhým důležitým vstupem strojového frázového systému je korpus cílového jazyka, ze kterého se vytvoří jazykový model. Slouží zejména k tomu, aby k sobě poskládané fráze v cílovém jazyce dobře „seděly“. Pokud máme překladový i jazykový model, můžeme vyjádřit pravděpodobnost překladu pomocí základního vzorce statistického strojového překladu. Překládáme řetězec f ve zdrojovém jazyce. K dispozici máme překladový model $p(f|e)$, který udává pravděpodobnost toho, že řetězec f ve zdrojovém jazyce je překladem řetězce e v cílovém jazyce. Jazykový model $p(e)$ nám vrací pravděpodobnost řetězce e v cílovém jazyce. Chceme získat takový řetězec $\tilde{e}$, pro který je pravděpodobnost $p(e|f)$ nevyšší. Pomocí Bayesova pravidla můžeme k hledání takové fráze použít překladový a jazykový model:

$$\tilde{e} = arg \max_{e \in e^*} p(e|f) = arg \max_{e \in e^*} p(f|e)p(e) \tag{4.1}$$

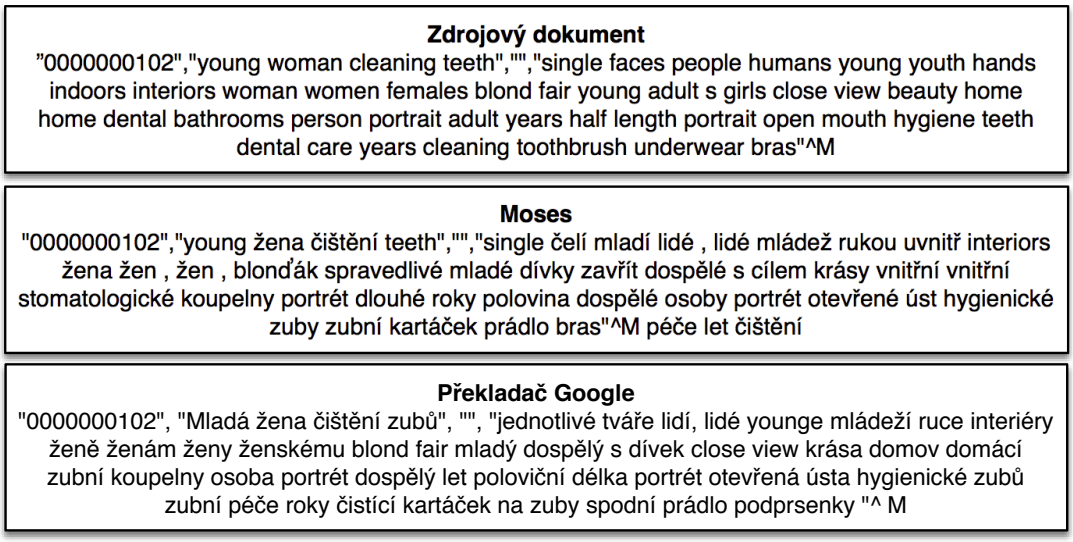
V praxi je potřeba u kvalitního strojového překladu vyřešit mnoho dalších problémů. K lepším výsledkům překladu potřebujeme reordering model, který umožňuje přesouvat pozice frází mezi zdrojovým a cílovým textem.

Dobrý frázový překlad potřebuje ke svému chodu miliony frází. Vyhledávání nejpravděpodobnějšího překladu v takovém množství dat je náročná úloha. Algoritmy, které umožňují vyhledávat ve velkém množství překladových dat jsou implementována v knihovně Moses[21], která je dostupná pod volnou licencí včetně základních překladových modelů pro některé jazykové páry.

4.3 Charakteristika dat pro překlad

Pro správné fungování vyhledávání v českém jazyce je potřeba přeložit klíčová slova v Profimedia datech do češtiny. Slova v korpusu jsou oddělena mezerou. Je ale zřejmé, že některá slova vedle sebe k sobě patří — jsou to fráze — zatímco některá nikoliv. Naskytují se tedy zhruba tři možnosti, jak takový text přeložit.

4.4 Překlad vět



Obrázek 4.1: Ukázka překladu metadat k obrázku pomocí Mosese a Překladače Google.

První možností je přistupovat k souboru klíčových slov u každého obrázku jako k větě a použít frázový strojový překlad — buď Moses, nebo Překladač Google[17] — k překladu z angličtiny do češtiny. Tento přístup má několik problémů. Frázový překlad se snaží aplikovat fráze z překladového modelu. V našem souboru klíčových slov ale mohou být vedle sebe slova, která tvoří frázi pouze zdánlivě. Například můžeme mít fotku dítěte, které stojí před automobilem značky Seat se dvěma klíčovými slovy vedle sebe — „child“ a „seat“. Frázový překlad z angličtiny do češtiny pochopí tato dvě slova jako fráze, které do češtiny přeloží frází „dětské sedadlo“, která ovšem neodpovídá popisku obrázku.

Dalším problémem tohoto přístupu je časová a finanční náročnost takového řešení. Frázový překlad je dosti náročný algoritmus a překlad dvaceti milionů vět může být dosti obtížný. Překlad pomocí Mosese nás omezuje výpočetní složitostí.

Na jednom stroji by takový překlad trval řádově několik dní. Pokud bychom k překladu 20 milionů vět použili Překladač Google, jsme zase omezení cenou za přístup k překladovému API společnosti Google.

Pro práci s Mosesem jsme zvolili překladový a jazykový model pro překlad z angličtiny do češtiny, které jsou poskytnuty přímo s Mosesem ve verzi XX. Obrázek 4.1 porovnává překlad metadat k jednomu z obrázků datasetu Profimedie v Mosesovi s distribuovanými překladovými daty s překladem pomocí Překladače Google. Je vidět, že Překladač Google poskytuje kvalitnější překlad.

4.5 Překlad slov

Jednodušším přístupem k překladu klíčových slov je přístup slovníkový, tedy překlad každého slova zvlášť. Nejprve je potřeba ze souboru klíčových slov u všech obrázků vyextrahovat slova. Ty pak lze přeložit přímo s použitím slovníku, nebo pomocí frázového strojového překlad (ten použije jednoslovné fráze) Mosesem, či Překladačem Google. Výhodou oproti předchozímu přístupu je menší množství dat a tedy i nižší finanční a časová náročnost takového překladu. Takový systém překladu ale nedokáže detekovat fráze a kvalita překladu slovo od slova je typicky horší než překlad delších frází. Vezměme si například anglická slova „weather“ a „vane“. Slovníkový překlad nám slova přeloží jako „počasí“ a „lopatka“. Pokud se ale tato slova nachází vedle sebe, je pravděpodobnějším překladem slovo „korouhvička“.

4.6 Překlad frází

Poslední možností je oba předchozí principy zkombinovat — nejprve detekovat v souboru klíčových slov fráze a ty pak přeložit statistickým strojovým překladem. Detekci frází z datasetu Profimedia provedl ve své bakalářské práci[5] Jan Botorek. Zkoušel detekovat N-gramy v databázi WordNet[25] a Wikipedii. Výsledky této detekce frází lze použít právě ke zlepšení překladu z angličtiny do cizích jazyků. Na slova, která nejsou detekována ve frázi, se použije slovníková metoda. Na překlad detekovaných frází lze použít přímo statistický strojový překlad.

4.7 Řešení

Výsledná aplikace nakonec používá poslední možnost. Bližší informace jsou v Kapitole 10. Slova byly přeloženy pomocí Překladače Google, který překládá s lepšími výsledky než dostupný model pro překladový nástroj Moses.

Překlad frází by ovšem pomocí Překladače Google byl příliš drahý a pomocí Mosese zase příliš pomalý. Použili jsme tedy jednoduchou metodu, která používá pouze překladový model. Fráze z datasetu Profimedia byla přeložena pouze tehdy, když se její překlad nacházel v překladovém modelu. Fráze, které se v překladovém modelu nenacházely, byly přeloženy slovo od slova. Metodou přesné shody s překladovým modelem bylo přeloženo 18 006 z celkového počtu 899 244 detekovaných frází v Profimedia datasetu.

4.8 Závěr

Překlad klíčových slov z korpusu Profimedia není typickou překladovou úlohou — nepřekládají se celé věty. Přesto je dokonalý výsledek nemožný. I lidští překladatelé s citem pro jazyk by v této překladové úloze dávali rozdílné výsledky. Strojový překlad zdaleka není na takové úrovni, aby dokázal z širšího kontextu vybrat správný překlad. Navíc lidský překladatel může u překladu klíčových slov využít přímo obrázek, ke kterému se klíčová slova vztahují. Může tak z obrázku posoudit, zda má slovo „single“ přeložit jako „jednolůžkový“, nebo ve významu „jeden“. Lepší překlad by mohl přinést překladový model natrénovaný na specifičtější množině dat bližší korpusu Profimedie. Vytvořit takový model by ovšem bylo nad rámec této práce.

Navrhovaný mechanismus překladu poskytuje uspokojivý, i když značně nedokonalý, překlad klíčových z angličtiny do češtiny. Jednoduše jde zevšeobecnit i pro překlad do dalších jazyků, pro které máme slovník a překladový model.

5. Stemmer

Důležitou vlastností systémů na vyhledávání v textu je prohledávání ve více tvarech hledaného slova. Pokud uživatel hledá slovo „praha“, ve většině případů očekává, že se mu zobrazí i výsledky obsahující slovo „praze“. Je tedy potřeba mít nějaký algoritmus, který k sobě slova „praha“ a „praze“ přiřadí.

První možností je použít lemmatizér. Úkolem lemmatizéru je ke každému slovu přiřadit jeho základní tvar. U podstatných jmen je to většinou první pád jednotného čísla („praha“), u sloves infinitiv.

Alternativou lemmatizátoru může být stemmer. Stemmer nemusí, na rozdíl od lemmatizéru, nemusí vrátit regulérní slovo jazyka. Například pro slova „praha“ a „praze“ může stemmer vrátit slovo „prah“. Výhodou stemmeru oproti lemmatizéru je, že většinou používá pouze jednoduché heuristiky. Je tedy často rychlejší a méně náročný na zdroje než lemmatizér. Ve vyhledávacích aplikacích je stemmer dostačující.

Pro angličtinu je nejznámějším stemmerem pro angličtinu je Porterův stemmer[26] popsaný Martinem Porterem již v roce 1980. Kromě oficiální implementace existují porty do různých jazyků včetně Ruby. Aplikace využívá implementaci z Ruby Gemu „stemmify“[1] v licenci MIT.

5.1 Český stemmer

Pro jazyky jako je čeština, která má bohatší morfologii než angličtina, je tvorba stemmeru náročnější. Testovali jsme několik implementací českých stemmerů. Jako nejkvalitnější byla nakonec vybrána implementace českého stemmeru v knihovně Lucene. Tuto implementaci využívá i Elasticsearch.

V rámci této práce byl portován soubor `CzechStemmer.java` ze zdrojového kódu knihovny Lucene do jazyka Ruby. Výsledkem je Ruby Gem pod licencí MIT, který lze použít nezávisle na zbytku aplikace. Instaluje se příkazem

```
gem install czech-stemmer
```

Knihovna obsahuje pouze jednu třídu CzechStemmer s funkcí stem, která přijímá i vrací řetězec:

```
require 'czech-stemmer'

CzechStemmer.stem("praha") # => "prah"
CzechStemmer.stem("praze") # => "prah"
CzechStemmer.stem("předseda") # => "předsd"
CzechStemmer.stem("mladými") # => "mlad"
```

[1] `https://rubygems.org/gems/stemmify`

6. Detekce jazyka

Aplikace může přijímat vstup ve více jazycích. Jedním ze způsobů, jak uživateli zpříjemnit práci s aplikací, je použít automatický detektor jazyka. Při zadání vstupu aplikace se aplikace sama pokusí detekovat jazyk zadaného textu. Tato detekce nemusí být vždy stoprocentně správná, takže by uživatel měl mít možnost volbu jazyka manuálně změnit.

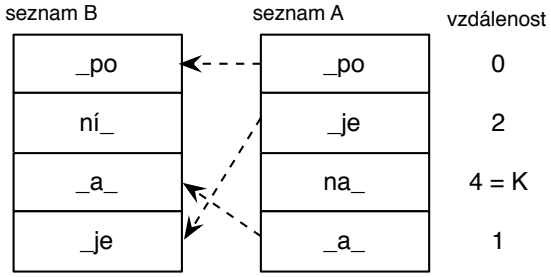
Aplikace používá algoritmus založený na statistice nejčastějších N-gramů pro daný jazyk[8]. Algoritmus nejprve použije velký korpus textů pro každý detekovaný jazyk. Pro češtinu a angličtinu lze použít například korpus Wikipedie. Z tohoto korpusu získá uspořádaný seznam K nejčastějších N-gramů. Při samotné detekci jazyka textu, pak samotný algoritmus získá stejný uspořádaný seznam nejčastějších N-gramů pro vstupní text. Tento seznam pak porovnává se seznamy nejčastějších N-gramů pro každý jazyk.

Nechť A a B jsou seznamy N-gramů s K položkami, $A[w]$ je pořadí N-gramu w v seznamu A. Potom lze vzdálenost mezi seznamy $D(A, B)$ vyjádřit vztahem:

$$D(A,B) = \sum_{w \in A} \begin{cases} |A[w] - B[w]| & \text{pokud } w \in B \\ K & \text{jinak} \end{cases} \tag{6.1}$$

Na Obrázku 6.1 je ukázka porovnání dvou seznamů pomocí funkce $D(A, B)$. Trigram *na_* se v seznamu B nenachází, proto dostane hodnotu $K = 4$.

Nyní pokud máme množinu S všech seznamů N-gramů pro jednotlivé jazyky a X je seznam N-gramů pro vstupní text, vrátí algoritmus jako jazyk takový jazyk, pro který má seznam $C \in S$ minimální hodnotu $D(X, C)$.



Obrázek 6.1: Ukázka porovnání dvou trigramových seznamů. $D(A, B) = 7$

Naše implementace používá v seznamech N-gramů trigramy. Pro texty delší než několik slov funguje velmi spolehlivě. Pro několikaslovné texty se může stát, že v seznamu nejfrekventovanějších trigramů pro vstupní text není ani jeden trigram, který by se nacházel v seznamech pro jednotlivé jazyky. Pak algoritmus může vrátit špatně detekovaný jazyk.

Vylepšení by jistě přineslo spolu s použitím trigramů použít i bigramy a unigramy. Také konstanta K by šla zvětšit z používané hodnoty 50 výš. Implementace detekce jazyka ale probíhá na klientovi, takže všechna tato vylepšení algoritmu by zvýšila množství dat, která si klient musí stáhnout. Navíc uživatel má vždy možnost detekovaný jazyk manuálně změnit a typicky pracuje spíše s delšími tex-

ty. Implementace s použitím 50 nejčastějších trigramů se tedy zdá dostačující pro daný účel.

7. Podobné obrázky

Jednou ze služeb, které výsledná aplikace poskytuje, je vyhledávání podobných obrázků. Uživatel rozhraní najde pomocí textových dotazů ilustrační obrázky a má možnost u každého z nalezených obrázků získat obrázky vizuálně podobné. Tato kapitola pojednává o tvorbě backendové služby, která vyhledávání podobných obrázků umožňuje.

Vstupními daty je soubor s vektory pro každý obrázek datasetu Profimedie. Vektor má 4096 složek s reálnými nezápornými čísly. Vektory jsou vizuální deskriptory obrázků. Tyto deskriptory byly vygenerovány pomocí software Caffe[19] a jsou jsou odezvami předposlední vrstvy hluboké konvoluční neuronové sítě natrénované pro klasifikaci obrázků z datasetu Profimedie do 1000 kategorií.

Mějme obrázek I_1 s deskriptorem D_1 a obrázek I_2 s deskriptorem D_2. Míru podobnosti obrázků *Similarity* pak můžeme definovat jako

$$Similarity(I_1, I_2) = \sum_{i=1}^{4096} |D_1[i] - D_2[i]|. \tag{7.1}$$

V praxi se dají výsledky této míry klasifikovat zhruba do 3 kategorií. Tyto vypozorované kategorie popisuje tabulka 7. Ukázky jednotlivých kategorií podobných obrázků poskytuje obrázek 7.2. Rozdělení na 3 kategorie podobnosti podle míry *Similarity* je pouze přibližné. Nelze například zaručit, že obrázek, který by některý uživatel mohl uznačit za podobný, nebude mít míru *Similarity* vyšší než 1500.

Kategorie	*Similarity*
téměř shodné	$0 - 500$
podobné	$500 - 1500$
nepodobné	> 1500

Obrázek 7.1: Kategorie podobnosti obrázků podle *Similarity*.

Otázkou zůstává, které výsledky uživatel očekává jako výsledky vyhledávání podobných obrázků. Pravděpodobně nechce získat obrázky z kategorie „nepodobné“. Pak je otázkou, jestli uživatel chce jako výsledek získat obrázky z kategorie „téměř shodné“. V korpusu je spousta podmnožin obrázků, které byly vyfoceny v rámci jedné série. Často se liší jen malou změnou úhlu fotky, nebo jen kompresí uloženého obrázku. Pokud bychom vraceli obrázky seřezené podle *Similarity*, uživatelé u takovýchto podmnožit nedostanou příliš rozmanité obrázky. Vracet obrázky seřazené podle *Similarity* tedy nemusí být vždy vhodné, náhodné pořadí obrázků z kategorií „téměř shodné“ a „podobné“ může uživatel chápat jako lepší výsledek. Tato úvaha umožňuje použít algoritmy, které nevrací obrázky seřazené podle *Similarity*, ale jsou výrazně rychlejší.

7.1 Bootstrap implementace

První implementace nahrála všechny deskriptory do databáze Elasticsearch. V Elasticsearch lze implementovat vyhledávání pomocí vlastní porovnávací funkce.

Obrázek 7.2: Dvojice obrázků s různou kategorií podobnosti.

Jako porovnávací funkci tedy zvolíme funkci *Similarity*. Bohužel i přes některé menší optimalizace algoritmu se vyhledávání nepodařilo implementovat efektivně, takže fungovalo v přijatelném čase pouze pro pár tisíc deskriptorů. Naše aplikace ovšem potřebuje pracovat s více než dvaceti miliony deskriptorů. Bylo by možné použít další optimalizace a sharding databáze na více strojů, aby algoritmus fungoval efektivněji i na větším množství dat. To by ale bylo příliš drahé a navíc se objevily jiné možnosti řešení.

7.2 Předgenerování výsledků

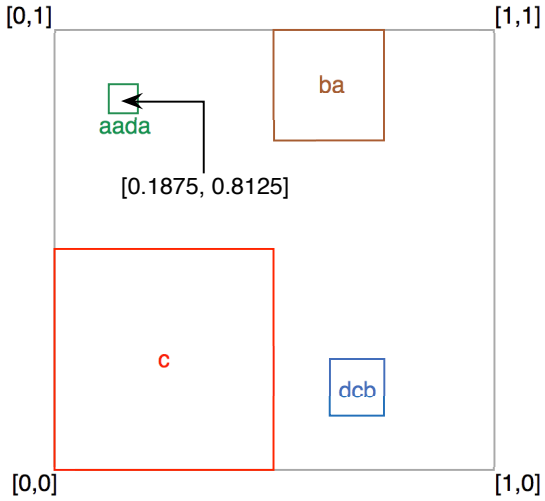
Efektivnější variantou je ke každému obrázku vygenerovat nějaké množství podobných obrázků předem. Služba, která by vracela podobné obrázky, by pouze vrátila položku z databáze a neprováděla žádný výpočet. Tato služba by tak byla velmi rychlá a nenáročná na zdroje. Jako problém se ovšem ukázalo právě předgenerování obrázků. Přes snahu o co nejrychlejší implementaci v jazyce go s použitím více vláken a optimalizačních heuristik, by vygenerování podobných obrázků pro každý z 20 milionů obrázků v datasetu Profimedie trvalo na běžném počítači několik týdnů.

7.3 Geohash

Další pokus o implementaci vyhledávání podobných obrázků se inspiruje algoritmem Geohash[29]. Algoritmus Geohash byl vyvinut v rámci služby geohash.org a jedná se o způsob zakódování prostorových dat. Jeho hlavním využitím je efektivní vyhledávání bodů (určených zeměpisnými souřadnicemi) v oblasti (na mapě). Algoritmus Geohash využívá i Google, nebo databáze Elasticsearch.

Zjednodušení algoritmu Geohash popíšeme na jednotkové podmnožině (jednotkovém čtverci) $\mathbb{R}^2$. Algoritmus postupuje tak, že čtvercovou plochu rozdělí na 4 čtverce a pojmenuje je písmeny „a“, „b“, „c“ a „d“. Každý ze čtverců rekurzivně rozdělí na další čtyři čtverce, kterým dá jméno pomocí suffixů „a“, „b“, „c“ a „d“ k vlastnímu jména. Rekurzi provádíme do nějaké předem určené hloubky. Bodu v jednotkovém čtverci přiřadíme jméno podle čtverce s nejdelším jménem, který bod obsahuje. Každý bod v jednotkovém čtverci tedy bude mít jméno, které má stejnou délku jako hloubka rekurze. Jako oblast pak můžeme označit jakoukoliv množinu pojmenovaných čtverců. Bod leží v oblasti právě tehdy, když je jméno nějakého čtverce z množiny oblasti prefixem jména bodu. K ukládání názvů bodů pak lze použít prefixový strom.

Obrázek 7.3 ukazuje některé čtverce a jejich názvy. Pokud má algoritmus hloubku rekurze 4, má bod ležící na souřadnicích $[0.1875, 0.8125]$ název *aada* a leží tedy ve čtvercích *a*, *aa*, *aad* a *aada*.

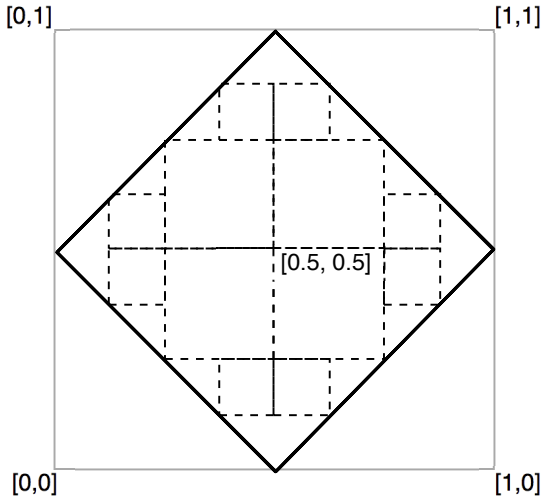


Obrázek 7.3: Ukázka čtverců algoritmu Geohash v jednotkovém čtverci.

Nyní bychom chtěli algoritmus Geohash použít pro hledání podobných obrázků v Profimedia datasetu. Každému deskriptoru bychom nejprve přiřadily název. Množina podobných obrázků by pak obsahovala obrázky, které mají *Similarity* s porovnávaným obrázkem nějak omezenou. Problémem je, že pomocí Geohash čtverců nedokážeme takovou oblat přesně definovat. Můžeme se ale pokusit co nejvíce oblast pomocí čtverců aproximovat.

Pokud bychom chtěli použít Geohash pro hledání podobných vektorů přímo, narazili bychom na několik problémů. Například oblast s ohraničenou mírou *Simiarity* nejde popsat přesně pomocí čtverců. Lze libovolně přesně aproximovat, ale s každou přesnější aproximací vzrůstá počet potřebných čtverců a zhoršuje se

efektivita algoritmu. Obrázek 7.4 ukazuje jak bychom mohli použít Geohash algoritmus pro hledání podobných deskriptorů ve dvojrozměrné dimenzi. Na obrázku je popsána situace, kdy hledáme deskriptory, které mají od bodu $[0.5, 0.5]$ vzdálenost 0.5. Tučně ohraničený čtverec vyznačuje hledanou oblast, 12 čárkovaných čtverců tvoří aproximovanou oblast.



Obrázek 7.4: Ukázka čtverců algoritmu Geohash v jednotkovém čtverci.

Ve dvou dimenzích by toto řešení fungovalo poměrně dobře. Deskriptory obrázků ovšem mají 4096 složek a počet potřebných čtverců pro stejnou míru aproximace roste s každou dimenzí exponenciálně.

7.4 Řešení

Naše řešení využívá princip algoritmu Geohash. Jde nám o to, převést hledání podobných vektorů na fulltextové vyhledávání. Vektory reálných čísel převedeme na množinu slov oddělenou mezerami — řetězec. Slova pochází z množiny $\{0, 1, \ldots 4095\}$. Každé složce vektoru deskriptoru tedy odpovídá právě jedno slovo z množiny slov. Nyní přiřadíme každému deskriptoru nějakou nějakou podmnožinu slov. Způsob přiřazení je klíčovým faktorem algoritmu, který ovlivňuje jeho efektivitu. Základním metodou přiřazení je přiřadit vektoru slova, která odpovídají nenulovým vektorům. Na deskriptorech obrázků z datasetu Profimedie to znamená, že každý vektor bude mít přiřazeno zhruba 1500 slov. To je stále příliš mnoho slov pro efektivní fulltextové vyhledávání. Elasticsearch v základní konfiguraci ani neumožňuje vyhledávat s dotazy delšími než tisíc slov. Musíme tedy množinu slov dále zmenšit. Můžeme například místo nenulovosti zvýšit požadovanou mez velikosti složky vektoru.

V našem řešení používáme jinou metodu. Seřadíme složky vektorů podle velikosti a slova přiřadíme pouze K složkám s nejvyšší hodnotou. Pro $K = 4$ pak každému obrázku přiřadíme řetězec, například "2013 432 1065 3433". Při vyhledávání podobného obrázku nejprve získáme jeho deskriptor a jeho přiřazený řetězec. Pomocí řetězce najdeme fulltextovým vyhledáváním L obrázků. Výsledky seřadíme podle podobnosti s hledaným obrázkem mírou *Similarity* a vrátíme

obrázky, které jsou v kategorii „podobné“, nebo „téměr shodné“. Pro správné fungování je nutné dobře nastavit hodnoty K a L. Jejich zvýšení vede k větší přesnosti na úkor rychlosti algoritmu.

7.5 Implementace

Celá služba je implementována nezávisle na zbytku projektu. Jedná se o program `similar_img_finder` napsaný v jazyce go. Jako databáze je použita Elasticsearch. Technické detaily služby vyhledávání podobných obrázků jsou uvedeny v Sekci 10.7.

8. Backend

Moderní webové aplikace lze rozdělit na back a frontend. Backend je část aplikace, která běží na serveru. Pomocí svých služeb poskytuje přístup k databázi, k souborům uloženým na serveru a zpracovává uživatelské operace. Frontend těchto služeb využívá. Tato kapitola popisuje proces výběru backendových technologií pro tuto práci. Některé technologie se osvědčily, jiné se ukázaly pro daný účel nevhodné.

8.1 Databáze

Úkolem databáze je uložit data a umožnit jejich prohledávání. Důležitou vlastností naší aplikace je, že uživatelé nemají možnost databázi modifikovat. Zápis do databáze provede administrátor pouze jednou, před startem aplikace. Důležitým požadavkem je důraz na rychlost a snadnou škálovatelnost. V posledních letech vzniklo několik nových databází v kategorii vágně označené jako NoSQL[28]. Tato kategorie databází se těžko popisuje, na každou popsanou vlastnost existuje NoSQL databáze, která danou vlastnost nesplňuje. Obecně ale lze říct že NoSQL databáze nepracují s prvky v tabulkovém uspořádání a oproti standardním SQL databázím nekladou tolik omezujících požadavků na data. Jejich výhodou oproti standardním relačním databázím může být vyšší výkon a snadná škálovatelnost.

Nevýhodou je většinou obtížnější práce s daty. Většina NoSQL databází například mapodporuje databázové transakce a vůbec celý ACID. Pro práci s daty v NoSQL databázi nelze použít klasické SQL dotazy. Místo nich se používá například model MapReduce[12] vyvinutý ve formě Google.

První verze aplikace byla postavena na databázi CouchBase[10], což je právě jedna z NoSQL databází podporující MapReduce mechanismus. Výhodou CouchBase je vysoký výkon, snadná škalovatelnost, velmi dobrá dokumentace a také existence oficiálních knihoven pro nejrozšířenější programovací jazyky. Ukázalo se však, že pro implementaci vyhledávání pro naší aplikaci je model MapReduce nedostatečný a implementace vyhledávání v CouchBase by byla prakticky nemožná.

Vhodnější pro daný účel se ukázala knihovna Elasticsearch[4]. Nejedná se v pravém smyslu o databázi. Jejím hlavním cílem je poskytnout vyhledávání nad daty. Je postavená nad knihovnou Apache Lucene ke které přidává snadnou horizontální i vertikální škálovatelnost a komunikaci pomocí REST HTTP JSON API. Díky tomu, že je Elasticsearch postaven na knihovně Lucene[1], může programátor využít velkou množinu možností, které Lucene poskytuje. Například v textovém vyhledávání může využít všechny tokenizery a stemmery z knihovny Lucene. V průběhu implementace aplikace navíc vyšla stabilní verze 1.0.

8.2 Programovací jazyk

Volba programovacího jazyka je při tvorbě backendové části klíčová. Tato sekce rozebírá některé výhody a nevýhody třech programovacích jazyků.

8.2.1 NodeJS

JavaScript je jazyk především pro programování frontendu a dlouho byl brán jako nutné zlo — bez použití problematických rozšíření nejde ve webovém prohlížeči programovat jiným jazykem. Každý větší webový projekt tak byl nucen používat nejméně 2 technologie — jednu pro frontend a jednu pro backend. Knihovna NodeJS toto paradigma obrací, umožňuje programovat v JavaScriptu i na backendu. Výhodou je sdílení stejného kódu mezi backendem a frontendem. To se hodí například při validacích dat, které potřebujeme kontrolovat na frontendu i backendu.

Původní záměr byl využít pro vývoj naší aplikace právě NodeJS. Nakonec však převážily nevýhody takového řešení.Javascript byl navržen pro programování webového frontendu, jeho standartní knihovna je v porovnání s ostatními jazyky velmi chudá, podpora objektového programování není přímočará. Obsáhlý kód v Javascriptu může být poměrně nepřehledný a jazyk svádí k vytvoření takzvaného „callback hell“. Ukázka takového problémového kódu je v článku [18]

```
doAsync1(function () {
  doAsync2(function () {
    doAsync3(function () {
      doAsync4(function () {
    })
  })
})
```

Pokud chce navíc programátor sdílet kód z backendu i na frontendu, musí být kód kompatibilní s podporovanými prohlížeči. Pokud aplikace potřebuje podporovat starší verze prohlížečů, nemůže sdílený kód obsahovat novinky z páté verze ECMAScript[31].

8.2.2 Go

Go (známý také jako golang)[16] je staticky typovaný programovací jazyk od společnosti Google. Syntaxe je inspirována jazykem C s důrazem rychlou kompilaci. Velkou výhodou je snadná práce s vlákny pomocí „goroutines“. Programátoři v Javě, nebo C++ může překvapit poněkud netypická podpora práce s objekty.

První verze aplikace byla napsána právě v jazyce Go. Později byla přepsána většina v jazyce Ruby. Výhodou řešení v jazyce Go je velká výkonost a stabilita aplikace. Zkompilovaná aplikace je pouze jeden binární soubor, takže oproti řešení v nekompilovaných jazycích se lépe distribuuje a odpadají problémy se závislostmi.

Go je stále poměrně mladý jazyk, první stabilní byla vydána teprve v roce 2012[15]. Největším problémem takto mladých jazyků je nedostatek kvalitních knihoven. Pokud už na daný problém existuje knihovna, těžko se odhaduje, jestli její autor projekt neopustí a jestli bude knihovna podporována i další rok. Pro naší aplikaci se ukázal jako problém neexistence kvalitní knihovny pro komunikaci s databází Elasticsearch. S tou jde komunikovat pomocí HTTP requestů, takže jsme pro naší potřebu vytvořili jakousi mikroknihovnu. Experimentální povaha aplikace ovšem vyžadovala rychlé prototypování a úprava kódu na komunikaci

s Elasticsearch začala být příliš omezující. Pro naše účely se ukázalo výhodnější přepsat hlavní část aplikace do dynamického jazyka s lepší podporou pro Elasticsearch.

V jazyce Go je napsána nezávislá komponenta vyhledávání podobných obrázků, která je popsána v Kapitole 7.

8.2.3 Ruby a Ruby on Rails

Ruby[24] je dynamicky typovaný jazyk, inspirovaný jazyky jako Perl, nebo Smalltalk, s důslednou podporou objektového programování. Popularita Ruby vzrostla zejména díky webovému frameworku Ruby on Rails (Rails)[27], který je v Ruby napsaný. Rails zpopularizovaly architekturu Model-View-Controller[30] pro tvoru webových aplikací. Backend aplikace s vyjímkou služby pro vyhledávání podobných obrázků, byl nakonec přepsán právě do Rails. Elasticsearch poskytuje pro práci s Ruby oficiálně podporovanou knihovnu[3]. Další výhodou se knihovna Rake, což je obdoba příkazu Make pro skripty v Ruby. Pomocí Rakefilu jdou napsat přehledné úlohy pro manipulaci s daty.

8.3 Komunikace mezi backendem a frontendem

Tato sekce popisuje možnosti komunikace mezi frontendem a backendem.

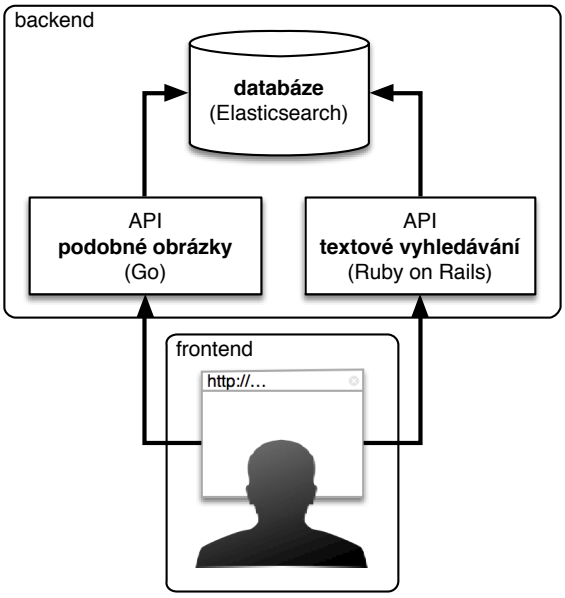
8.3.1 Formát dat

Nejprve je potřeba zvolit formát dat. V současnosti existují dva nejrozšířenější textové komunikační formáty — XML[6] a JSON[11]. XML umožňuje popis velkého množství vztahů mezi entitami. JSON je oproti XML úspornější. XML se hodí v případech, kdy potřebujeme zaznamenat nějaké složité datové struktury. Pro většinu webových aplikací je výhodnější použít JSON. Oproti XML je zápis dat v JSON většinou kratší, což je výhodné zejména při přenosu dat po síti. JSON je založen na datových typech JavaScriptu a moderní prohlížeče umí tento formát zpracovávat nativně. Na frontendu je tedy práce s JSON oproti práci s XML jednodušší. Z těchto důvodů poskytuje backend naší aplikace služby frontendu ve formátu JSON.

8.3.2 REST API

Dále je potřeba specifikovat způsob výměny dat. Jednou z nejrozšířenějších metod je REST, zkratka pro anglické „Representational state transfer“. Jedná se o architekturu rozhraní mezi klientem a serverem. Pro specifikaci rozhraní využívá protokol HTTP. REST využívá k definici práce s daty metody protokolu HTTP „GET“, „PUT“, „POST“ a „DELETE“.

Rozhraní REST[14] s použitím formátu JSON je v aplikaci využíváno pro komunikaci s databází Elasticsearch i pro komunikaci mezi backendem a frontendem.



Obrázek 8.1: Architektura aplikace.

8.3.3 WebSocket

Alternativou k REST API a protokolu HTTP je protokol WebSocket[13]. WebSocket je stejně jako protokol HTTP postaven nad protokolem TCP. Na rozdíl od HTTP ale poskytuje duplexní spojení. Klient je tedy stále spojený se serverem a oba mohou posílat zprávy bez ohledu na druhou stranu. Spojení přes WebSocket má většinou menší latenci než použití protokolu HTTP[22]. Protokol je podporován všemi moderními verzemi webových prohlížečů a podpora existuje i v knihovnách pro backendové programovací jazyky.

První verze aplikace používali pro komunikaci mezi klientem a serverem právě protokol WebSocket. Nakonec však převážily nevýhody takovéhoto řešení nad výhodami. Jednou z nevýhod je nutnost udržovat spojení s klientem na serverové i klientské straně. Přináší to několik netriviálních technických problémů. Například v okamžik, kdy se toto spojení přeruší. Pokud se naproti tomu přeruší spojení server-klient při HTTP požadavku, může klient zkusit vyslat stejný požadavek znovu.

Druhým problémem je emulace HTTP požadavků v protokolu WebSocket. Z klienta můžeme odeslat HTTP dotaz na server a dostaneme k němu přiřazenou odpověď. V protokolu WebSocket pošleme serveru zprávu a za nějaký čas můžeme dostat zprávu od serveru jako odpověď. Párování došlých zpráv z odeslanými zprávami — požadavky — ale musíme implementovat sami, například pomocí unikátních identifikátorů v těle zprávy. Naše implementace navíc musí umět řešit situaci, kdy žádná odpověď ze serveru nedojde. Například nastavením timeoutu pro čekání na odpověď.

Řešení pomocí protokolu WebSocket využijí zejména aplikace, které potřebují komunikovat obousměrně mezi klientem a serverem, nebo je pro ně důležitá nízká latence získání odpovědi. Takovými aplikacemi mohou být chatovací služby, nebo online hry. Pro většinu ostatních aplikací budou zatím spíše převažovat nevýhody obtížnějšího technického řešení nad přínosy technologie WebSockets.

8.4 Shrnutí

Architektura je zobrazena na obrázku 8.1. Část aplikace, která se stará extrakci klíčových slov a textové vyhledávání obrázků je napsána v jazyce Ruby a frameworku Ruby on Rails. Frontend dále komunikuje se službou pro vyhledávání podobných obrázků, která je napsána v jazyce Go. Data jsou uložená v databázi Elasticsearch. Komunikace mezi všemi komponentami probíhá pomocí HTTP REST API, data jsou přenášena ve formátu JSON.

9. Frontend

Frontend je část webové aplikace, která komunikuje přímo s uživatelem, posílá dotazy na služby backendu a zobrazuje uživateli výsledky.

9.1 HTML5

Historicky měli frontendoví weboví developeři na výběr mezi několika technologiemi. Proprietální řešení jako Adobe Flash, nebo Microsoft Silverlight byly dříve jsdnou z mála možností, jak vyvíjet složitější frontendové aplikace. Jejich místo dnes nahrazuje skupina otevřených standardů pod souhrným označením HTML5.

HTML5 je označení jednak pro nejnovější verzi standardu značkovacího jazyka HTML, jednak je to termín zastřešující další moderní webové technologie, jako je například CSS3. HTML5 poskytuje frontendovým vývojářům široké spektrum nových API. Bez proprietálních pluginů je dnes možné v modernch prohlížečích například přehrávat video, nebo vykreslovat 2D grafiku.

Další zlepšení, které frontendovým vývojářům zlehčuje práci, jsou nová verze standardu ECMAScript, což je standard jazyka JavaScript. JavaScript je nyní daleko produktivnější jazyk, než byl ještě před pár lety a může využívat více API ze standardu HTML5.

9.2 JavaScriptové frameworky

Jednotlivé webové prohlížeče se dlouhá léta lišily v implementaci JavaScriptu do takové míry, že pro podporu více prohlížečů bylo použití nějakého frameworku, který sjednocuje API téměř nezbytné. Dnes už do značné míry nejsou při vývoji frontendu frameworku nezbytné, ale stále jsou velmi užitečné.

9.2.1 jQuery

Nejpopulárnějším frameworkem současnosti je jQuery. Ten nabízí jednoduché rozhraní pro práci s DOM a javascriptovými událostmi a pro velké množství menších webových aplikací je dostačující. Ve větších aplikacích se začne projevovat nižší výkon aplikace používající jQuery a také obtížné odstraňování chyb v aplikacu, debuggování.Například můžeme pomocí jQuery chtít přidat CSS třídu „red“ HTML elementu s id „okno“

```
$("#okno").addClass("red");
```

Tento kód proběhne bez nahlášené chyby i v případě, že žádný element s id „okno“ neexistuje, nebo když uděláme překlep v zápisu řetězce. Pokud chce web knihovnu jQuery, musí ji celou nejdříve stáhnout. Aktuální verze 1.11.1 má velikost téměř 100 KB. Knihovna jQuery je použita v implementaci anotačního rozhraní popsaném v Kapitole 11.

9.2.2 Google Closure

Jiný přístup k vývoji frontendových aplikací přináší Google. Pro svou první webovou aplikaci Gmail vyvinul sadu nástrojů, kterou později vydal jako sadu nástrojů Google Closure. Kromě služby Gmail používá Google sadu nástrojů i pro vývoj Google Vyhledávání, Google Map, nebo Google Dokumentů. Skládá se ze tří částí - Closure Compiler, Closure Library a Closure Templates.

Closure Library je javascriptová knihovna funkcí pro práci s DOMem, javascriptovými událostmi, matematickými výpočty a spoustou dalších věcí, které webový programátor může využít.

Closure Compiler je minifikátor javascriptového kódu. Minifikace javascriptového kódu je důležitá, protože snižuje množství dat, které musí webová stránkapřed spuštěním stáhnout. Compiler odstraňuje nedůležité mezery v kódu, odstraňuje funkce, které nejsou volány, přejmenovává názvy funkcí a proměnných na co nejkratší řetězce a v ADVANCED módu se snaží i o pokročilejší optimalizace kódu (například inlineing funkcí). Nejlepích výsledků dosahuje s použitím typových anotací. Anotace jsou silně inspirovány jazykem Java a díky nim lze z JavaScriptu vytvořit typovaný objektový jazyk s privátními metodami, nebo dědičností tříd. Closure Compiler tyto anotace vyhodnocuje a vrací chyby a varování, podobně jako jiné kompilované jazyky. To umožňuje snadněji psát spolehlivý javascriptová kód.

Třetí částí sady nástrojů Google Closure jsou Closure Templates, šablonovací systém pro JavaScript a Javu. Pomocí Closure Templates se snadno vytváří zanořené HTML šablony se zadanými parametry. Closure Templates automaticky ošetřují vstupní data, což zabraňuje bezpečnostním útokům na stránku.

Částí Templates a Compiler jdou použít odděleně v jakémkoliv Javascriptovém projektu. Používat Closure Library bez Compileru nedává příliš smysl, uživatel by při návštěvě webu musel stahovat ohromné množství zbytečných dat.

Frontendová část aplikace je napsána s použitím všech částí Google Closure. Díky tomu je frontendová část serveru tvořena jediným javascriptovým souborem velikosti 65 KB.

9.3 CSS

CSS ve verzi 3 přidává mnoho nových vlastností, které jdou HTML elementům přiřadit. Nejviditelnějšími jsou kulaté okraje, nebo stínování. Pro programátory je zajímavá vlastnost „calc“ pomocí níž lze zadat hodnoty v CSS jako výpočet. Pokud například chceme, aby se element roztáhl na polovinu obrazovky a měl vedle sebe ještě 10 pixelů místo, nastavíme mu vlastnost

```
width: calc("50% - 10px");
```

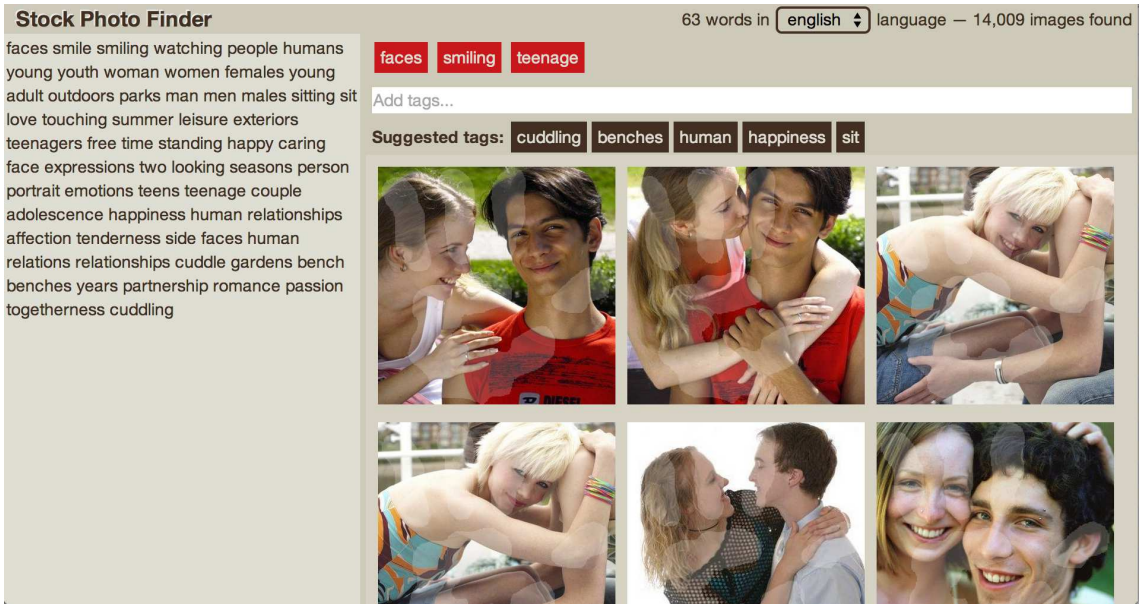
Bez této vlastnosti šlo dříve podobné chování simulovat pouze pomocí JavaScriptu. Vlastnosti jako „calc“ umožňují programátorům snadněji psát responzivní aplikace. To jsou takové aplikace, které se dobře přizpůsobují velikosti okna prohlížeče. Responzivnost webových aplikací začala být důležitá zejména s rozvojem mobilních technologií. Stále větší počet uživatelů přistupuje na weby ze

svých mobilních telefonů, nebo tabletů a tito uživatelé chtějí mít při používání aplikace podobný komfort jako na při používání na počítači. Při vývoji frontendu

9.4 Uživatelské rozhraní

Uživatelské rozhraní aplikace tvoří jediná webová stránka. Uživatel pracuje ve dvou základních režimech — vyhledávání obrázků a detail obrázku.

9.4.1 Rozhraní pro vyhledávání obrázků



Obrázek 9.1: Uživatelské rozhraní pro vyhledávání ilustračních obrázků.

Rozhraní pro vyhledávání obrázků slouží uživateli k vyhledání obrázku podle jím zadaných textových parametrů. Ukázka rozhraní je na Obrázku 9.1. Rozhraní lze rozdělit na 3 části — záhlaví, levý panel a pravý panel. Přetahováním myší za oblast rozdělující levý a pravý panel je možné měnit jejich velikost.

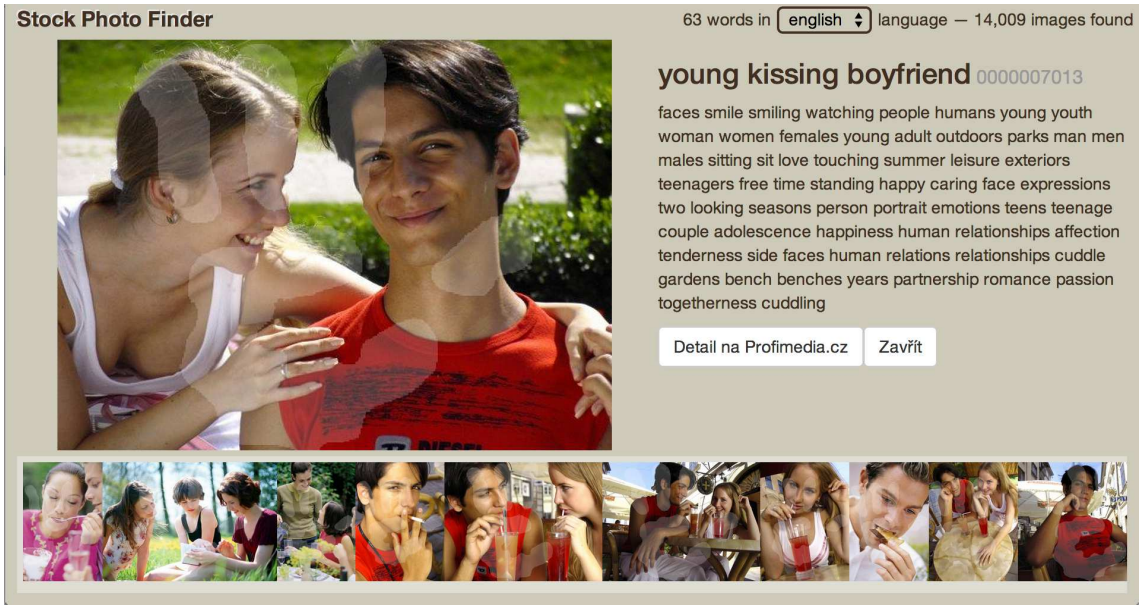
V levém panelu se nachází oblast na vložení textu. Zde může uživatel vložit text článku ke kterému hledá ilustrační obrázky. Uživatelské rozhraní samo detekuje změny v textu a zobrazuje ilustrační obrázky bez toho, aby uživatel musel stisknout tlačítko.

Záhlaví obsahuje statistiky o počtu nalezených obrázků a počtu slov v textu. V rozbalovacím menu je vybrán automaticky detekovaný jazyk vstupního textu. Uživatel má možnost tento jazyk manuálně změnit.

Pravý panel je rozdělen na horní a spodní část. Ve spodní části jsou zobrazeny náhledy nalezených obrázků. Po najetí myší nad obrázek se zobrazí u obrázku i jeho název a klíčová slova z datasetu Profimedie. Celý levý panel funguje na principu „nekonečného“ skrolování — po naskrolování na spodek panelu se načtou další obrázky z výsledků vyhledávání (pokud ještě nějaké nezobrazené výsledky vyhledávání existují). Po kliknutí na obrázek myší se zobrazí rozhraní s detailem obrázku.

Horní část pravého panelu obsahuje komponenty pro explicitní zadání klíčových slov, které uživatel požaduje u vyhledaných obrázků. Klíčová slova může uživatel přidat buď přímo zápisem do textového vstupu „Add tags…“ a stisknutím klávesy `ENTER`, nebo může kliknout na některé z detekovaných klíčových slov pod textovým vstupem. Nad textovým vstupem se na červeném pozadí nachází explicitně požadovaná klíčová slova. Kliknutím na explicitně vybrané slovo se slovo odstraní z požadované množiny klíčových slov.

9.4.2 Rozhraní pro detail obrázku



Obrázek 9.2: Uživatelské rozhraní s detailem obrázku.

Rozhraní s detailem obrázku vyvolá uživatel kliknutím na obrázek ve vyhledávacím rozhraní. Ukázka tohoto rozhraní je na Obrázku 9.2.

V levé horní části vidí uživatel celý obrázek. Pravá horní část rozhraní obsahuje textove popisky obrázku v datasetu Profimedie — název, id a klíčová slova. Pod nimi jsou dvě tlačítka. První je odkazem na detail obrázku na webu Profimedia, druhým tlačítkem uživatel zavře detail obrázku a vrátí se zpět do vyhledávacího rozhraní.

Ve spodní části rozhraní je seznam podobných obrázků k obrázku v detailu. Po kliknutí na některý z podobných obrázků se zobrazí detail příslušného obrázku.

9.5 Závěr

Frontend aplikace zpřístupňuje uživateli prohledávání datasetu Profimedie. Při vývoji byl kladen důraz na uživatelskou přívětivost a použití nejmodernějších frontendových technologií. V celém rozhraní jsou tak například pouze dvě tlačítka. Rozhraní se samo přizpůsobuje velikosti obrazovky uživatele. Pro uživatele komfortní by měla být i automatická detekce jazyku vloženého textu, nebo možnost měnit velikost panelů ve vyhledávacím rozhraní.

10. Instalace a zprovoznění

Backend aplikace je webová aplikace napsaná v jazyce Ruby a frameworku Ruby on Rails. Je k dispozici pod svobodnou licencí MIT. K jejímu spuštění potřebujete ruby verze alepoň 2.0 (nižší verze nejsou otestované), Javu a ke stažení zdrojového kódu git. Aplikaci jde spustit na platformách Linux a Mac.

10.1 Instalace aplikace

Zdrojový kód je volně dostupný na webu GitHubu[1]. Stáhnou tedy lze příkazem

```
git clone https://github.com/hypertornado/diplomka
```

Tento příkaz vytvoří adresář diplomka. Závislosti aplikace nainstalujete pomocí bundleru:

```
bundle install
```

Instalace může vyžadovat přístup administrátora. Dále je potřeba stáhnout knihovnu Elasticsearch[2] do adresáře bin/elasticsearch. Stačí verze 1.0 a vyšší.

Nyní je možné celý projekt spustit pomocí Bash příkazů. Nejprve se spustí elasticsearch databáze pomocí příkazu `rake es:start`, poté je možné spustit samotnou aplikaci příkazem `rails server`. Po spuštění severu je uživatelské rozhraní dostupné ve webovém prohlížeči na adrese `http://localhost:3000`. Po načtení stránky se zobrazí uživatelské rozhraní, ale AJAXové dotazy skončí chybou. V databázi nejsou importována data.

10.2 Vložení vstupních dat

Metadata k obrázkům a obrázky samotné jsou poskytovány firmou Profimedia a nejsou volně dostupné. Ke zprovoznění aplikace je nutné vložit CSV soubor `profi-text-cleaned.csv` obsahující dataset Profimedie do adresáře `data`. Dále potřebujeme do adresáře `data` vložit soubor `keyword-cleaned-phrase-export.csv` z bakalářské práce Jana Botorka, který obsahuje dataset Profimedie s detekovanými frázemi.

10.3 Překlad

Pro překlad použijeme metodu popsanou v Kapitole 4. K překladu budeme potřebovat frázovou tabulku pro anglicko-český překlad. Použijeme tu, která je vydána s překladovým nástrojem Moses verze 2.1[3].

Nyní můžeme použít příkaz

[1] `https://github.com/hypertornado/diplomka`

[2] `http://www.elasticsearch.org/downloads/1-0-3/`

[3] `http://www.statmt.org/moses/RELEASE-2.1/models/en-cs/model/`

```
rake data:export_profimedia_words_for_translation
```

který vytvoří soubor `word_list.txt` s 352 862 slovy. Nyní je potřeba vytvořit soubor s překladem všech slov. Pro naší aplikaci jsme využili překladový nástroj Překladač Google. Při tak velkém objemu dat jsme byly nuceni použít přístup přes placené API, které společnost Google poskytuje. Překladové API jsme použili skrz překladový nástroj firmy Memsource. Tímto strojovým překladem jsme získali soubor `word_list_translated_cs.txt` se všemi slovy z datasetu Profimedia přeloženými do češtiny. Příkazem

```
rake data:create_word_dictionary
```

můžeme seznam anglických a českých slov spárovat do jednoho souboru `word_dictionary_e`

Kvůli lepší kvalitě překladu jsme se snažili přeložit i detekované fráze z datasetu Profimedie. Ze souboru `keyword-cleaned-phrase-export.csv` se fráze exportují příkazem

```
rake data:export_profimedia_phrases_for_translation
```

do souboru `phrases_list.txt`. Příkaz

```
rake data:translate_phrases
```

pak tyto exportované fráze porovná s frázemi přítomnými v překladovém modelu Mosese. Fráze, které se v překladovém modelu nachází jsou pak i s překladem uloženy v souboru `phrase_table_en_cs.txt`.

Získali jsme tedy překladový slovník pro slova (`word_dictionary_en_cs.txt`) i fráze (`phrase_table_en_cs.txt`). Algoritmy, které tato data používají jsou implementovány v souboru `language_tool.rb`.

10.4 Jazykový korpus

Aplikace potřebuje jazykový korpus pro podporované jazyky ze dvou důvodů. Zaprvé je korpus potřeba pro správné fungování algoritmu TF-IDF popsaném v Kapitole 3. Zadruhé je potřeba jazykový korpus pro získání nejfrekventovanějších trigramů jazyka. Tato data se pak využívají v algoritmu pro detekci jazyka, který je popsán v Kapitole 6.

Samotný dataset Profimedie se jako korpus pro naše účely nehodí. Struktura textů v datasetu neodpovídá běžnému textu a navíc je dataset pouze v angličtině. Jako korpus jsme použili data z Wikipedie. Ta jdou stáhnout jako databázový dump ve formátu XML a pod licencí Creative Commons. Stáhli jsme tedy soubory `wiki_dump_cs.xml` a `wiki_dump_en.xml`. Wikipedie nabízí pro převod z XML do obyčejného textu vlastní pythonovský skript `lib/WikiExtractor.py`[2]. Pro extrakci dat z englického XML souboru lze použít příkaz

```
rake wiki:extract_words_from_wiki en
```

který vytvoří adresářovou strukturu s textovými soubory. Není potřeba převést všechna data, pouze tolik, abychom dostali reprezentativní korpus. Pro angličtinu stačí převést zhruba 10 000 článků, pro podobné množství českých dat je potřeba převést zhruba 20 000 článků (údaj o exportovaných článcích je průběžně vypisován na konzoli).

Nyní je možné vytvořit frekvenční seznam slov z korpusu Wikipedie. Ve skutečnosti potřebujeme frekvenční seznam stemů slov. Ten získáme příkazem

```
rake wiki:frequency_list_from_wiki
```

Nyní příkazem

```
rake data:create_tf_df_list
```

získáme pro každý stem v Profimeda datasetu jejich celkovou četnost (TF) a četnost metadat obrázků, kde se stem nachází.

Nakonec spárujeme frekvenční informace o stemech z Profimedia datasetu a korpusu Wikipedie příkazem

```
rake data:pair_profimedia_and_wiki_data
```

Získali jsme dva soubory — `paired_wiki_and_profimedia_cs.txt` pro češtinu a `paired_wiki_and_profimedia_en.txt` pro angličtinu. Soubory obsahují tabulátorem odsazené statistiky četnosti pro všechny stemy v datasetu Profimedie.

10.5 Import dat do databáze

Nyní máme připravená všechna data pro nahrání do databáze. Jako databáze nám slouží Elasticsearch, který musí být připojen na portu 9200. Pro práci s Elasticsearch se velmi hodí plugin elascticsearch-head, který umožňuje prohlížení databáze ve webovém prohlížeči.

Nejprve importujeme frekvenční data pro všechny stemy příkazem

```
rake es:import_word_data
```

Pro každý z podporovaných jazyků takový import trvá zhruba jednu hodinu.

Dále příkazem

```
rake es:import_image_metadata
```

importujeme metadata pro všechny obrázky z datasetu Profimedia. Data jsou importována pro všechny podporované jazyky. Pro neanglické jazyky jsou použity na překlad slovníky slov a frází vytvořené v Sekci . Nahrání těchto dat do databáze může trvat několik hodin. Nahrávací skript zobrazuje odhad zbývajícího času.

10.6 Data pro detekci jazyků

Algoritmus pro automatickou detekci jazyka, popsaný v Kapitole 6 potřebuje ke své práci seznam nejfrekventovanějších trigramů pro každý jazyk. Tento seznam 300 nefrekventovanějších slov pro každý jazyk získáme z korpusu Wikipedie příkazem

```
rake trigrams:extract_most_frequent_trigrams
```

Detekce jazyka probíhá ve frontendové části aplikace (viz Kapitola [**?**]). Příkaz

```
rake trigrams:trigrams_to_javascript_classes
```

vytvoří ze seznamů nejfrekventovanějších seznamů javascriptovou třídu `oo.diplomka.Langu` v notaci Google Closure. Ta je uložena v souboru `public/js/js/diplomka/languages/data.j` Tuto třídu pak využívá frontend pro automatickou detekci jazyka vloženého textu.

10.7 Vyhledávání podobných obrázků

Služba vyhledávání obrázků je nezávislá na zbytku aplikace. Princip jejího fungování je popsán v Kapitole 7. Zdrojový kód služby je v jazyce go a je dostupný na GitHubu[4]. Pro kompilaci služby je potřeba go alespoň verze 1.2. Příkaz `go build` v adresáři se zdrojovými kódy služby zkompiluje program do souboru `similar_img_finder`.

Pro zprovoznění služby je nejprve nutné naimportovat data do databáze příkazem

```
./similar_img_finder -a import -n 1000000 -e 9200 -k
   500 -f data.gz
```

Parametr `-n` určuje, kolik dat se má naimportovat, parametr `-e` určuje na kterém portu běží databáze Elasticsearch (můžeme použít stejnou instanci databáze na které běží hlavní aplikace), parametr `-k` odpovídá hodnotě K z Kapitoly 7 a parametr `-f` je cesta ke komprimovanému souboru s importovanými deskriptory.

Nyní můžeme spustit službu hledání podobných obrázků na portu 8585 příkazem

```
./similar_img_finder -a server -p 8585 -l 100 -e 9200
```

Parametr `-l` odpovídá hodnotě L z Kapitoly 7.

Pokud nyní chceme získat id podobných obrázků k obrázku s id „0000000003“, můžeme využít JSON HTTP API a získat výsledky na adrese `localhost:8585/similar?id=00` Služba vrátí pole obsahující id podobných obrázků a míru *Similarity* pro každý vrácený obrázek.

[4] `https://github.com/hypertornado/similar_img_finder`

10.8 Podpora dalších jazyků

Popsaný postup vytvoří aplikaci, která funguje pro anglický a český vstupní text. Aplikace je ale navržena tak, aby rozšíření do dalších jazyků nebylo obtížné. Popíšeme postup pro přidání dalšího jazyka — francouzštiny. Podle standardu ISO 639-1 je kód francouzštiny „fr“. Upravíme tedy konstantní pole aplikace v souboru `application.rb` — do pole `SUPPORTED_LANGUAGES` přidáme řetězec „fr“ a jako název jazyka přidáme do pole `LANGUAGE_NAMES` řetězec „french“.

Pro každý přidaný jazyk potřebujeme implementovat stemmer. Pro podporu francouzského stemmeru budeme muset rozšířit metodu `stem_word` v souboru `language_tool.rb`. Pro francouzštinu je v Ruby volně dostupných několik knihoven pro převod slov na stemy. Dále ve stejné třídě můžeme pro nově přidaný jazyk rozšířit datovou strukturu `@stopwords` o další stop-slova.

Dále pro francouzštinu stáhneme dump Wikipedie jako soubor `wiki_dump_fr.xml` a zopakujeme celý postup ze Sekcí 10.5, 10.4, 10.5 a 10.6 této kapitoly.

Podobnými kroky můžeme implementovat poměrně přímočaře i podporu pro další jazyky vstupních článků.

10.9 Demo aplikace ve virtuálním stroji

Na přiloženém DVD se nachází virtuální stroj s demoverzí aplikace. Demoverze z důvodů licence datasetu Profimedie a také omezené kapacitě DVD pracuje pouze s prvními 1000 obrázky v Datasetu. Podrobnosti o datech na přiloženém DVD obsahuje příloha

11. Anotační rozhraní

Vrámci této práce bylo implementováno anotační rozhraní pro vyhodnocování algoritmů, které přiřazují vhodné obrázky k textům. Anotační rozhraní je velmi univerzální. Anotátor má ve webové aplikaci v levém sloupci novinový text a v pravém sloupci galerii obrázků. Jeho úkolem je označit obrázky, které se k danému textu hodí a obrázky které se k textu nehodí. Má také možnost nechat obrázek neoznačený, pokud by se nemohl rozhodnout ani pro jednu variantu.

11.1 Instalace rozhraní

Celé rozhraní je aplikace napsaná v jazyce Ruby a frameworku Ruby on Rails. Aplikace je volně šiřitelná pod licencí MIT. Pro zprovoznění anotační aplikace je potřeba UNIXový systém (Linux, Mac). Zdrojový kód aplikace je uložen na severu GitHub[1] a nejlépe jde stáhnout pomocí gitu. Pro běh serveru je potřeba verze ruby 2.0 a vyšší. Celá aplikace se zprovozní následujícím pořadím BASH příkazů:

```
git clone https://github.com/hypertornado/cemi_anotace
cd cemi_anotace
bundle install #nainstaluje vsechny ruby zavislosti
rake db:migrate #vytvori sqlite databazi s tabulkami
rails server #spusti anotacni server na portu :3000
```

11.2 Přidání uživatelů

Po spuštění serveru je možné přidat anotátory v administračním rozhraní. Přístup je zaheslován HTTP autentifikací. Defaultní uživatelské jméno je cfo a heslo cfo85. Administrátorské přístupové údaje lze změnit v souboru `ROOT_APLIKACE/app/controlle`
Uživatelé mají pouze dvě datové položky, uživatelské jméno (Name) a heslo (Password). Uživatele jde přidávat, mazat a upravovat. Nepředpokládá se, že by anotovaná data byla vysoce citlivá, heslo je proto v databázi uloženo v plaintextu.

11.3 Import anotačních dat

Data pro anotaci lze nahrát pomocí příkazu

```
rake data:import
```

Příkaz očeká existenci souboru `ROOT_APLIKACE/public/annotation_inputs.csv`. Ten musí mít speciální formát, kdy je každý řádek rozdělen mezerami na šest sloupců s následujícími položkami:

[1] `https://github.com/hypertornado/cemi_anotace`

INDEX
unikátní číslo jedné anotace

LABEL
interní popis pokusu

PRIORITY
priorita, celé číslo >= 0. Určuje prioritu s jakou se má anotace přiřadit. Čím vyšší číslo, tím vyšší priorita.

PREFER_USER
uživatelské jméno preferovaného anotátora. Pokud není žádný anotátor preferován, použije se pomlčka

TEXT_FILE
cesta k textovému souboru s referenčním článkem

IMAGE_FILES
seznam cest k obrázkům. Cesty nemohou obsahovat mezery a jsou oddělené středníkem.

Ukázka importovaných dat:

```
1 basics 0 - ./text/aha/aha-00263.txt.gz
   img/1.jpg;img/2.jpg;img/3.jpg
2 basics 1 - ./text/aha/aha-00006.txt.gz
   img/1.jpg;img/2.jpg
3 basics 0 - ./text/aha/aha-00009.txt.gz  img/2.jpg
```

11.4 Export anotačních dat

Hotové anotace lze exportovat příkazem

```
rake data:export
```

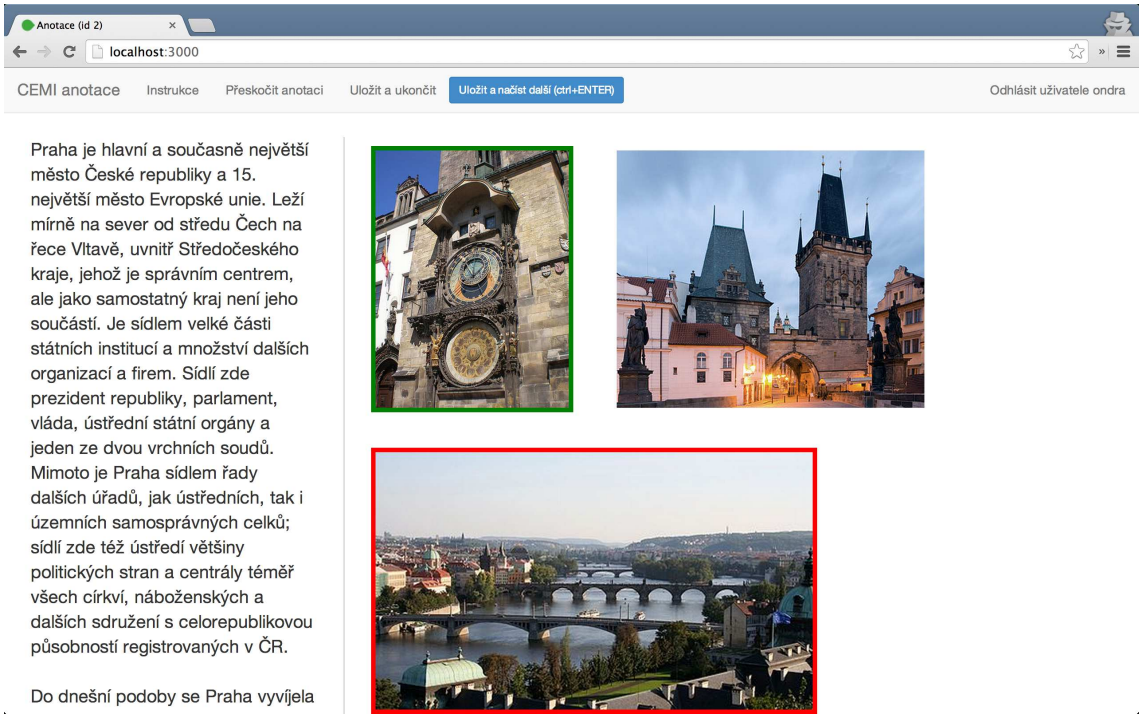
Tento příkaz vypíše na konzolu řádky, které mají tabulátorem oddělené položky:

INDEX
ID anotace. Stejné jako u importovaných dat.

USER
Jméno anotátora, který anotaci vytvořil.

TIME
Čas uložení hotové anotace ve formátu UNIX timestamp.

SKIPPED
Pokud uživatel anotaci přeskočil, je hodnota True, jinak False.



Obrázek 11.1: Anotační rozhraní. Vhodné obrázky jsou označené zeleným rámečkem, nevhodné červeným.

APPROPRIATE
: Seznam obrázků které anotátor označil jako vhodné k textu ve formátu relativních cest oddělených středníkem.

NOT_APPROPRIATE
: Seznam obrázků které anotátor označil jako nevhodné k textu ve formátu relativních cest oddělených středníkem.

11.5 Import obrázků a textů

Anotační texty a obrázky musí být nahrány do adresáře `ROOT_APLIKACE/public` tak, aby jejich cesty odpovídali cestám v souboru `ROOT_APLIKACE/public/annotation_inputs.csv`. Pokud tedy importovaný soubor obsahuje cestu k obrázku `img/1.jpg`, musí být nahrán odpovídající soubor do `ROOT_APLIKACE/public/img/1.jpg`.

Obrázky musí být ve formátu, který podporují webové prohlížeče, tedy hlavně JPEG a PNG. Texty musí být uloženy v textových souborech s kódováním utf-8 a komprimovaný pomocí gzip[2].

11.6 Anotační proces

Úkolem anotátora je přiřadit vhodné a nevhodné obrázky. Po přihlášení do anotačního rozhraní vidí v levé části test a v pravé obrázky. Levým tlačítkem myši může označit obrázky, které odpovídají textu, pravým tlačítkem myši označí

[2] `http://www.gzip.org/`

obrázky, které textu neodpovídají. Pokud si anotátor není jistý, nechá obrázek neoznačený. Uživatel může použít klávesovou zkratku ctrl+ENTER k uložení a načtení další anotace. Může také anotaci přeskočit (pak se označí jako přeskočená a nepřiřadí se jinému anotátorovi), nebo uložit a ukončit.

12. Testování výsledků

Výsledný program se skládá z několika komponent. Funkčnost každé komponenty jde testovat. Některé části jdou otestovat automaticky, například rozpoznávání jazyka. Na některé části je potřeba testování s uživateli.

Klíčovou částí celého projektu je algoritmus, který k textu přiřadí ilustrační obrázek. Úspěšnost tohoto algoritmu byla otestována na uživatelích. Rozhraní může fungovat pro více jazyků. Testování však proběhlo pro vstupní texty v češtině. Jedním z důvodů je snadnější získání anotátorů v českém jazyce. Hlavním důvodem je ovšem pravděpodobná vyšší nepřesnost algoritmu v češtině. Dataset Profimedie má data v anglickém jazyce a pro české použití musel být přeložen. Pokud se tedy ukáže, že algoritmus funguje dobře pro češtinu, měl by pro angličtinu fungovat ještě lépe. Pro testování bylo využito testovací rozhraní popsané v Kapitole 11.

12.1 Vyloučení narušitele (o_test1)

První metodou testování bylo „vyloučení nepřítele“ (anglicky „intruder detection“). Tato metoda se používá k evaluaci automaticky detekovaných shluků slov a je popsána například v [9]. V naší variantě se evaluují obrázky přiřazené k textu. Nejprve se k danému textu najdou pomocí testovaného algoritmu 4 nejvíce odpovídající obrázky. K nim se přidá jeden náhodně vybraný obrázek z datasetu a poté se náhodně zamíchá pořadím těchto obrázků. Anotátor vidí v anotačním rozhraní text a 5 obrázů. Jeho úkolem je označit obrázek, který danému textu, podle jeho názoru, odpovídá nejméně. Pokud algoritmus přiřazující obrázky textu funguje správně, měl by být uživatel schopen označit obrázek, který byl do sady vybrán náhodně a rozlišit ho od obrázků, který vybral algoritmus přiřazující obrázky.

Texty pro testování algoritmu pochází z online článků českých webových serverů. Jedná se o texty článků, které obsahují ilustrační obrázky z datasetu Profimedie. Takto omezená množina novinových textů je pro náš účel velmi výhodná. Pro různé druhy článků dataset Profimedie neobsahuje žádné vhodné ilustrační obrázky. Jedná se například o politické zpravodajství, pro které v datasetu aktuální fotky událostí, nebo například archivní fotky osobností. Oproti tomu u článků, které již nějaký ilustrační obrázek z Profimedie obsahují, jsou možnosti vhodného obrázku daleko vyšší. Jedná se většinou o různé hobby a společenské články.

Pro naše testování jsme využili články ze serveru iDnes, který obsahoval nejvíce článků s ilustračními obrázky z Profimedia. Konkrétně jich máme k dispozici 4223.

Z těchto 4223 článků jsme náhodně vybrali pro testování 60 článků. Ke každému z článků byly získány algoritmem 4 doporučené ilustrační obrázky a byl přidán jeden obrázek náhodný. Každá tato testovací sada byla otestována dvěma anotátory. Bylo k dispozici 5 anotátorů. Dohromady to znamenalo pro každého anotátora 24 a dohromady 120 anotací. Anotace byly rozděleny tak, aby každá dvojice anotátorů měla právě 6 stejných testovacích sad.

Během anotace se zjistilo, že u dvou testovacích sad se jeden z obrázků nena-

	o_test1	o_test2	celkem
anotátorů	5	5	7
textů	58	120	178
anotací	116	240	356
pozitivní shoda	45/58 = 78%	76/120 = **63**%	121/178 = 67%
negativní shoda	3/58 = 5%	15/120 = 13%	18/178 = 10%
neshoda	10/58 = 17%	29/120 = 24%	39/178 = 22%

Obrázek 12.1: Přehled výsledků uživatelského testování.

čítá. Tyto sady byly z testování vyřazeny a anotovaných testovacích sad je tedy pouze 58. Výsledky testování jsou shrnuty v tabulce 12.1 ve sloupci „o_test1“. Pobrobné výsledky testování, včetně Cohenovy kappy pro všechny dvojice anotátorů je v příloze 13.

Výsledky ukazují, že pro 78 % testovacích sad měli anotátoři pozitivní shodu. Pozitivní shoda znamená, že se oba anotátoři shodli na stejném obrázku, který je pro vstupní text nejméně vhodný. Tento obrázek byl zároveň vybrán do testovací sady náhodně. Čím je toto procento vyšší, tím lépe náš algoritmus pracuje, protože uživatelé jsou schopní detekovat náhodný obrázek. Pro 5 % testovacích sad máme negativní shodu. Oba anotátoři vybrali obrázek, který nebyl přiřazen náhodně. Znamená to tedy, že tento obrázek nebyli schopní detekovat a algoritmus tedy pro vstupní text nepracuje dobře (pokud vyloučíme možnost, že náhodně přiřazený obrázek je k textu relevantní). U poslední skupiny testovacích dat – 17 % – byl pouze jeden z anotátorů schopen vybrat náhodně vybraný obrázek.

Získaná data ukazují, že algoritmus pracuje poměrně dobře. Pokud by algoritmus přiřazoval automaticky obrázky k textům i v praxi, čtenáři by mohli by poznali rozdíl od algoritmu, který přiřazuje obrázky nahodile.

Během testování se objevil jeden zásadní problém s testovací metodou. V datasetu Profimedie jsou i obrázky, které jsou si velmi vizuálně podobné, například fotky stejné osoby z různých úhlů. Tyto fotky mají často i stejné textové popisky. Mějme tedy 4 obrázky, které jsou si vizuálně velmi podobné a mají stejné textové popisky. Pokud algoritmus označí jako nejvhodnější obrázek k textu jeden z těchto obrázků, budou i na dalších třech doporučených pozicích vizuálně podobné obrázky (pokud tedy nemáme jinou množinu obrázků, která má stejné textové popisky, ale je vizuálně odlišná). Pokud se takový text objeví v naší testovací metodě, uvidí anotátor 4 velmi podobné obrázky a k nim jeden náhodně vybraný. Nejméně vhodný obrázek pak snadno označí, aniž by vůbec četl anotační text. Ukázalo se, že v našem testování k takovému problému opravdu došlo – anotátorovi se zobrazily obrázky, z nichž ani jeden nebyl vhodným obrázkem k danému textu, přesto anotátor snadno označil nejméně vhodný obrázek. Tento problém zkresluje dosažené výsledky testování touto metodou.

12.2 Detekce správného obrázku (o_test2)

Abychom předešli problémům s metodou popsaným v předchozí sekci, provedli jsme nové testování. Úkolem uživatelů bylo nyní vybrat obrázek, který se ke vstupnímu textu hodí nejvíce. Množinu obrázků nyní tvořil jeden výstup z algorit-

mu a 4 náhodně vybrané obrázky. Zvětšili jsme i testovací sadu. Bylo testováno 120 textů náhodně vybraných z iDnes datasetu. Každý z textů byl anotován dvěma anotátory. Na každém z pěti anotátorů tedy bylo 48 anotací. Výsledky testování jsou shrnuty v tabulce 12.1 ve sloupci „o_test2“. Pobrobné výsledky testování, včetně Cohenovy kappy pro všechny dvojice anotátorů je v příloze 13.

Výsledky jsou oproti předchozí metodě o něco horší. Pro 63 % testovacích sad měli oba anotátoři pozitivní shodu, pro 13 % měli anotátoři negativní shodu a pro 24 % se anotátoři neshodli.

12.3 Shrnutí

Uživatelské testování ukazuje, že je algoritmus schopen v poměrně velkém procentu případů přiřadit automaticky „dostatečňě vhodný“ ilustrační obrázek. To, že je ilustrační obrázek dostatečně vhodný ovšem neznamená, že je nejvhodnější. Výběr nejvhodnějšího ilustračního obrázku je ovšem velmi individuální a nedá se obecně měřit.

Testování ukázalo také na slabiny algoritmu. Možná největší slabinou je fáze překladu, díky níž popisky k některým obrázkům nesprávné. Tímto problémem netrpí algoritmus pro anglické vyhledávání, který pracuje s nepřeloženými popisky. Ten však nebyl uživatelsky testován.

Jedním z problémů testování je právě omezená doména testovaných textů. Můžeme říci, že algoritmus funguje dobře na nějaké doméně textů, ale pro obecné zpravodajské články může algoritmus fungovat velmi špatně. Toto ovšem není problém testovaného algoritmu, ale dat na kterých pracuje. Rozšíření domény obrázků v korpusu Profimedie by rozšířilo i doménu textů pro které algoritmus funguje dobře.

13. Závěr

Seznam použité literatury

[1] APACHE SOFTWARE FOUNDATION. *Apache Lucene 4.9.0* [software]. 2014. Dostupné z: `http://lucene.apache.org/`.

[2] ATTARDI, Giuseppe. *Wikipedia Extractor* [software]. 2013. Dostupné z: `http://medialab.di.unipi.it/wiki/Wikipedia_Extractor`.

[3] ELASTICSEARCH BV. *elasticsearch-ruby* [software]. 2014. Dostupné z: `https://github.com/elasticsearch/elasticsearch-ruby`.

[4] ELASTICSEARCH BV. *Elasticsearch 1.3.0* [software]. 2014. Dostupné z: `http://www.elasticsearch.org/`.

[5] BOTOREK, Jan. *Tvorba nástroje pro zpracování textových popisů multimediálních dat* [online]. 2012 [cit. 2014-07-17]. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Petra Budíková. Dostupné z: `http://is.muni.cz/th/359815/fi_b/`.

[6] BRAY, Tim, et al. *Extensible markup language (XML).* World Wide Web Consortium Recommendation REC-xml-19980210. http://www.w3.org/TR/1998/REC-xml-19980210, 1998.

[7] BUDÍKOVÁ, Petra, Michal BATKO a Pavel ZEZULA. *Evaluation Platform for Content-based Image Retrieval Systems.* In *International Conference on Theory and Practice of Digital Libraries 2011, LNCS 6966.* Berlin: Springer, 2011. s. 130-142, 12 s. ISBN 978-3-642-24468-1.

[8] CAVNAR, William B., et al. *N-gram-based text categorization. Ann Arbor MI*, 1994, 48113.2: 161-175.

[9] CHANG, Jonathan, et al. *Reading tea leaves: How humans interpret topic models.* In: *Advances in neural information processing systems.* 2009. s. 288-296.

[10] COUCHBASE. *Couchbase 2.5.1* [software]. 2014. Dostupné z: `http://www.couchbase.com/`.

[11] CROCKFORD, Douglas. *The application/json media type for javascript object notation (json).* 2006. Dostupné z: `http://tools.ietf.org/html/rfc4627.txt`.

[12] DEAN, Jeffrey; GHEMAWAT, Sanjay. *MapReduce: simplified data processing on large clusters.* Communications of the ACM, 2008, 51.1: 107-113.

[13] FETTE, Ian; MELNIKOV, Alexey. *The WebSocket Protocol.* 2011. Dostupné z: `http://tools.ietf.org/html/rfc6455`.

[14] FIELDING, Roy. *Representational state transfer.* Architectural Styles and the Design of Netowork-based Software Architecture, 2000. s. 76-85.

[15] GERRARD, Andrew. *Go version 1 is released* [online]. 2012 Dostupné z: `http://blog.golang.org/go-version-1-is-released`.

[16] GOOGLE INC. *The Go Programming Language* [software]. 2014. Dostupné z: http://golang.org/.

[17] GOOGLE INC. *Překladač Google* [online]. 2014 [cit. 2014-07-25]. Dostupné z: http://tranlslate.google.cz.

[18] HARTER, Mark. *Managing Node.js Callback Hell with Promises, Generators and Other Approaches*. 2014. Dostupné z: http://strongloop.com/strongblog/node-js-callback-hell-promises-generators

[19] JIA, Yangqing. *Caffe: An open source convolutional architecture for fast feature embedding*. 2013 Dostupné z: http://caffe.berkeleyvision.org.

[20] KOEHN, Philipp. *Statistical machine translation*. Cambridge University Press, 2009.

[21] KOEHN, Philipp, et al. *Moses: Open source toolkit for statistical machine translation*. In: Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions. Association for Computational Linguistics, 2007. s. 177-180.

[22] LUBBERS, Peter; GRECO, Frank. *HTML5 Web Sockets: A Quantum Leap in Scalability for the Web* [online]. 2013 [cit. 2014-07-25]. Dostupné z: http://www.websocket.org/quantum.html.

[23] LOTT, Brian. *Survey of Keyword Extraction Techniques*. UNM Education, 2012.

[24] MATSUMOTO, Yukihiro, et al. *Ruby Programming Language* [software]. 2014. Dostupné z: https://www.ruby-lang.org/.

[25] MILLER, George A. *WordNet: A Lexical Database for English*. Communications of the ACM, 1995, roč. 38, č. 11, s. 39-41.

[26] PORTER, Martin F. *An algorithm for suffix stripping*. Program: electronic library and information systems. MCB UP Ltd, 1980, roč. 14, č. 3, s. 130-137.

[27] RAILS CORE TEAM. *Ruby on Rails* [software]. 2014. Dostupné z: http://rubyonrails.org/.

[28] STRAUCH, Christof; SITES, Ultra-Large Scale; KRIHA, Walter. NoSQL databases. Lecture Notes, Stuttgart Media University, 2011.

[29] WIKIPEDIA *Geohash — Wikipedia, The Free Encyclopedia* [online]. 2014 [cit. 2014-07-24] Dostupné z: http://en.wikipedia.org/w/index.php?title=Geohash&oldid=609521802.

[30] WIKIPEDIA *Model–view–controller — Wikipedia, The Free Encyclopedia* [online]. 2014 [cit. 2014-07-28] Dostupné z: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93conti

[31] ZAYTSEV, Juriy. *ECMAScript 5 compatibility table* [online]. 2014 [cit. 2014-07-28] Dostupné z: http://kangax.github.io/compat-table/es5/.

Seznam tabulek

Seznam použitých zkratek

Příloha 1

```
Výsledek testu o_test1: 103 / 116 = 88.79 %
Výsledek testu o_test2: 196 / 240 = 81.66 %
Celkové výsledky dohromady: 299 / 356 = 83.98 %


havel
 o_test1: 20 / 22 = 90.90 %
 o_test2: 36 / 48 = 75.0  %
 celkem:  56 / 70 = 80.0  %
  odchazel_o
   o_test1
    3   0
    1   1
    Cohen's kappa = 0.5454545454545455
  odchazel_v
   o_test2
    10  2
    0   0
    Cohen's kappa = 0.0
  paroubkova
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN
   o_test2
    6   1
    3   2
    Cohen's kappa = 0.2727272727272727
  pavlovic
   o_test1
    5   0
    0   0
    Cohen's kappa = NaN
  rakosova
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN
   o_test2
    9   0
    1   2
    Cohen's kappa = 0.7499999999999999
  semerad
   o_test2
```

```
    6   2
    0   4
    Cohen's kappa = 0.6666666666666667

odchazel_o
 o_test1: 21 / 23 = 91.30 %
 celkem:  21 / 23 = 91.30 %
  havel
   o_test1
    3   1
    0   1
    Cohen's kappa = 0.5454545454545455
  paroubkova
   o_test1
    4   1
    0   1
    Cohen's kappa = 0.5714285714285715
  pavlovic
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN
  rakosova
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN

odchazel_v
 o_test2: 42 / 48 = 87.5  %
 celkem:  42 / 48 = 87.5  %
  havel
   o_test2
    10  0
    2   0
    Cohen's kappa = 0.0
  paroubkova
   o_test2
    10  1
    0   1
    Cohen's kappa = 0.625
  rakosova
   o_test2
    9   1
    0   2
    Cohen's kappa = 0.7499999999999999
  semerad
   o_test2
```

```
    11  0
    0   1
    Cohen's kappa = 1.0

paroubkova
 o_test1: 20 / 24 = 83.33 %
 o_test2: 43 / 48 = 89.58 %
 celkem:  63 / 72 = 87.5  %
  havel
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN
   o_test2
    6   3
    1   2
    Cohen's kappa = 0.2727272727272727
  odchazel_o
   o_test1
    4   0
    1   1
    Cohen's kappa = 0.5714285714285715
  odchazel_v
   o_test2
    10  0
    1   1
    Cohen's kappa = 0.625
  pavlovic
   o_test1
    4   0
    2   0
    Cohen's kappa = 0.0
  rakosova
   o_test1
    5   1
    0   0
    Cohen's kappa = 0.0
   o_test2
    11  1
    0   0
    Cohen's kappa = 0.0
  semerad
   o_test2
    10  2
    0   0
    Cohen's kappa = 0.0

pavlovic
```

```
 o_test1: 21 / 23 = 91.30 %
 celkem:  21 / 23 = 91.30 %
  havel
   o_test1
    5   0
    0   0
    Cohen's kappa = NaN
  odchazel_o
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN
  paroubkova
   o_test1
    4   2
    0   0
    Cohen's kappa = 0.0
  rakosova
   o_test1
    3   1
    1   1
    Cohen's kappa = 0.24999999999999986

rakosova
 o_test1: 21 / 24 = 87.5  %
 o_test2: 39 / 48 = 81.25 %
 celkem:  60 / 72 = 83.33 %
  havel
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN
   o_test2
    9   1
    0   2
    Cohen's kappa = 0.7499999999999999
  odchazel_o
   o_test1
    6   0
    0   0
    Cohen's kappa = NaN
  odchazel_v
   o_test2
    9   0
    1   2
    Cohen's kappa = 0.7499999999999999
  paroubkova
   o_test1
```

```
    5   0
    1   0
    Cohen's kappa = 0.0
   o_test2
    11  0
    1   0
    Cohen's kappa = 0.0
  pavlovic
   o_test1
    3   1
    1   1
    Cohen's kappa = 0.24999999999999986
  semerad
   o_test2
    9   0
    0   3
    Cohen's kappa = 1.0

semerad
 o_test2: 36 / 48 = 75.0  %
 celkem:  36 / 48 = 75.0  %
  havel
   o_test2
    6   0
    2   4
    Cohen's kappa = 0.6666666666666667
  odchazel_v
   o_test2
    11  0
    0   1
    Cohen's kappa = 1.0
  paroubkova
   o_test2
    10  0
    2   0
    Cohen's kappa = 0.0
  rakosova
   o_test2
    9   0
    0   3
    Cohen's kappa = 1.0


o_test1
 nesprávné označení (text, uživatel)
  00207 o_paroubkova
  01337 o_pavlovic
  01595 o_paroubkova
```

```
  01921 o_rakosova
  03758 o_odchazel_ondrej
  03758 o_paroubkova
  03907 o_paroubkova
  04107 o_rakosova
  07604 o_havel
  07604 o_odchazel_ondrej
  09093 o_pavlovic
  09093 o_rakosova
  09551 o_havel
celkem 116 anotací 58 textů
každý text oanotován dvěma anotátory
 oba správně : 45 / 58 = 77.58 %
 oba špatně  : 3 / 58 = 5.172 %
 jeden špatně: 10 / 58 = 17.24 %

o_test2
 nesprávné označení (text, uživatel)
  00124 o_semerad
  00405 o_havel
  00405 o_paroubkova
  00623 o_paroubkova
  01907 o_rakosova
  01911 o_odchazel_vojtech
  01911 o_paroubkova
  02505 o_odchazel_vojtech
  02505 o_rakosova
  03610 o_havel
  03610 o_semerad
  04211 o_odchazel_vojtech
  04282 o_havel
  04282 o_semerad
  04368 o_odchazel_vojtech
  04368 o_semerad
  04480 o_semerad
  04530 o_havel
  05102 o_rakosova
  05102 o_semerad
  05692 o_semerad
  06379 o_paroubkova
  06777 o_rakosova
  07326 o_rakosova
  07326 o_semerad
  07587 o_odchazel_vojtech
  07587 o_rakosova
  08015 o_havel
  08015 o_semerad
  08621 o_havel
```

```
  12931 o_odchazel_vojtech
  13674 o_havel
  13674 o_semerad
  13803 o_havel
  13803 o_paroubkova
  14257 o_havel
  14257 o_rakosova
  14329 o_rakosova
  14329 o_semerad
  14353 o_havel
  14353 o_rakosova
  14988 o_semerad
  15726 o_havel
  15770 o_havel
celkem 240 anotací 120 textů
každý text oanotován dvěma anotátory
 oba správně : 76 / 120 = 63.33 %
 oba špatně  : 15 / 120 = 12.5  %
 jeden špatně: 29 / 120 = 24.16 %
```

Příloha 2

Tato příloha obsahuje informace o datech na přiloženém DVD.