FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Dvojjazyčné vyhledávání v dokumentech
Student:	Lukáš Rynt
Vedoucí:	Ing. David Bernhauer
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Fulltextové vyhledávání v dokumentech naráží na problémy s použitím synonym nebo s použitím různých jazyků.
Popište základní algoritmy pro indexaci a vyhledávání v textových dokumentech (minimálně booleovský a vektorový model).
Popište princip word embedding algoritmů obecně a dále se zaměřte na Word2Vec.
Proveďte rešerši některých přístupů pro vícejazyčné použití Word2Vec.
Využijte vícejazyčného znění zákonů (např. nový občanský zákoník, právní akty EU, ...) pro natrénování dvojjazyčného Word2Vec (čeština a angličtina).
Navrhněte a implementujte prototyp jednoduchého webového vyhledávače v dokumentech na základě natrénovaného dvojjazyčného Word2Vec modelu.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 19. listopadu 2021 v Praze.

Bakalářská práce

DVOJJAZYČNÉ VYHLEDÁVÁNÍ V DOKUMENTECH

Lukáš Rynt

Fakulta informačních technologií
Katedra teoretické informatiky
Vedoucí: Ing. David Bernhauer
11. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Chtěl bych poděkovat především svému vedoucímu Ing. Davidu Bernhauerovi za jeho čas, trpělivost, rady a ochotu kdykoliv pomáhat s prací. Dále patří díky mé rodině, spolubydlícím a přátelům za poskytnutí prostředí a podpory, díky které jsem mohl práci zdárně dokončit.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

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

V Praze dne 11. května 2022 .....

Abstrakt

Tato práce se zabývá výzkumem modelů získávání informací, nejmodernějších technik vnoření slov (word embedding) a jejich možným využitím pro vícejazyčné vyhledávání. Moderní přístupy k vícejazyčnému vyhledávání, které staví na word embedding technikách, většinou pracují s určitou transformací, která převádí reprezentace slov z jednoho jazyka do druhého. Cílem této práce je zkoumat model, který s touto transformací nepracuje a namísto toho rovnou získává závislosti mezi překlady. Nad tímto modelem by pak měl být vystavěn prototyp webového vyhledávače.

Práce v tomto dostála všem očekáváním a výsledný model byl schopný reprezentovat dvojjazyčné překlady napřímo bez použití jakékoliv transformace. Toho bylo dosaženo s využitím paralelně přeložených dokumentů Evropské Unie, které byly pro oba jazyky spojeny na úrovni odstavců. Prototyp vyhledávače poté fungoval na základě naučených reprezentací jednotlivých slov, vyvozených z tohoto modelu.

Klíčová slova získávání informací, přirozené zpracování textu, vnoření slov, Word2Vec, vícejazyčné vyhledávání, vektorový model

Abstract

This thesis is concerned with the research of information retrieval models, state-of-the-art word embedding techniques and their possible use for multilingual retrieval. Modern approaches to multilingual retrieval that build on word embedding techniques usually work with some transformation that converts word representations from one language to another. The aim of this thesis is to investigate a model that does not work with this transformation and instead directly extracts dependencies between translations. A prototype web search engine should then be built on top of this model.

The work has met all expectations and the resulting model was able to represent bilingual translations directly without using any transformation. This was achieved using parallel translated European Union documents, which were linked at paragraph level for both languages. The prototype search engine then operated based on the learned word representations derived from this model.

Keywords information retrieval, natural language processing, word embedding, Word2Vec, mutilingual information retrieval, vector space model

Seznam zkratek

CBOW	Continuous Bag-of-Wsords
CLIR	Cross-Language Information Retrieval
DF	Document frequency
HTML	Hypertext Markup Language
IDF	Inverted document frequency
IR	Information Retrieval
JSON	JavaScript Object Notation
LSI	Latent Semantic Indexing
LSTM	Long short-term memory
MUSE	Multilingual Unsupervised and Supervised Embeddings
MVC	Model-View-Controller
NLP	Natural Language Processing
NN	Neural Network
NNLM	Feedforward Neural Net Language Model
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
t-SNE	t-distributed Stochastic Neighbor Embedding
TF	Term frequency
ÚFAL	Ústav formální a aplikované lingvistiky
XML	Extensible Markup Language

Kapitola 1
Úvod

Klasické modely vyhledávání ve fulltextových dokumentech naráží na problém s hledáním synonym, případně na problém vyhledávání ve více jazycích. S růstem webových technologií a obecnou globalizací je toto přitom poměrně důležitý aspekt toho, co bychom měli od takovýchto modelů očekávat. Vzhledem ke stále narůstajícímu objemu informací na internetu, kde široce využívané jazyky jako angličtina ztrácí své prominentní postavení, je potřeba brát v potaz nutnost vyhledávat i ve zdrojích psaných v jiných jazycích.

Příkladů využití vícejazyčného vyhledávání v dokumentech je mnoho, jak v soukromé, tak ve veřejné sféře. Mohou ho využívat business analytici, hledající aktuální informace z různých zemí, žurnalisté píšící o aktuálním dění v zahraničí nebo vědci a akademici zkoumající problém, který už byl řešen, ovšem v jiném jazyce. Vícejazyčné vyhledávání nám dává možnost rychle a efektivně řešit rozdrobenost a neprovázanost dokumentů psaných v různých jazycích.

Tato práce se zabývá metodami zpracování informací (anglicky *information retrieval*). Konkrétně se zaobírá problematikou vyhledávání v kolekcích dokumentů – od samotných modelů, které se tímto zabývají, po indexaci dat a vyhodnocování úspěšnosti těchto modelů.

Dále pak zkoumá techniky zpracování přirozeného jazyka. Zejména pak *word embeddingy*, zabývající se kódováním slov do vektorové reprezentace, která je výhodnější pro strojové zpracování. Tyto techniky jsou schopné poradit si se zachycením asociací mezi slovy a zároveň jsou schopné reprezentovat vazby mezi přeloženými slovy. Jmenovitě je kladen důraz na model Word2Vec, jeho možnosti a využití pro vícejazyčné překlady.

Cílem této práce je spojení *information retrieval* modelů a jejich indexačních schopností s modelem Word2Vec, uzpůsobeným na dva jazyky, který by měl být schopný zachytit vazby mezi přeloženými slovy, které by klasické modely nebyly schopny postihnout. Model vzniklý tímto spojením by pak byl schopen pracovat jako vícejazyčný fulltextový vyhledávač. Pro učení takového modelu můžeme s výhodou využít plně a explicitně přeložené texty, jako jsou například právní nařízení a zákony Evropské Unie.

1.1 Cíle práce

Cílem práce v teoretické rovině je rešerše existujících postupů zabývajících se indexací a vyhledáváním v textových dokumentech, word embedding algoritmů s důrazem na model Word2Vec a dále prozkoumání různých technik vícejazyčného vyhledávání. Hlavním cílem práce je poté na základě těchto nabytých znalostí natrénovat Word2Vec model na dvou jazycích a nad tímto modelem vystavět prototyp jednoduchého dvojjazyčného webového vyhledávače.

1.2 Struktura práce

Tato práce je členěna na kapitoly zabývající se nejprve rešerší vybraných modelů a principů důležitých k pochopení dané problematiky, v další části jsou pak popisovány postupy využité ke konstrukci vícejazyčného vyhledávacího modelu.

Ve druhé kapitole je rozebíráno přirozené zpracování jazyka, což je odvětví strojového učení, které se zaměřuje na zpracování přirozené lidské mluvy a psaného projevu. V této kapitole jsou rozebrány základní techniky, které jsou při této práci s jazykem používány – jmenovitě pak předzpracování textu.

Cílem třetí kapitoly je seznámit čtenáře se základy *information retrieval* (tedy získávání informací) a představit nejznámější modely, které jsou k těmto účelům využívané. Zejména se pak tato kapitola zaměřuje na modely jako jsou booleovský model, rozšířený booleovský model a vektorový model.

Čtvrtá kapitola se zabývá předvedením pojmu *word embedding*, představení nejznámějších modelů a podrobnější popis některých vybraných. Hlavním záměrem této práce je výzkum modelu Word2Vec, a proto je zde na něj kladen největší důraz. Vzhledem ke své značné podobnosti je zde pro uvedení do problematiky představen také model NNLM. V poslední části kapitoly je předvedena alternativa zvaná GloVe.

Poslední kapitolou rešeršní části práce je výzkum vícejazyčných modelů postavených na word embedding technikách, které jsou představeny v předchozí kapitole. Tyto modely umožňují pracovat s více jazyky a blíže prozkoumávat vztahy mezi nimi. Na výzkumu těchto modelů poté stojí samotný cíl praktické části práce. Konkrétně jsou zde představeny modely jako je CL-LSI, Duet a MUSE.

Šestá kapitola je první kapitolou praktické části práce, která představuje problematiku vytvoření vstupních dat pro výsledný vícejazyčný model. Jsou zde vytvořeny dvě kolekce, přeložené do angličtiny a češtiny. V této kapitole jsou využity jako zdroje dat portál Evropské Unie obsahující právní nařízení a vyhlášky, a také kolekce přeložených abstraktů z akademických prací.

V sedmé kapitole je řešena problematika vytváření vektorových reprezentací na základě vytvořených kolekcí. Je zde také ukázáno, jak jednotlivé vektory reprezentují vztahy mezi přeloženými slovy na základě vizualizací a zkonstruovaných metrik.

Obsahem poslední kapitoly je popis implementace prototypu webového vyhledávače, který staví na těchto naučených vektorových reprezentacích. Jsou zde popsány postupy vedoucí k vytvoření tohoto programu a popis jeho výstupů.

Část I

Teoretická část

Kapitola 2

Přirozené zpracování jazyka

Přirozeným jazykem chápeme mluvenou a psanou formu, která přirozeně vznikla mezi lidmi za účelem komunikace. V technikách zpracování přirozeného jazyka se snažíme takto člověkem vytvořená data analyzovat a přiřadit jim strojově pochopitelný význam. Výsledkem je tedy stroj, který rozumí přirozenému jazyku a dokáže reagovat na základě příchozích dat v nestrukturovaném formátu. Účelem této kapitoly je představit základní koncepty a techniky používané při práci s přirozeným jazykem.

Přirozené zpracování jazyka (*natural language processing*) je odvětví umělé inteligence, které vytváří komunikační rozhraní mezi stroji a lidmi. Na rozdíl od programování, kdy se lidé snaží přizpůsobit počítači, se v tomto případě stroje přizpůsobují přirozenému jazyku lidí. Další důležitou součástí NLP je transformace textové reprezentace do číselné, která je lépe srozumitelná počítači. Tímto problémem se zabývají *word embedding* techniky probírané v kapitole 4. Díky tomu jsou poté stroje schopné řešit problémy zadané v přirozeném jazyce, případně reagovat na dotazy v něm vytvořené. [1]

2.1 Praktické aplikace

Praktických využití přirozeného zpracování jazyka v reálném světě je mnoho. Mezi hlavní kategorie patří rozpoznávání mluvené řeči, porozumění přirozenému jazyku a samotné generování přirozeného jazyka – tedy schopnost stroje vytvořit smysluplné odpovědi v přirozeném jazyce na základě daného dotazu. Pro představu je zde uvedeno několik příkladů, kde se NLP vyskytuje: [1]

- *automatické shrnutí textů*
- *chatboti* – umělá inteligence, která je schopná v určité oblasti napodobovat rozhovor s reálným člověkem
- *analýza sentimentu* – rozpoznávání emocí nebo názoru z textu
- *překladače*
- *osobní hlasoví asistenti* – jako Siri nebo Alexa
- *automatická oprava gramatických chyb*

2.2 Předzpracování textu

Většina úloh strojového zpracování vyžaduje očištěná a předzpracovaná data. V našem případě, kdy pracujeme s přirozeným jazykem, je tato potřeba ještě důležitější s ohledem na neurčitost a absenci jakékoliv struktury, což jsou vlastnosti, které jsou pro přirozený jazyk velmi charakteristické. Techniky probírané v této kapitole jsou potřebné jak pro účely NLP, tak pro *information retrieval* (získávání informací), kterým se zaobírají pozdější kapitoly.

Předzpracování samotné zahrnuje převedení přirozeného textu do takové reprezentace, která je srozumitelná počítači. Za tímto účelem se používají techniky, které se snaží provádět různé morfologické transformace pro snazší zpracování nebo rozdělují text na menší logické celky.

V průběhu zpracování mohou vyvstat různé lingvistické problémy, které se váží ke konkrétnímu jazyku. Jedná se především o morfologii (tvarosloví) slova, použití synonym a homonym nebo použití abstraktních možností jazyka, jako jsou například metafory. Dále také používání zájmen, metafor a jiných „nejasností“ může znesnadnit interpretaci textu. Všechny tyto zmíněné fenomény negativně ovlivňují kvalitu zpracování přirozeného textu.

Techniky, které jsou uvedeny níže, se snaží vyřešit zejména strukturní problémy – tedy především samotnou morfologii daných slov. Většina z nich je silně závislá na jazyku, který zrovna používáme a obecně nejsou známy takto jednoduché techniky, které by bylo možné aplikovat na libovolný jazyk vždy stejně. Problémem synonym a homonym se budeme zaobírat v dalších kapitolách, vzhledem k tomu, že je zde potřeba pracovat s více informacemi (jmenovitě s kontextem, ve kterém se slova vyskytují).

Kromě lingvistických problémů také můžeme narazit na špatné formátování textu, případně výskyt různého šumu. Text může obsahovat irelevantní symboly jako jsou například XML nebo HTML značky, případně různé typy seznamů, tabulek a číslování. S těmito problémy je potřeba si poradit ještě před použitím technik popisovaných níže. [2, 1]

2.2.1 Tokenizace

Tokenizací rozumíme techniku ve které se snažíme rozdělit sekvenci znaků na jednotlivé části – takzvané *tokeny*. Způsoby, jakými můžeme takovou sekvenci rozdělit je více, nejčastěji se však spokojíme s rozdělením po slovech. Z toho důvodu se často pojmy *token* a *slovo* zaměňují. Nejčastěji se k tokenizaci po slovech využívají mezery nebo jiné bílé znaky. V některých případech toto ovšem není dostačující – vezměme si například anglické slovo *„aren't“* – to můžeme rozdělit jako `are` `n't` nebo ho nechat v celku: `aren't`. V tomto konkrétním případě záleží na kontextu, kde danou problematiku řešíme. Můžeme si ale všimnout, že výběr techniky, kterou používáme, může místy měnit i slova, která jsou obsažená ve finální kolekci tokenů. [2]

Problémy nekončí anglickými apostrofy. Některé znakové jazyky jako čínština přiřazují význam i jedinému znaku, ale není to u nich pravidlem. V takovém případě přichází na scénu tokenizace po jednotlivých znacích, případně dvojicích znaků (*bigramů*). Obecněji můžeme používat také *n-gramy*, kdy skupině n znaků přiřazujeme vlastní význam. Mimo tyto klasické možnosti jsou přípustné například tokenizace po větách nebo tokenizace s použitím speciálních regulárních výrazů. [3, 1]

Pro účely následujících technik předpokládejme využití klasické tokenizace po slovech – můžeme tedy volně zaměňovat pojmy *slovo* a *token*.

2.2.2 Stemming

Stemming i *lemmatizace* se zabývají redukcí slova na jeho kořen nebo určitou základní formu. V případě *stemmingu* není úplně zaručené, že to, co získáme bude kořen slova, ale pouze určitá kanonická forma. Jedná se o techniku, která na základě předem daných pravidel odstraňuje části slov, jako jsou přípony a předpony. Takto postupuje bez ohledu na gramatickou správnost

a jazykovou smysluplnost výsledku. Využívá se zejména při vyhledávání, kdy chceme porovnávat dotaz se slovy, která můžou mít různý tvar – většinou nás totiž ve výsledku nezajímá absolutně přesná shoda. Zároveň díky těmto technikám docílíme redukce počtu využívaných tokenů, což poté v praxi znamená poměrně velký rozdíl ve využití paměti a rychlosti zpracování. [2, 1]

Nejčastěji používaným algoritmem pro angličtinu je **Porterův algoritmus**[1] z roku 1980. Tento algoritmus postupně redukuje slovo v několika fázích, kde v každé fázi vybírá redukci na základě předepsaných pravidel. Dalšími příklady známých stemming algoritmů jsou *Lovinsův* nebo *Paiceův*.[2]

Například pro slovo *„opposing“* by tento algoritmus vytvořil kořen *„oppos“*, což není úplně přesný kořen slova, ale pro pozdější porovnávání s dotazem je to vhodný tvar. V některých případech může být ale toto chování nechtěné. Vezměme si například slova *„machinations“* a *„machine“*, které by tento algoritmus zredukoval na stejný kořen *„machin“*[2]. V případě pozdějšího porovnávání s dotazem by toto vedlo k nechtěným shodám, vzhledem k odlišným významům obou slov (machinace vs. stroj). [3]

2.2.3 Lemmatizace

Na rozdíl od *stemming* algoritmů je **lemmatizace** proces ve kterém provádíme celkovou morfologickou analýzu slova. Výsledkem je pak *lemma*, což je tvar slova, který koresponduje se slovníkovým záznamem – tedy gramaticky korektní, základní forma bez skloňování nebo časování. Například lemmatizace slova *„better“* by vrátila výsledek *„good“*. [1]

Většinou tyto algoritmy využívají rozsáhlých lexikálních databází, které obsahují informace o morfologii slov. Pro angličtinu je nejpoužívanější WordNet databáze vyvíjená na Princetonské univerzitě[3]. Výhodou je vyšší kvalita výsledných lemmat, která může mít za následek přesnější shody při vyhledávání. Na druhou stranu jsou tyto techniky poměrně komplexní a časově náročnější. Z tohoto důvodu se příliš nedoporučuje je využívat při práci s větší kolekcí dokumentů. [2, 1]

2.2.4 Normalizace textu

Normalizace textu je poměrně široký pojem zahrnující více technik, které se zabývají převedením slov do jednotné standardní podoby. Chceme, aby jednotlivá slova byla jednotná předtím, než nad nimi budeme provádět další operace. V tomto případě velmi záleží na jazyce a kolekci dokumentů se kterou pracujeme. Typickým příkladem je namapování zkratek na jednotnou nezkrácenou formu nebo přepis nesprávně psaných slov na jednotný tvar. V některých případech také hraje roli spojovník a mapování výrazů, které ho využívají. Ty chceme převést buď na jednotlivé části nebo na slova bez něj – jako příklad můžeme uvést značku *Hewlett-Packard*, kterou bychom pravděpodobně chtěli namapovat na dvě slova. [2, 1]

Dalším příkladem normalizace je práce s diakritikou, která v některých slovech nepřidává valný význam a můžeme ji odstranit – například *„naive“* vs. *„naïve“*. Podobně můžeme chtít u akronymů vynechat tečky, tak abychom dostali jednotnou verzi napříč dokumenty. [2]

Obdobně můžeme zanedbat velká nebo malá písmena. K této problematice existují dva přístupy – *case-folding* a *truecasing*. Zatímco *case-folding* redukuje všechna velká písmena na malá, *truecasing* se snaží predikovat velikost písmen pomocí předtrénovaného modelu. Většinou si ale vystačíme právě s první jednodušší variantou, vzhledem k tomu, že uživateli většinou nezáleží na velikostech. [2]

[1] Oficiální verze tohoto algoritmu je dostupná na stránce `https://tartarus.org/martin/PorterStemmer/`.
[2] Tento příklad byl zkonstruován s použitím *PorterStemmer* algoritmy z NLTK knihovny v jazyce Python.
[3] Viz `https://wordnet.princeton.edu/`.

2.2.5 Odstranění stop slov

Zejména při práci s *information retrieval* modely nám v textu začnou překážet takzvaná **stop slova** – slova, která sama o sobě nenesou prakticky žádný význam, ale v textu jsou extrémně častá. Typickým příkladem pro angličtinu jsou slova jako *„the“* nebo *„by“*, případně různé spojky. [2, 1]

K odstranění *stop slov* se používají *stop seznamy*, které obsahují nejčastější slova v daném jazyce. Tyto seznamy mohou být buď ručně konstruované nebo mohou vzniknout na základě statistiky v kolekci dokumentů. Slova, která jsou nejčastější poté budeme považovat za *stop slova*. Využití těchto seznamů výrazně redukuje například počet indexovaných slov. [2]

Některé speciální sekvence mohou tímto přístupem trpět – vezměme si například jména anglických písniček, jako je *Let It Be*. S použitím stop seznamu bychom v podstatě zanedbali celý vstupní dotaz. Původní přístup používat rozsáhlé seznamy se slovy čítajícími stovky záznamů, se v poslední době přesunul k využití pouze malých seznamů (kolem 7-12) nebo k variantám, kdy žádné seznamy nepoužíváme (to se v praxi děje například u webových vyhledávačů). [2]

Kapitola 3
Information Retrieval

„*Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).*“ [2]. Zjednodušeně řečeno se tato oblast zabývá hledáním informací v nestrukturovaných datech. S rozmachem webových technologií a velkým nárůstem nestrukturovaných dat z různých zdrojů se **Information Retrieval** stává dominantní formou přístupu k informacím. Postupně převládá i nad databázovým přístupem, který je například pro potřeby webu – kde jsou informace z různých zdrojů – neudržitelný vzhledem k rozličné struktuře a formátům dat. [4]

Nejběžnější aplikací jsou webové vyhledávače, které jsou s každodenním růstem stránek na webu stále více potřebné. V této doméně je poměrně důležité vyvíjet rychlé a přesné výsledky jako odezvu na krátké dotazy poskytnuté uživatelem (většinou se jedná jen o sekvence několika málo slov). Důležitým aspektem těchto systémů je mimo rychlost také řazení výsledků vyhledávání na základě určité relevance. Web samozřejmě není jedinou doménou IR systémů, jejich použití můžeme zaznamenat například v digitálních knihovnách nebo vyhledávacích systémech pro větší firmy. [3]

3.1 Základní pojmy a koncepty

Při práci s informačními systémy má jejich uživatel určitou *informační potřebu*, kterou potřebuje naplnit – tato potřeba řídí proces vyhledávání. Na jejím základě posílá do systému *dotazy*, což jsou krátké sekvence slov – většinou se jedná o klíčová slova nebo fráze. Samotný proces vyhledávání poté probíhá formou dialogu. Uživatel většinou předem nemá jasné povědomí o tom, co přesně chce hledat a pomocí dotazů si upřesňuje svou informační potřebu. Toto je také výrazný rozdíl oproti databázovému přístupu, kde máme pevně dané potřeby a jasné dotazy. Už z podstaty této nejasnosti jsou dotazy často velmi vágní a v *Information Retrieval* systémech musíme být schopni s touto neurčitostí pracovat – hledání tedy v podstatě neznamená spojení dokumentů a dotazů na základě přesné shody, ale uspokojení informační potřeby. [5, 3]

V kontextu IR systémů chápeme **dokument** jako jednotku, nad kterou provádíme vyhledávání – může se jednat například o kapitolu knihy nebo jednotlivé články v časopisu. Kolekci dokumentů pak označujeme jako **corpus** $\mathcal{D}$, kde j-tý dokument corpu označujeme d_j. [2]

Proces vyhledávání můžeme pro IR systémy rozdělit na dvě jasně oddělené fáze, které probíhají každá v jiný okamžik.

Indexační fáze Tato fáze probíhá většinou jednou při konstrukci kolekce nebo při její aktualizaci. Z toho důvodu se také označuje jako *offline* fáze. Během ní se vytváří **index**, což je určitá

struktura, která shromažďuje informace o dokumentech a slovech, které jsou v nich obsaženy. Slova jsou zde označována jako **termy** a většinou se jedná o již předzpracované *tokeny* (viz kapitola 2). Pro potřeby této kapitoly zavádíme slovník $\mathcal{T}$ obsahující všechny indexované termy – i-tý term je pak označován jako t_i. Výstupem indexační fáze je poté index obsahující jednotlivé termy namapované na jejich výskyty v dokumentech. [5, 2]

Vyhledávací fáze Oproti předchozí fázi se fáze vyhledávání (porovnávání) provádí při každém zadání nového dotazu – z toho důvodu se označuje jako *online*. Pracuje se zde s již vytvořeným indexem, se kterým jsou porovnávány dotazy. Dominující přístupy v této oblastí berou termy jako neseřazenou strukturu, označovanou jako *„bag of words“*. Tímto přístupem ztrácíme jakékoliv informace o pořadí a struktuře termů, posléze slov ve větách. Systémy tedy poté považují dokumenty obsahující stejná slova jako velmi podobné – nezáleží na vnitřním uspořádání. Tato informace je ovšem poměrně důležitá a z tohoto důvodu v této kapitole předvedené modely naráží na různá úskalí. [5, 2]

3.1.1 Hodnocení efektivity systému

Hlavním cílem IR systému by měla být maximalizace pravděpodobnosti, že dotaz vrátí relevantní dokumenty. Zároveň chceme nejlepší možné řazení takových dokumentů na základě spočítané pravděpodobnosti relevance. Systém musí být schopen vyhodnocovat i neúplné shody s termy – v těchto případech chceme vracet i dokumenty, které obsahují částečné shody. Z podstaty nejasnosti dotazu nemůžeme nikdy garantovat, že systém bude vždy vracet pouze relevantní dokumenty jako odpověď na dotaz. Z tohoto důvodu má smysl vyhodnocovat efektivitu systému. Tu můžeme chápat například jako míru spokojenosti uživatele se systémem. Pro účely této práce ale budeme uvažovat exaktnější míry předvedené dále. [5]

3.1.1.1 Precision a recall

Dvě klíčové charakteristiky používané pro měření efektivity jsou **precision** a **recall**. Tyto metriky jsou založené na předpokladu, že uživatel chce získat co možná nejvíce relevantních dokumentů a zároveň co nejméně irelevantních dokumentů. [5, 3]

Precision vyjadřuje pravděpodobnost, že dokument, který je ve výsledku, je relevantní. Když označíme množinu vrácenou jako výsledek dotazu *Res* a množinu relevantních dokumentů vyhovující informační potřebě jako *Rel* můžeme *precision* vyjádřit následující rovnicí. [5, 2, 3]

$$\mathrm{P} = \frac{|Rel \cap Res|}{|Res|} \tag{3.1}$$

Druhá klíčová míra je **recall**, která vyjadřuje pravděpodobnost, že relevantní dokument se objevil i ve výsledku. Při stejném značení jako v minulém případě můžeme *recall* vyjádřit následovně. [5, 3]

$$\mathrm{R} = \frac{|Rel \cap Res|}{|Rel|} \tag{3.2}$$

Často je potřeba optimalizovat výsledek s ohledem na obě tyto míry, ale ty jsou v praxi dost často protichůdné. Optimalizací pro *recall* dostáváme zbytečně dokumenty navíc, které nás tolik nezajímají, protože se snažíme obsáhnout co nejvíce relevantních dokumentů. Oproti tomu při snaze optimalizovat systém pro míru *precision* se dostáváme do situace, kdy porovnáváme dotazy s dokumenty velmi striktně a riskujeme, že nedostaneme některé relevantní dokumenty. [5]

3.1.1.2 F-skóre

Občas se používá také kombinace výše uvedených – takzvané **F-skóre**. Nejjednodušší taková míra označovaná jako F_1 lze vypočítat následujícím způsobem. [3]

$$\mathrm{F}_1 = \frac{2}{\frac{1}{\mathrm{R}} + \frac{1}{\mathrm{P}}} \tag{3.3}$$

Jedná se o takzvaný harmonický průměr, jehož vlastností je vyvážení obou částí výsledku – v případě, že se jedna z částí příliš blíží k nule, tato míra bude také velmi blízká nule. [3]

Alternativně můžeme použít také parametrizované *F-skóre* s parametrem β.

$$\mathrm{F}_\beta = \frac{(\beta^2 + 1) \cdot \mathrm{R} \cdot \mathrm{P}}{\beta^2 \cdot \mathrm{R} + \mathrm{P}} \tag{3.4}$$

V tomto případě při volbě parametru $\beta = 0$ dostáváme samotný *recall* a při $\beta = \infty$ dostáváme *precision*. Výhodou je, že volbou parametru zde můžeme dávat důraz na jednu z dílčích měr. [3]

3.1.1.3 Seřazené výsledky

V případě, že uživatele zajímá pouze prvních k dokumentů, můžeme využít míry pro seřazené dokumenty – takzvané *ranked retrieval* míry. Pro tyto potřeby si označme prvních k dokumentů vrácených z výsledku dotazu *Res* jako $Res[1..k]$. Poté můžeme definovat míry pro relevance prvních k dokumentů pro *recall* a *precision* pomocí následujících. [3]

$$\text{recall@}k = \frac{|Rel \cap Res[1..k]|}{|Rel|} \tag{3.5}$$

$$\text{precision@}k = \frac{|Rel \cap Res[1..k]|}{|Res[1..k]|} \tag{3.6}$$

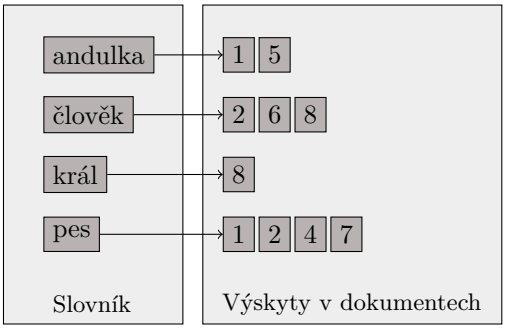
Pro precision@k se také často používá značení P@k. Z podstaty věci recall@k monotónně roste s ohledem na k, zatímco P@k má tendenci s rostoucím k klesat (tedy alespoň za předpokladu, že dokumenty jsou sestupně seřazené na základě jejich relevance vůči dotazu). [3]

3.1.2 Invertovaný index

Jak už bylo zmíněno výše, důležitou součástí IR systému je index. Tato struktura musí určitým způsobem zachycovat vazby mezi termy a jejich výskyty v dokumentech. V případě, že bychom naivně brali každý term a uchovávali si u každého dokumentu příznak, jestli se v něm term vyskytuje nebo ne, dostali bychom v případě větších datasetů velmi rozsáhlou strukturu, v níž by vyhledávání určitě nemohlo probíhat uspokojivě rychle. V podstatě bychom pak získali matici o rozměrech $M = |\mathcal{T}| \times |\mathcal{D}| = N$ – takzvanou *term-document matici*. Její struktura je vyjádřená rovnicí níže. Každá hodnota v ní je binární, pokud se term t_i vyskytuje v dokumentu d_j, na pozici $\mathbf{TDM}_{i,j}$ bude hodnota 1, jinak 0. [2]

$$\mathbf{TDM} = \begin{pmatrix} \boldsymbol{d_1} \\ \vdots \\ \boldsymbol{d_N} \end{pmatrix} = \begin{pmatrix} d_{1;1} & d_{1;2} & \dots & d_{1;M} \\ \vdots & \vdots & \ddots & \vdots \\ d_{N;1} & d_{N;2} & \dots & d_{N;M} \end{pmatrix} \tag{3.7}$$

Takto vytvořena matice bude ve výsledku poměrně řídká – tedy na většině indexů bude obsahovat 0, až na pár vybraných kombinací indexů dokumentů a slov. Z tohoto důvodu se zavádí takzvaný **invertovaný index**, který v sobě obsahuje dvě části – slovník termů a jejich výskyty v dokumentech. Obrázek 3.1 jednoduše vystihuje strukturu invertovaného indexu. Pro potřeby této struktury přiřazujeme každému dokumentu unikátní číslo, díky kterému se na něj



Obrázek 3.1 Příklad invertovaného indexu

můžeme později odkazovat. V tomto případě by například dokument označený číslem 1 obsahoval termy *„andulka“* a *„pes“*.

Tato struktura se většinou ukládá v paměti a jednotlivé termy jsou abecedně seřazeny z důvodu rychlejšího hledání v ní. Při tvorbě jsou termy, které se v kolekci vyskytují duplicitně, sjednoceny a namapovány na dokumenty, ve kterých se vyskytují. [5, 2, 3]

3.2 Booleovský model

Booleovský model je jedním z prvních a nejjednodušších vyhledávacích IR modelů. Využívá exaktní shody mezi dotazy a dokumenty, kde každý dotaz je tvořen indexovanými termy a logickými operátory AND, OR a NOT. Například při použití operátoru AND pak musí být při dotazu „x AND y“ obsaženy v dokumentu oba termy x a y. Naproti tomu při použití operátoru OR stačí aby byl v dokumentu obsažen pouze jeden z nich. [4]

Booleovský model je z podstaty svého fungování binární – dokument buď potřeby dotazu splňuje nebo ne. Nemáme zde žádný způsob, jak ohodnotit relevanci jednotlivých dokumentů, což pro reálné použití není příliš vhodné. Z tohoto důvodu budou výsledné kolekce vždy různě velké. V případě, kdy se dotaz neshoduje s žádným dokumentem z kolekce nedostaneme žádné výsledky, zatímco v případě použití častého slova dostaneme kolekci téměř celou. [4, 6]

Na druhou stranu je jeho implementace v zásadě velmi jednoduchá a dotazy jsou jasně formulované. Všechny dokumenty a termy se v tomto případě považují za stejně důležité a neexistuje zde způsob, jak jim přiřadit určitou váhu. Zároveň z pohledu uživatele nejsou dotazy v booleovské formě příliš přívětivé. [6]

3.3 Rozšířený booleovský model

Tento problém řeší **rozšířený booleovský model**, který přidává váhy jednotlivým dokumentům – ty vyjadřují relevanci vůči dotazu. Konkrétně pro dokument d_j a term t_i dostáváme váhu $w_{i,j}$[1] vyjádřenou rovnicí 3.8. Zde $\mathrm{f}(t_i, d_j)$ vyjadřuje frekvenci termu t_i v dokumentu d_j a $\mathrm{idf}(t_i)$ vyjadřuje *inverted document frequency*, která je většinou předpočítaná v průběhu indexace a detailněji bude popsána dále – zjednodušeně jde ale o globální frekvenci termu napříč celým corpem. První část tedy vyjadřuje normovanou frekvenci termu v daném dokumentu a druhá normovanou globální frekvenci termu v kolekci. Z důvodu této normalizace budou váhy termů vždy v intervalu mezi 0 a 1. [6]

$$\mathrm{w}(t_i, d_j) = w_{i,j} = \frac{\mathrm{f}(t_i, d_j)}{\max\limits_k \mathrm{f}(t_i, d_k)} \cdot \frac{\mathrm{idf}(t_i)}{\max\limits_k \mathrm{idf}(t_k)} \tag{3.8}$$

[1] Značení $w_{i,j}$ zde považujeme jako zkrácený zápis pro hodnotu vzniklou použitím váhovací funkce $\mathrm{w}(t_i, d_j)$.

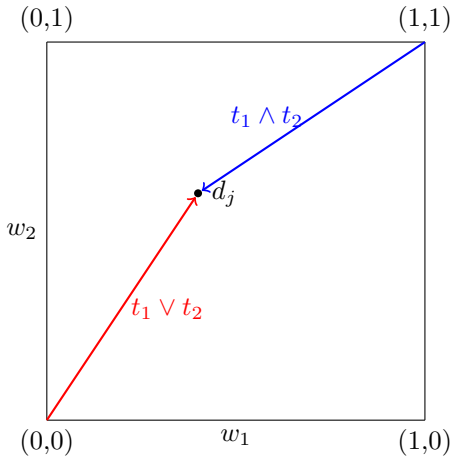
Pro účely této části je potřeba představit si pojem **metrika**, neboli vzdálenost. Zjednodušeně se jedná se o zobrazení, splňující určité vlastnosti, převádějící dvojici vektorů na kladná reálná čísla. Nejznámějším příkladem je **Eukleidovská metrika**, kterou pro dva vektory z $\mathbb{R}^k$ $\boldsymbol{x} = (x_1, \ldots, x_k)$ a $\boldsymbol{y} = (y_1, \ldots, y_k)$ můžeme vyjádřit následovně. [7]

$$\|\boldsymbol{x} - \boldsymbol{y}\|_2 = \mathrm{d}_2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.9}$$

Alternativou pro tuto základní metriku jsou pak takzvané $\mathbf{L}_k$ **metriky** (také označované jako Minkovského k-metriky nebo k-normy). Například L_2 je nám známá Eukleidovská metrika. Obecně tyto metriky můžeme vyjádřit následujícím předpisem. [7]

$$\|\boldsymbol{x} - \boldsymbol{y}\|_k = \mathrm{d}_k(\boldsymbol{x}, \boldsymbol{y}) = \sqrt[k]{\sum_{i=1}^{k} |x_i - y_i|^k} \tag{3.10}$$

Pro ilustraci si vezměme dotaz q složený ze dvou termů t_1 a t_2, spojený v jednom případě operátorem AND ($\wedge$) a ve druhém operátorem OR ($\vee$). Takový dotaz můžeme namapovat na 2-dimenzionální vektor $\boldsymbol{q} = (t_1, t_2)$. Tento příklad je znázorněný obrázkem 3.2. Pro dotaz q značí bod (1,1) ve 2-dimenzionálním prostoru případ, kdy se oba termy vyskytují v dokumentu. Naproti tomu bod (0,0) reprezentuje případ, kdy ani jeden z termů přítomný není. V tomto prostoru poté máme vyznačený dokument d_j, který je v určité vzdálenosti od obou těchto bodů.



Obrázek 3.2 Hledání pomocí rozšířeného booleovského modelu, inspirace v [6]

Maximální *Eukleidovská vzdálenost* v tomto prostoru je $\mathrm{dist}_{\max}$ rovna $\sqrt{2}$ – tedy vzdálenost mezi body (0,0) a (1,1). Pro AND dotaz poté měříme vzdálenost od bodu (1,1) a počítáme ji opět přes Eukleidovskou míru – ve výsledku dostáváme rovnici 3.11. Obdobně můžeme vypočítat vzdálenost od bodu (0,0) pro OR dotaz přes rovnici 3.12. V obrázku 3.2 je AND vzdálenost dokumentu d_j zobrazena modře, zatímco OR červeně. [6]

$$\mathrm{dist}_{\wedge}(d_j) = \sqrt{2} - \sqrt{(1 - w_{1,j})^2 + (1 - w_{2,j})^2} \tag{3.11}$$

$$\mathrm{dist}_{\vee}(d_j) = \sqrt{w_{1,j}^2 + w_{2,j}^2} \tag{3.12}$$

Podobnost mezi dokumentem a dotazem s použitím těchto znalostí poté můžeme vyjádřit jako relativní vzdálenost vůči té maximální. Pro AND variantu dotazu dostáváme rovnici 3.13 a pro OR rovnici 3.14. [6]

$$\mathrm{sim}_{\wedge}(d_j, \boldsymbol{q}) = \frac{\mathrm{dist}_{\wedge}(d_j)}{\mathrm{dist}_{\max}} = 1 - \sqrt{\frac{(1-w_{1,j})^2 + (1-w_{2,j})^2}{2}} \tag{3.13}$$

$$\mathrm{sim}_{\vee}(d_j, \boldsymbol{q}) = \frac{\mathrm{dist}_{\vee}(d_j)}{\mathrm{dist}_{\max}} = \sqrt{\frac{w_{1,j}^2 + w_{2,j}^2}{2}} \tag{3.14}$$

Zatím jsme uvažovali pouze jednoduché dotazy obsahující dva termy. Výše uvedené rovnice lze zobecnit i na prostory větších dimenzí než jen 2. S pomocí výše uvedených L_k metrik můžeme zkonstruovat podobnosti pro více termů v obou variantách dotazů. Předpokládejme, že náš dotaz tentokrát obsahuje k termů, tedy $\boldsymbol{q_k} = (t_1, ..., t_k)$. Výsledná podobnost pro k termů AND, resp. OR dotazu a dokumentu je zde popsána následujícími rovnicemi. [6]

$$\mathrm{sim}_{\wedge}(d_j, \boldsymbol{q}_k) = 1 - \sqrt[k]{\frac{(1-w_{1,j})^k + (1-w_{2,j})^k + ... + (1-w_{n,j})^k}{k}} \tag{3.15}$$

$$\mathrm{sim}_{\vee}(d_j, \boldsymbol{q}_k) = \sqrt[k]{\frac{w_{1,j}^k + w_{2,j}^k + ... + w_{n,j}^k}{k}} \tag{3.16}$$

S pomocí takto zkonstruovaných metrik podobnosti mezi dokumentem a dotazem, můžeme řadit výsledné dokumenty na základě vypočítané podobnosti. Nevýhodou těchto metrik je ovšem to, že nesplňují všechny potřeby booleovské algebry a dotaz proto nebude exaktně vyhovovat všem dotazům. Zároveň je systém, který toto porovnávání využívá, poměrně pomalý. [6]

3.4 Vektorový model

Nejlepším řešením ze zatím představených modelů je **vektorový model**, ve kterém jsou dokumenty a dotazy reprezentovány jako vektory v mnohorozměrném prostoru. Každá složka těchto vektorů odpovídá jednomu termu ze slovníku kolekce. Každý dokument d_j je zde pak namapován na m-dimenzionální vektor $\boldsymbol{d}_j = (\mathrm{w}(t_0, d_j), \mathrm{w}(t_1, d_j), ..., \mathrm{w}(t_m, d_j))^\intercal$. Analogicky můžeme pro dotaz q vyjádřit vektor dotazu jako $\boldsymbol{q} = (\mathrm{w}(t_0, q), \mathrm{w}(t_1, q), ..., \mathrm{w}(t_m, q))^\intercal$. Každý term poté představuje jednu osu ve výsledném prostoru, který neovlivňuje ty ostatní. Předpokládáme tedy implicitně nezávislost těchto jednotlivých příznaků dokumentu (termů). To ale vede k systematické chybě, vzhledem k tomu, že přirozený jazyk v sobě určitou závislost mezi slovy obsahuje. Mezi takto utvořenými vektory ve výsledku počítáme podobnost, podobně jako jsme to již dělali u rozšířeného vektorového modelu a stejným způsobem můžeme i řadit dokumenty ve výsledku. Části, ve kterých se tento model liší jsou tedy zejména způsob přiřazení vah jednotlivým termům v dokumentech a počítání podobnosti mezi dokumentem a dotazem. [5, 3]

3.4.1 Váhovací schéma tf-idf

Pro účely určení vah jednotlivých termů v dokumentech se využívá schéma **tf-idf**. Na rozdíl od základního booleovského modelu, ve kterém jsme zkoumali pouze fakt, zda je term v dokumentu přítomný nebo ne, budeme brát v potaz další charakteristiky. Konkrétně frekvenci termu v daném dokumentu – dokument obsahující velmi frekventované slovo bude logicky relevantnější vůči dotazu, který takové slovo obsahuje. Zároveň můžeme díky tomuto přístupu zvolnit nároky na exaktnost dotazů. Namísto logických operátorů můžeme na termy pohlížet podobně jako v dokumentu na neseřazenou množinu termů. Tento způsob je preferovaný pro klasické uživatele systémů a nevyžaduje žádné specifické znalosti o systému – je využívaný například ve webových vyhledávačích. [2]

První část tohoto schématu *tf* – tedy **term frequency** – vyjadřuje samotnou frekvenci termu t_i v dokumentu d_j (tento fakt značíme jako $\mathrm{tf}(t_i, d_j)$). Intuitivně tuto část můžeme chápat tak, že

term z dotazu, který se vyskytuje v daném dokumentu vícekrát, by měl mít pro tento dokument větší relevanci než pro dokument, kde se vyskytuje zřídka. [5, 2]

Kdybychom pro určení váhy používali čistě frekvenci slova v daném dokumentu, byla by všechna slova stejně důležitá. To ale nutně nemusí být pravda. Vezměme si *stop slova* diskutovaná v předchozí kapitole, která budou velmi frekventovaná ve všech dokumentech, ale v reálu nenesou žádný význam pro relevanci dokumentu vůči dotazu. Obdobně to bude například u slov, která jsou velmi frekventovaná v tématu daného corpu. Corpus právních předpisů bude nejspíš ve většině svých dokumentů obsahovat slova jako *„článek“* nebo *„paragraf“*, která ale v tomto kontextu nenesou prakticky žádný význam. [2]

K zabránění tohoto efektu zavedeme **document frequency**, která je definována jako počet dokumentů v kolekci, které obsahují term t_i – označujeme jako $\mathrm{df}(t_i)$. Pro účely váhování se ale používá invertovaná hodnota s použitím počtu dokumentů v kolekci nazývaná jako **inverted document frequency** (*idf*). Ta je definovaná následovně.

$$\mathrm{idf}(t_i) = \log \frac{N}{\mathrm{df}(t_i)} \tag{3.17}$$

Ve vztahu se využívá logaritmus, a to z důvodu zmírnění dopadu míry idf na celkový výsledek. Zároveň chceme, aby slovo, které se vyskytuje téměř ve všech dokumentech, mělo co možná nejmenší váhu – tomu odpovídá $\log \frac{N}{N} = 0$. Naopak v momentě, kdy se slovo vyskytuje jen v jednom specifickém dokumentu bude hodnota $\log \frac{N}{1}$ největší možná.

Výslednou váhu, přiřazenou jednotlivým termům v dokumentech, vypočítáme pomocí následujícího vztahu – tedy jako násobek výše popsaných dílčích metrik *tf* a *idf*. [2]

$$\text{tf-idf}(t_i, d_j) = \mathrm{tf}(t_i, d_j) \cdot \mathrm{idf}(t_i) \tag{3.18}$$

3.4.2 Kosinová podobnost

Následující sekce předpokládá, že má čtenář alespoň základní znalosti pojmů z oblasti lineární algebry. Pro vektor $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^k$ můžeme vyjádřit jejich vzdálenost pomocí již definované Eukleidovské metriky. Skalární součin těchto vektorů je pak určen následujícím vztahem.

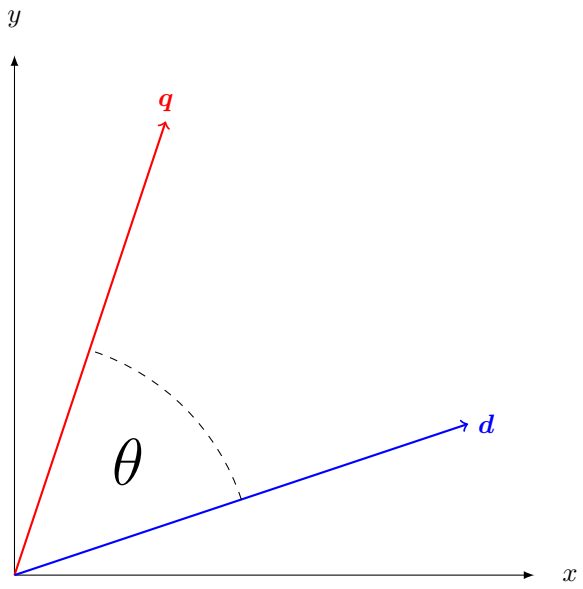
$$\boldsymbol{x} \cdot \boldsymbol{y} = \sum_{i=1}^{p} x_i y_i \tag{3.19}$$

Druhá odlišná část, kterou jsme zmínili při úvodu vektorového modelu, je samotný výpočet podobnosti mezi dokumentem a dotazem. Pro tyto účely se využívá **kosinová podobnost**. Jednoduchá představa, jak taková metrika funguje, je naznačena obrázkem 3.3. Pro jednotlivé vektorové reprezentace dokumentů $\boldsymbol{d}$ počítáme úhel θ mezi nimi a vektorem dotazu $\boldsymbol{q}$. Čím menší úhel je, tím je pak větší podobnost mezi dokumentem a dotazem. Pro popis úhlu můžeme využít lineární algebry, která ho vyjadřuje pomocí následujícího vztahu. [2, 3]

$$\boldsymbol{d} \cdot \boldsymbol{q} = \|\boldsymbol{d}\| \cdot \|\boldsymbol{q}\| \cdot \cos(\theta) \tag{3.20}$$

Jednoduchým přeuspořádáním této rovnice můžeme dostat *kosinovou podobnost* určenou vztahem popsaným níže. Mohli bychom přímo dopočítat úhel θ, ale vzhledem k monotónní povaze kosinu vzhledem k úhlu můžeme jako míru brát rovnou samotný $\cos(\theta)$. [9]

$$\mathrm{sim}_{\cos}(\boldsymbol{d}, \boldsymbol{q}) = \cos(\theta) = \frac{\boldsymbol{d} \cdot \boldsymbol{q}}{\|\boldsymbol{d}\| \cdot \|\boldsymbol{q}\|} \tag{3.21}$$



Obrázek 3.3 Kosinová podobnost mezi dokumentem a dotazem

Kapitola 4

Word embedding techniky

V kapitole 3 jsme si představili jednoduché information retrieval modely založené na paradigmatu *bag of words*, kde je jakákoliv informace o vnitřní struktuře textu zanedbána. Významným problémem těchto modelů, který možná není na první pohled natolik zřejmý, je vysoká dimenzionalita reprezentací jednotlivých slov. Ať už pracujeme s vektorovým nebo booleovským modelem, přiřazujeme každému slovu váhy, resp. binární indikátory, na základě jejich výskytů v dokumentu. Vezměme ale v potaz, že některé corpy mohou čítat i desetitisíce slov a z toho plyne nepřiměřená velikost jejich vektorové reprezentace. V takovém prostoru budeme narážet jak na nedostatečnou rychlost výpočtu vzdáleností mezi těmito reprezentacemi, tak na fenomén *prokletí dimenzionality*[1], který v zásadě způsobuje velké rozdíly ve vzdálenostech a obecně nejasnost interpretace vypočtených vzdáleností.

Tyto problémy do velké míry řeší pokročilejší **word embedding** techniky. Tento pojem, do češtiny překládaný jako *vnoření slov*, v zásadě postihuje i dříve představené techniky. Jedná se obecně o vytvoření vektorové reprezentace z daného slova. Jednoduchým příkladem může být i **one-hot encoding**, který jsme v podstatě již použili při představení booleovského modelu, i když ne explicitně. Pro každé slovo z dokumentu, které chceme zakódovat dostáváme vektor, jehož i-tá složka obsahuje binární příznak, popisující výskyt slova v dokumentu d_i z kolekce. Vezměme si jako příklad kolekci $\mathcal{D}_5 = \{d_1, \ldots, d_5\}$ a slovo t, které se vyskytuje pouze v dokumentech $d_1, d_3 \in \mathcal{D}_5$. Zakódováním takového slova přes *one-hot encoding* bychom dostali vektor $\boldsymbol{w} = (1, 0, 1, 0, 0)$. Obdobně můžeme tento přístup aplikovat na jednu větu, kde namísto dokumentů bereme v potaz výskyt ve větách. [1]

Dalším problémem dříve představených modelů, je práce se *synonymy* a *homonymy*. Když se budeme soustředit na vyhledávací fázi IR systémů, tak v případě práce se synonymy nebudeme často dostávat dostatek relevantních dokumentů. Pokud uživatel zadá dotaz, nemusíme dostat dokumenty, které obsahují synonyma s tímto slovem. To může být velmi často problém, vzhledem k množství synonym a obecné neurčitosti jazyka. V opačném případě, kdy má slovo více významů, nemůžeme bez jakéhokoliv kontextu vědět, jaký z nich má uživatel na mysli a jako výsledek dotazu mu můžeme nabízet irelevantní dokumenty. [5]

Základem pro fungování pokročilejších technik je takzvaná *distribuční struktura* zkoumaná lingvisty 20. století, jako byli například Z. S. Harris a J. R. Firth. Tento princip lze jednoduše vyjádřit výrokem druhého zmíněného:

„*You shall know a word by the company it keeps*“ (Firth, J. R. 1957:11)

Distribuční struktura se zabývá teorií, že slova se vyskytují vždy v určitém kontextu a jejich

[1] „*The curse of dimensionality, [...], indicates that the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with respect to the number of input variables (i.e., dimensionality) of the function.*“ [8]

výskyt není nahodilý. Tento fakt lze aplikovat i na obecnou mluvu a nejedná se tedy o strukturu danou pevnou gramatikou a pravidly jazyka. Pro každý kontext poté má smysl odhadovat slovo, které by do něj patřilo. Právě na tomto faktu staví pokročilejší word embedding techniky jako je *NNLM* a *Word2Vec*, které si v této kapitole představíme. [9]

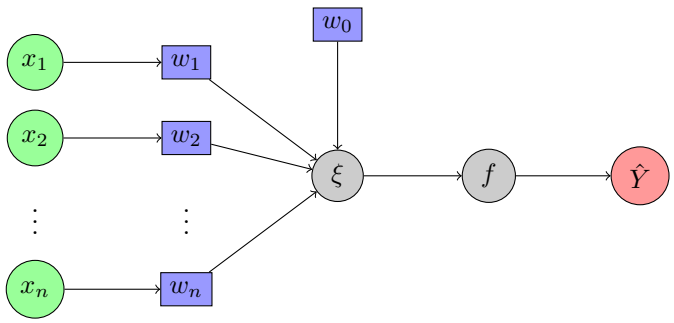
Díky této struktuře slov můžeme při práci s kontextem zachytit i vztahy, jako je sémantika, podobnost nebo odhadnutí synonym pro dané slovo. Samotným cílem těchto modelů je pak vytvoření vektorových reprezentací, které shlukují podobná slova blízko sebe. Jedná se o modely využívající *nesupervizovaného učení* – tedy učení bez učitele – které se snaží samy z dat získat informace o struktuře. Zpravidla při práci s těmito modely budeme předpokládat, že máme k dispozici předzpracovaný text, rozdělený na jednotlivé tokeny, jak bylo popsáno v kapitole 2. Pro konzistenci s dostupnou literaturou bude v rámci této kapitoly použito pro i-té slovo označení w_i a slovník všech slov z daného corpu bude nadále značen jako $\mathcal{T}$. [1, 10]

4.1 Základy neuronových sítí

„*An artificial neural network (ANN), usually called neural network (NN), is a mathematical model or computational model that is inspired by the structure and/or functional aspects of biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. In most cases, an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Modern neural networks are nonlinear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data.*“ [11]

Trochu konkrétněji bych v této sekci předvedl základní architektury neuronových sítí a používané aktivační funkce. Nejjednodušší model, který můžeme představit je **jednovrstvý perceptron**. Jedná se o model obsahující jediný neuron, který můžeme vysvětlit pomocí obrázku 4.1. Na vstupu takového neuronu máme vektor $\boldsymbol{x} = (x_1, ..., x_n)$ a na výstupu výslednou predikci $\hat{Y}$ na základě těchto vstupních dat. Zároveň má samotný neuron váhy, které řídí výpočet a které se v průběhu tréninku aktualizují – můžeme je reprezentovat vektorem $\boldsymbol{w} = (w_1, ..., w_n)$, kde každá individuální váha odpovídá jednomu vstupu na neuronu. Dále je jeho součástí speciální váha w_0, nazývaná *intercept*, což je v podstatě hodnota predikce při nulových příznacích. Výsledkem kombinace vstupů $\boldsymbol{x}$ a vah $\boldsymbol{w}$ je vnitřní potenciál ξ vyjádřený následovně:

$$\xi = w_0 + \sum_{i=1}^{n} w_i x_i = \boldsymbol{w}^\intercal \boldsymbol{x} + w_0 \tag{4.1}$$

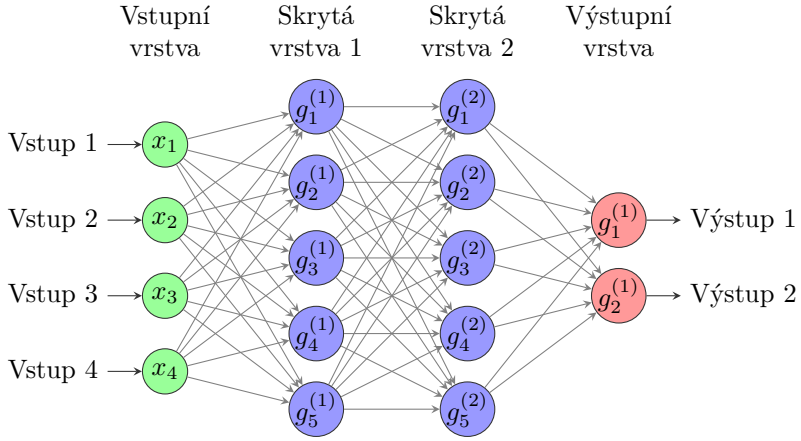


Obrázek 4.1 Jednovrstvý perceptron

Výstup neuronu vzniká aplikací takzvané *aktivační funkce*, která je používaná v různých alternativách – ty jsou popsány dále. Obecně ale pro výstup platí následující. [7]

$$\hat{Y} = f(\xi) \tag{4.2}$$

„V tomto modelu se neuronová síť skládá z vrstev, které jsou propojené tak, že výstupy neuronů z jedné vrstvy tvoří výstupy neuronů do další vrstvy.“ Občas je tento model označován také jako *feedforward neural network*. Zjednodušené schéma této architektury máme na obrázku 4.2, všechny vrstvy, kromě vstupní a výstupní se nazývají skryté. Každý neuron poté převádí vstupy na výstupy tak, jak to bylo popsáno v předchozí sekci. Celá neuronová síť je poté vlastně složená funkce, která převádí vektor na vstupu do vektoru na výstupu. [7]



Obrázek 4.2 Vícevrstvá neuronová síť

Mějme takto vytvořenou síť, která má l vrstev, kde každá vrstva má postupně $n_1, \ldots, n_l$ neuronů. Pro tuto síť můžeme i-tou vrstvu reprezentovat jako vícehodnotovou funkci $\boldsymbol{g}^{(i)} : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, kde $\boldsymbol{g}^{(i)} = (g_1^{(1)}, \ldots, g_{n_i}^{(1)})$. Celá neuronová síť pak vznikne spojením těchto vrstev do funkce $\boldsymbol{g} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$. Ta má následující podobu. [7]

$$\boldsymbol{g} = \boldsymbol{g}^{(l)} \circ \boldsymbol{g}^{(l-1)} \circ \cdots \circ \boldsymbol{g}^{(2)} \circ \boldsymbol{g}^{(1)} \tag{4.3}$$

Existují různé speciální architektury neuronových sítí. Zajímavým příkladem jsou *rekurentní neuronové sítě (RNN)*, které využívají jako vstup do neuronu výstupy z předchozích kroků. Dále se používají *konvoluční neuronové sítě (CNN)*, které berou v potaz lokální oblasti a na nich staví globální znalosti. [7]

Pro výběr aktivační funkce máme více alternativ. Nejjednodušší je **prahová funkce**, která klasifikuje výstup neuronu binárně na základě prahové hodnoty volené jako 0. Samotná binární klasifikace pak tedy probíhá podle následujícího vzoru:

$$f(\xi) = \begin{cases} 1 & \text{když } \xi \geq 0 \\ 0 & \text{když } \xi < 0 \end{cases} \tag{4.4}$$

Alternativou pro binární klasifikaci, která je využívaná podstatně častěji je **sigmoida**. Převádí vnitřní potenciál neuronu do intervalu $[0, 1]$ – výstupem této funkce je poté vlastní příslušnost ke třídě 1. Tato funkce má zároveň výhodu, že je všude diferencovatelná, což je důležitý atribut při učení neuronových sítí.

$$f(\xi) = \sigma(\xi) = \frac{1}{1 + e^{-\xi}} \tag{4.5}$$

Další z aktivačních funkcí je poté **hyperbolický tangens**, který je v zásadě velmi podobný jako sigmoida, ale převádí hodnoty do intervalu $[-1, 1]$.

$$f(\xi) = \tanh(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}} \tag{4.6}$$

Dále se používá také **RELU** neboli *oříznutá lineární funkce*. Jedná se o nelineární funkci s nenulovými derivacemi v kladné části, které zajišťují dobré výsledky v procesu učení.

$$f(\xi) = \begin{cases} \xi & \text{když } \xi \geq 0 \\ 0 & \text{když } \xi < 0 \end{cases} \tag{4.7}$$

V případě, kdy řešíme klasifikaci do c tříd na výstupní vrstvě, používáme většinou **softmax** funkci. Jedná se o kombinaci několika sigmoid, kdy jedna z nich určuje příslušnost k jedné z c tříd. Pro c neuronů na výstupů, které mají vnitřní potenciály $\xi_1, ..., \xi_c$ ji definujeme následovně.

$$f_i(\xi) = \frac{\mathrm{e}^{\xi_i}}{\sum_{k=1}^{c} \mathrm{e}^{\xi_k}} \tag{4.8}$$

[7, 12]

„*Při učení se snažíme minimalizovat chybu predikce měřenou pomocí průměrné hodnoty ztrátové funkce L na trénovací množině.*“

$$J(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} L(Y_i, g(\boldsymbol{x}_i, \boldsymbol{w})) \tag{4.9}$$

Samotná minimalizace se poté provádí gradientním sestupem, při kterém se používá derivace složené vícehodnotové funkce reprezentující samotnou neuronovou síť. „*Při výpočtu tedy dochází k postupnému pronásobování a sčítání parciálních derivací vrstev ve směru od výstupní vrstvy směrem k vstupní – tedy k tzv. zpětnému šíření*“. [7]

4.2 Klasický neuronový jazykový model (NNLM)

Pro bližší přiblížení hlavního zájmu této práce bych v této části představil model, který pracuje s použitím výše uvedených principů a stal se základem pro později vytvořený *Word2Vec*. Tento model, představený kanadskou skupinou vědců pod vedením Yoshua Bengia v článku *A neural probabilistic language model* [13], je zde nazýván jako **Feedforward Neural Net Language Model** (**NNLM**). Toto označení vychází z pozdějšího článku Tomáše Mikolova. [14]

Před představením samotného modelu je na tomto místě potřeba nastínit pravděpodobnostní pohled na kontext slova. Mějme M slov z corpu, se kterým pracujeme, tedy $w_1, \ldots, w_M$. Obecně se modely, které se zabývají word embeddingem snaží predikovat pravděpodobnost následujícího slova w_t na základě $n-1$ předchozích slov. To můžeme vyjádřit následujícím způsobem, kde $\hat{P}(x|y) = p(x|y)$ označuje odhad podmíněné pravděpodobnosti. [10]

$$p(w_t | w_{t-1}, \ldots, w_{t-n+1}) \tag{4.10}$$

Zatím ale uvažujeme pouze slova, která se v našem corpu vyskytují. Problém nastává v momentě, kdy predikujeme slovo, které v corpu obsaženo není. Navíc takto nedáváme důraz na podobná slova. Například slova „kočka“ a „pes“ se budou velmi pravděpodobně vyskytovat v podobném kontextu. Na základě této podobnosti by měl být model schopný odhadovat jedno slovo na základě toho druhého. [13]

Model navržený Bengiem, který by měl tyto problémy řešit je obecně dán jako funkce.

$$f(w_t, \ldots, w_{t-n+1}) = p(w_t | w_{t-1}, \ldots, w_{t-n+1}) \tag{4.11}$$

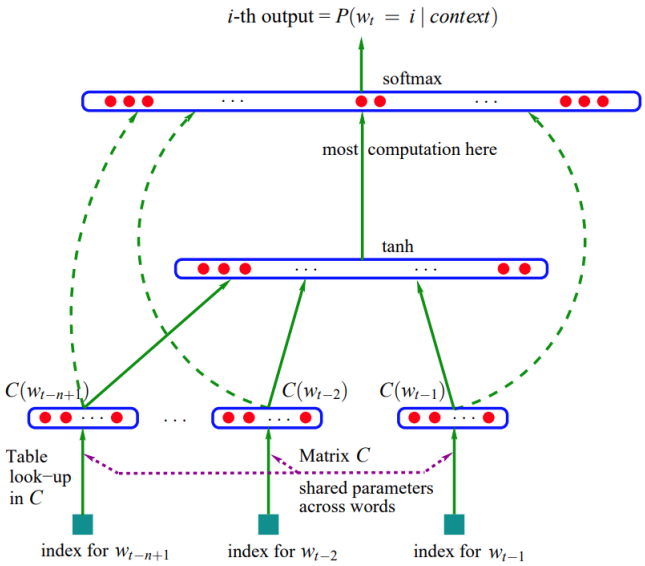
Tu můžeme rozdělit na dvě části. První z nich je zobrazení C, které mapuje jakékoliv slovo w_i na reálný vektor $\boldsymbol{v}_{w_i} = C(w_i) \in \mathbb{R}^l$. V praxi se jedná o matici $\boldsymbol{C} \in \mathbb{R}^{M,l}$ s volnými parametry. Tato matice ve výsledku obsahuje po řádcích vektorové reprezentace jednotlivých slov v kontextu.

Druhou částí je pravděpodobnostní funkce g, která mapuje vstupní vektorové reprezentace jednotlivých slov $\big(\boldsymbol{C}(w_{t-n+1}), \ldots, \boldsymbol{C}(w_{t-1})\big)$ na pravděpodobnostní rozdělení nad slovy z V pro další slovo w_t. Výsledkem této funkce je poté vektor, jehož i-tá složka vyjadřuje odhad pravděpodobnosti $p(w_t = i | w_{t-1}, \ldots, w_{t-n+1})$. Funkci jako takovou můžeme vyjádřit následovně. [13]

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1})) \tag{4.12}$$

Funkce f je tedy ve výsledku spojení těchto dvou zobrazení C a g, kde C je sdílené všemi slovy v kontextu.

Architektura Zobrazení g je v praxi implementováno jako dopředná neuronová síť, zobrazena na obrázku 4.3. V této síti jsou použity tři vrstvy [13, 10]:



Obrázek 4.3 Architektura NNLM, obrázek převzat z [13]

- *embedding vrstva* využívající matici $\boldsymbol{C}$ a generující vektorové reprezentace pro jednotlivá slova.
- mezilehlé *skryté vrstvy*, kterých může být potenciálně více[2]. Aplikují nelineární zobrazení propojující informace z jednotlivých vektorových reprezentací slov z kontextu. Bengio navíc ve svém článku navrhoval místo této vrstvy použití LSTM sítě[3]. [13].
- výstupní *softmax vrsta*, která produkuje výsledné pravděpodobnostní rozdělení nad slovy z V. Tato vrstva je podle Bengia poměrně problematická, vzhledem k tomu, že cena výpočtu je závislá na velikosti slovníku V. Konkrétně lze její výpočet vyjádřit pomocí následujícího vztahu. [13].

[2] Pro zjednodušení zde uvažujeme ale pouze jednu.
[3] *Long short-term memory* je NN architektura využívající rekurentních neuronových sítí.

$$p(w_t|w_{t-1},...,w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_{w_i}}} \tag{4.13}$$

kde y_{w_i} jsou pravděpodobnosti pro každé výstupní slovo w_i vypočítané jako výstup mezilehlých vrstev. Ty jsou součástí vektoru $\boldsymbol{y} \in \mathbb{R}^n$, který je vypočtený následovně. [13].

$$\boldsymbol{y} = \tanh(\boldsymbol{h}^\intercal \boldsymbol{X}) \tag{4.14}$$

Zde je vhodné poznamenat, že pro jednodušší pochopení byla odebrány případná volitelná přímá projení se softmax vrstvou (v obrázku 4.3 označena přerušovanou zelenou čárou) a vynechány *bias* hodnoty použité v původním článku. V této rovnici jsou $\boldsymbol{h} \in \mathbb{R}^l$ váhy skryté vrstvy a $\boldsymbol{X} \in \mathbb{R}^{l,n}$ je spojení vstupních vektorových reprezentací jednotlivých slov z kontextu vyjádřená výrazem níže. Hyperbolický tangens je v tomto případě aplikován na jednotlivé složky vektoru. [13]

$$\boldsymbol{X} = (C(w_{t-1}), C(w_{t-2}), \ldots, C(w_{t-n+1})) \tag{4.15}$$

Při učení tohoto modelu minimalizujeme následující funkci přes všechna slova ve slovníku. Pro zjednodušení je zde vynechán regularizační člen. V tomto kontextu vyjadřuje parametr θ spojení volných parametrů matice $\boldsymbol{C}$ a vah neuronové sítě.

$$J_\theta = \frac{1}{M} \sum_{t=1}^{M} \log f(w_t, w_{t-1}, ..., w_{t-n+1}) \tag{4.16}$$

Oproti původnímu článku Bengia je vysvětlení značně zjednodušené pro účely této práce a místy jsou vynechány části, které by mohly být pro efektivní fungování modelu klíčové. Pro bližší pochopení mohu čtenáře odkázat přímo na původní Bengiův článek o tomto modelu. [13].

4.3 Word2Vec

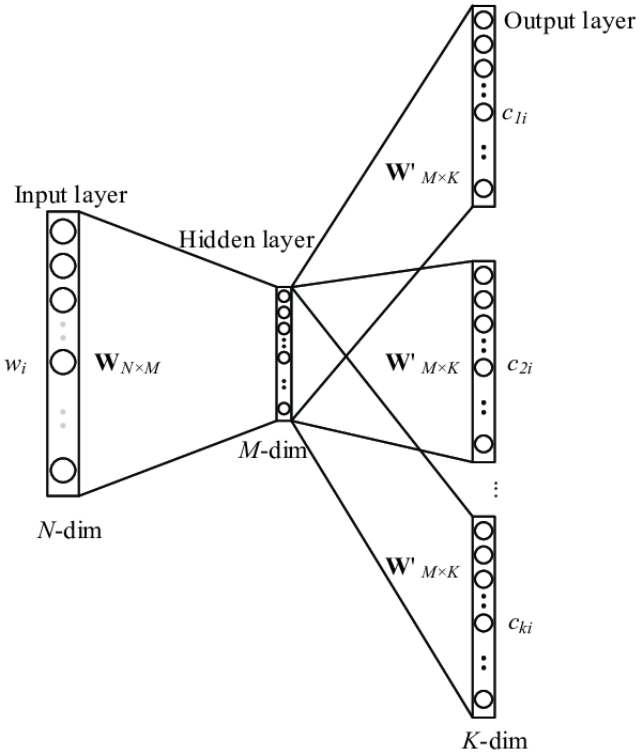
Tento model, založený opět na neuronových sítích, byl poprvé představen v článku *Efficient Estimation of Word Representations in Vector Space* [14] publikovaném týmem z Google. Tomáš Mikolov a kolektiv ve svém článku staví na dříve představeném modelu NNLM, který byl v té době době velmi populární, a zjednodušuje ho. Tento model, respektive modely, se staly velmi oblíbenými, především z důvodu rychlosti učení a vysoké přesnosti. Jsou schopné naučit se reprezentace z miliardy slov za méně než den, což bylo v té době poměrně revoluční. V článku byly představeny dva modely – v zásadě mají velmi podobnou architekturu, ale pracují jiným způsobem s kontextem. Jedná se o modely *Continuous Bag-of-Words* a *Continuous Skip-Gram*. [14, 10]

Narozdíl od NNLM bere **Word2Vec** v potaz kontext, který se vyskytuje před i po zkoumaném slově. Cílem je ale stejně jako v předchozím případě porozumění pravděpodobnosti spoluvýskytu dvou nebo více slov. Na základě této naučené pravděpodobnosti můžeme pozorovat spojení mezi různými slovy, které mají mezi sebou podobný vztah – například slova „king“ a „man“ budou mít podobný vztah jako slova „queen“ a „woman“. Z toho vychází také notoricky známý příklad $king - man + woman = queen$. Tento model je schopný naučit se i vztahy mezi synonymy, což může být velmi přínosné například pro vyhledávání. [1]

4.3.1 Architektura

Vzhledem k velké podobnosti obou architektur můžeme detailnější popis nechat společný pro oba výše uvedené modely. V této části můžeme navázat na architekturu *NNLM* a vycházet z rozdílů s ní. Obdobně jako v předchozím případě máme v modelu tři vrstvy, ale tentokrát nemáme žádná volitelná propojení mezi vstupní a výstupní vrstvou a nelineární transformace v ní. Jako vstupní vektor se považuje *one-hot encoding* pro dané slovo, respektive kontext, což je rozdíl oproti projekční matici $\boldsymbol{C}$. Ta byla sdílená pro daný kontext a její složky byly proměnlivými parametry. Vstupní vektory mají tedy velikost M, což vychází z počtu unikátních slov v kolekci.

Pro další práci uvažujme, že máme na vstupu jedno slovo a na výstupu k slov. Toto je případ *Skip-Gram* modelu. U *CBOW* architektury bychom akorát zaměnili význam vstupu a výstupu. Každý z těchto modelů má své využití. Hlavní přednost *CBOW* oproti Skip-Gram modelu je, že se učí rychleji a má vyšší přesnost pro frekventovaná slova. Oproti tomu Skip-Gram má lepší výsledky při potřebě brát v potaz slova, která se nevyskytují tak často. Z tohoto důvodu je CBOW varianta preferovaná pro větší kolekce dokumentů. [1, 14]



Obrázek 4.4 Architektura Skip-Gram modelu, obrázek převzat z `https://towardsdatascience.com/a-word2vec-implementation-using-numpy-and-python-d256cf0e5f28`

Hlavním rozdílem oproti *NNLM* je **skrytá vrstva**. Máme zde již pouze jednu a počet neuronů v ní se odvíjí od hyperparametru l, který ve výsledku určuje dimenzi výsledných vektorových reprezentací slov. Na vstupu a výstupu této vrstvy máme jednotlivé váhy. Jak si můžeme všimnout na diagramu 4.4 tyto váhy můžeme vyjádřit jako matice $\boldsymbol{W} \in \mathbb{R}^{M,l}$ (váhy na vstupu) a $\boldsymbol{W'} \in \mathbb{R}^{l,M}$ (váhy na výstupu). Oproti diagramu je zde použito jiné značení pro dimenze vektorů – konkrétně na vstupní vrstvě máme vektory dimenze M (na diagramu M) a výsledné vektorové reprezentace mají dimenzi l (na diagramu M). Pro každé slovo w_i pak máme dvě vektorové reprezentace – vstupní reprezentaci $\boldsymbol{v}_{w_i}$ obsaženou v i-tém řádku matice $\boldsymbol{W}$ a výstupní reprezentaci $\boldsymbol{v}'_{w_i}$ obsaženou v i-tém sloupci matice $\boldsymbol{W'}$. Obě tyto reprezentace už jsou vektory

dimenze l. Na rozdíl od klasických neuronových sítí na výstup této vrstvy neaplikujeme žádnou aktivační funkci a posíláme ho rovnou do výstupní vrstvy. Po aplikaci softmax funkce na výstupní vrstvu získáváme predikce kontextu pro dané slovo – opět ve formě *one-hot encoding*.

Pro středové slovo w_I a slovo kontextu w_O bychom mohli spočítat podmíněnou pravděpodobnost výskytu středového slova w_I při kontextovém slově w_O, která bude klíčová pro pozdější fázi tréninku. To provedeme pomocí softmax funkce následovně[4].

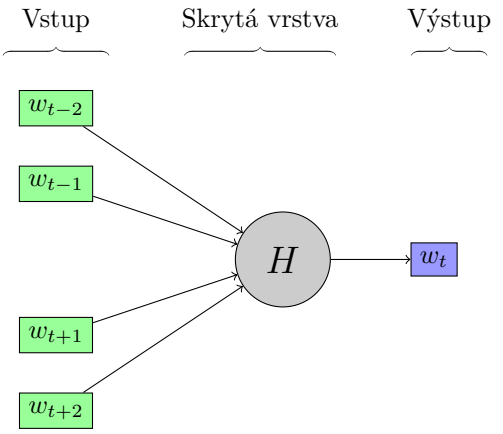
$$p(w_O|w_I) = \frac{\exp({\boldsymbol{v}'_{w_O}}^\intercal \boldsymbol{v}_{w_I})}{\sum_{w \in V} \exp({\boldsymbol{v}'_{w}}^\intercal \boldsymbol{v}_{w_I})} \tag{4.17}$$

Oproti výpočtu v *NNLM* (viz rovnice 4.13) zde využíváme vstupní a výstupní vektorovou reprezentaci slova a na skrytou vrstvu neaplikujeme žádné aktivační funkce. [10, 15, 16, 17]

Velkou změnou v tomto modelu je, že matice vah $\boldsymbol{W}$ a $\boldsymbol{W'}$ jsou sdílené pro všechna slova v kolekci, a ne pouze pro daný kontext, jako tomu bylo u matice $\boldsymbol{C}$ v NNLM. Z tohoto důvodu se v průběhu učení neustále zdokonalují vektorové reprezentace jednotlivých slov. V zásadě potřebujeme u obou modelů velké množství slov, na kterých se budou trénovat – většinou se pohybujeme v řádech miliónů slov. Výběr konkrétního typu architektury velmi závisí na corpu, se kterým pracujeme.

4.3.2 Continuous Bag-of-Words model (CBOW)

První z Word2Vec modelů predikuje každé slovo na základě jeho kontextu, což je vlastně chování, které jsme viděli i u NNLM. Rozdíl je v tom, že pro slovo w_t nebere $n-1$ slov před ním, ale pro předem určenou velikost okna c se zaměřuje na c slov před a c slov po zkoumaném slově. Samotný název modelu pak vychází podle autorů z faktu, že při projekci nejsou zohledňována slova v historii učení. Zjednodušený princip fungování je znázorněn na obrázku 4.5 – velikost okna c je zde nastavena na 2 a H zde znázorňuje skrytou vrstvu sítě. [18, 14]



Obrázek 4.5 Znázornění principu fungování CBOW modelu

Vidíme tedy, že na vstupu budou kontextová slova a na výstupu jedno středové, které predikujeme. Jako příklad bychom mohli vzít větu „Dnes venku svítí slunce“ a velikost kontextu $c = 2$. Z té bychom postavili vstup a výstup tak, jak je vidět v tabulce 4.1. Do modelu by poté přicházela tato slova zakódovaná pomocí *one-hot encoding*. V případě CBOW by na vstup přicházel kontext a na výstupu by bylo středové slovo. [1, 14]

Při učení minimalizujeme účelovou funkci, která se příliš neliší od té používané v případě NNLM – v podstatě se v tomto případě změnil jen kontext, na který pohlížíme a samozřejmě

[4] Tato funkce uvažuje Skip-Gram model. V případě CBOW bychom akorát zaměnili roli slov.

Věta	Středové slovo	Kontext
Dnes venku svítí slunce	Dnes	venku, svítí
Dnes venku svítí slunce	venku	Dnes, svítí, slunce
Dnes venku svítí slunce	svítí	Dnes, venku, slunce
Dnes venku svítí slunce	slunce	venku, svítí

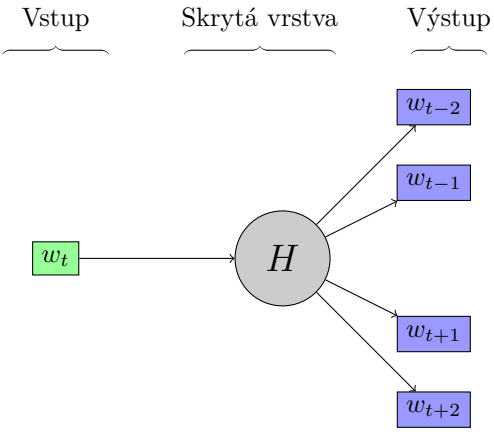
Tabulka 4.1 Tvorba kontextu a středového slova u Word2Vec modelů, červeně jsou označená středová slova.

parametry se kterými pracujeme. V našem případě už nemáme matici s volnými parametry $\boldsymbol{C}$ a namísto toho máme váhy $\boldsymbol{W}$ a $\boldsymbol{W}'$. [1, 14, 10]

$$J_\theta = \frac{1}{M}\sum_{t=1}^{M} \log p(w_t|w_{t-n},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+n}) \tag{4.18}$$

4.3.3 Continuous Skip-Gram model

Druhá architektura je podobná, ale namísto predikce aktuálního slova na základě kontextu se snaží predikovat slova kolem něj. Toto je schematicky znázorněno obrázkem 4.6. V tomto modelu se také bere v potaz, že vzdálenější slova mají v kontextu menší relevanci než ty bližší a přiřazují se jim proto jiné váhy. S rostoucí vzdáleností od středového slova se zlepšují naučené vektorové reprezentace, ale zároveň roste výpočetní náročnost. [1, 14, 10]



Obrázek 4.6 Znázornění principu fungování Continuous Skip-Gram modelu

Účelová funkce tohoto modelu je poměrně složitější než v předchozím případě. Pro slova $w_1,\ldots,w_t$ jí můžeme vyjádřit následujícím způsobem. [10, 15, 19]

$$J_\theta = \frac{1}{M}\sum_{t=1}^{M}\Big[\sum_{j=-k}^{k} \log p(w_{t+j}|w_t)\Big] \tag{4.19}$$

4.3.4 Optimalizace modelu

V rovnici 4.17 jsme si ve jmenovateli mohli povšimnout, že dochází k poměrně výpočetně náročným operacím – konkrétně se zde počítá exponenciální funkce pro každé ze slov ve slovníku. Obecně můžeme mít často ve slovníku V velké množství záznamů (kolem 10^5–10^7 slov). Ve

fázi trénování, kdy počítáme gradient pro tento výraz, se dostáváme ke znatelnému zpomalení. Tento problém identifikoval už Yoshua Bengio při návrhu svého modelu NNLM, a proto přišel s optimalizací, kterou později využívá i Mikolov. Jedná se o **hierarchický softmax**. [15]

V této technice se konstruuje binární strom, kde listy v tomto stromu jsou samotná slova. Namísto M kroků výpočtu pro každé slovo dostáváme v nejlepším případě[5] $\log_2 M$ kroků. Ke každému slovu se tedy můžeme dostat jedinou cestou od kořene a v každém uzlu postupně počítáme pravděpodobnost na základě zvolené cesty. Výsledná pravděpodobnost slova při daném kontextu je poté součinem pravděpodobností těchto jednotlivých uzlů. Použití tohoto stromu může pro velké slovníky rapidně zvýšit výkonnost modelu. Mikolov ve svém článku také navrhuje pro binární strom využití Huffmanova stromu[6], který přiřazuje jednotlivým slovům kratší kódy a vede k celkovému zvýšení rychlosti tréninku. [15, 20, 21]

Alternativou k hierarchickému softmaxu je *Noise Contrastive Estimation (NCE)* a jeho zjednodušení předvedené v rámci práce Tomáše Mikolova *Distributed Representations of Words and Phrases and their Compositionality* [15] **Negative Sampling (NEG)**. V zásadě se jedná o použití šumových slov, vygenerovaných z předem určené pravděpodobnostní distribuce pro každé slovo. Nad těmito slovy se poté vytváří binární klasifikátor, který určuje, zda je dané slovo relevantní nebo ne. Model využívající tuto optimalizaci je zkráceně označován jako *Skipgram with Negative Sampling*. [15, 20]

Další přístup, jak zlepšit naučené vektorové reprezentace je využití morfologie daných slov. Tímto se zabýval článek *Enriching Word Vectors with Subword Informations* [22]. V něm se představuje model využívající n-gramů daného slova pro vylepšení jeho vektorové reprezentace. Pro předem definované okno n se dané slovo rozdělí na jednotlivé části zahrnující speciální znaky pro konce slov. Tedy například pro slovo „královna“, by tento model při nastaveném okně $n = 4$ slovo rozdělil na následující části.

```
<krá, král, rálo, álov, lovn, ovna, vna>
```

Tyto části jsou pak brány v potaz v rámci skórovací funkce přidané k již existujícímu Skip-Gram modelu. Samotné n-gramy jsou pro větší efektivitu kódovány pomocí hashe. Tento model je v některých materiálech označován jako **FastText**[7]. [22]

4.4 GloVe

Další z generace word embedding modelů je **GloVe**, který vznikl jako zkrácení výrazu *global vectors* a na jeho výzkumu se podílela NLP skupina ze Stanfordu[8]. Jak již název napovídá, tento model se na rozdíl od Word2Vecu zaměřuje na globálnější charakteristiky slov vůči kolekci, kde se vyskytují. Problém s předchozím modelem byl, že se vždy zaměřoval jen na svoje bezprostřední okolí v nastavené šířce kontextu a nebral v potaz frekvence slov napříč dokumentem, posléze kolekcí. GloVe místo predikce výskytu slova využívá matici spolu-výskytů. Namísto samotné pravděpodobnosti, že se slova budou vyskytovat pospolu, jako jsme to předpokládali u Word2Vec modelu, GloVe napřímo spojuje vztahy mezi jednotlivými slovy a pracuje s nimi. [1, 23]

Pro lepší pochopení vztahů mezi jednotlivými spoluvýskyty slov autoři uvádí příklad se slovy *ice* a *steam*. Pravděpodobnost, že se tato jednotlivá slova budou vyskytovat v kontextu slov, která k nim jsou relevantní – například pro slovo *steam* slovo *gas* – bude vysoká. Naproti tomu, kdybychom vzali v potaz slovo *gas* v kontextu slova *ice*, pravděpodobnost, že se budou vyskytovat společně je nízká. Tato informace poskytuje určité souvislosti a vztahy mezi slovy podobně jako to na základě distribuční struktury dělal Word2Vec.

[5] Tedy v případě vyváženého stromu.

[6] Huffmanovo kódování přiřazuje jednotlivým slovům kód určité délky, která se odvíjí od jeho frekvence. Častější slova tedy budou mít kratší kódy a práce s nimi bude efektivnější.

[7] Konkrétně v jazyce Python existuje tento model právě pod jménem FastText v knihovně Gensim.

[8] Viz `https://nlp.stanford.edu/`.

Matice spoluvýskytů (*co-occurence matrix*) $\boldsymbol{X}$ je strukura, která zachycuje právě tyto informace. Slova obsažená ve slovníku jsou mapována na jejich frekvenci společných výskytů. V této matici tedy prvek $X_{i,j}$ označuje počet případů, kdy se slovo w_i vyskytovalo v kontextu slova w_j. Abychom vůbec mohli pracovat s touto řídkou maticí je potřeba využít redukci dimenzionality, konkrétně v tomto případě autoři využívají různé varianty *Singular Value Decomposition (SVD)*, což je technika, o které se podrobněji zmíníme v sekci 5.2.

Pro učení tohoto modelu je navržena následující účelová funkce[9].

$$J = \sum_{i,j=1}^{M} f(\boldsymbol{X}_{i,j})(\boldsymbol{v}_{w_i}^{\mathsf{T}} \boldsymbol{v}_{w_j} - \log \boldsymbol{X}_{i,j})^2 \tag{4.20}$$

V této rovnici představuje f váhovací funkci, která je zavedena pro práci s velmi řídkými spoluvýskyty slov. Ty při trénování modelu nepřináší žádnou informaci a působí jako nadbytečný šum, který zhoršuje výsledky modelu. Taková funkce je pak definovaná následujícím vztahem, kde α a $x_{\max}$ jsou volitelné parametry. Autoři zaznamenali nejlepší výsledky při zvolených parametrech $\alpha = \frac{3}{4}$ a $x_{\max} = 100$

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{pokud } x < x_{\max} \\ 1 & \text{jinak} \end{cases} \tag{4.21}$$

Výsledkem tréninku tohoto modelu jsou vektory, jejichž skalární součin dává dohromady logaritmus pravděpodobnosti spoluvýskytu daných slov. Ve finále je díky tomu tento model schopen poradit si s podobnými sémantickými závislostmi jako Word2Vec. [24, 25]

[9] Pro účely tohoto textu zjednodušena o bias složky a poupravena pro zde používané značení. Pro původní verzi mohu čtenáře odkázat na [24].

Kapitola 5

Vícejazyčné použití IR systémů a word embeddingů

5.1 Cross-Language Information Retrieval

Obecně se **Cross-Language Information Retrieval** (*CLIR*) modely snaží rozšířit vyhledávací schopnosti IR systémů o možnost překlenout jazykovou bariéru mezi dokumenty – tedy o vyhledávání v dokumentech, které jsou psané v jiném jazyce, než je jazyk dotazu. Tyto modely většinou naráží na velkou nejasnost překladů a jejich různé interpretace v rozličných kontextech. V systémech, které se potýkají s vícejazyčnými překlady můžeme narazit na různé situace.

- Corpus je jednojazyčný a uživatel se může dotazovat ve více jazycích.
- Corpus je vícejazyčný a uživatel se může dotazovat ve více jazycích.
- Jednotlivé dokumenty corpu obsahují slova z různých jazyků.

Všechny tři uvedené případy pro své řešení potřebují využití určitého *CLIR* systému. Systémům, které zahrnují všechny tyto případy, a navíc umožňují vyhledávání uvnitř jednoho jazyka, se říká *Multilingual Information Retrieval (MLIR)* systémy. [5]

5.1.1 Překladové strategie

Pro překlad můžeme využít různé strategie. První myšlenkou je přeložit **dotaz** do jazyka dokumentu. Tento způsob způsobuje zpomalení při porovnávací fázi. Zároveň vzhledem k tomu, že dotazy jsou většinou velmi krátké, nemůžeme bez dostatečného kontextu zaručit kvalitní překlad.

Místo překladu dotazu můžeme přeložit samotné **dokumenty**. V tomto případě si musíme držet kolekci duplicitně pro každý jazyk, což může být náročné na paměť. Na druhou stranu se všechny překlady provádí v offline indexační fázi a z tohoto důvodu nemají vliv na latenci ve fázi porovnávání a vyhodnocení dotazu. Díky širšímu kontextu v dokumentech zároveň máme k dispozici kvalitnější překlady, které jsou ale v rámci zachování konzistence a zlepšování systému potřeba periodicky znovu překládat. Třetí alternativou je využití určitého **mezi-jazyka**, do kterého přeložíme jak dotaz, tak dokumenty.

Další možností, jak přistupovat k překladům je ten, že ve výsledku žádný překlad neprovedeme. Tento způsob využívá takzvané **vzájemné srozumitelnosti** (anglicky *mutual intelligibility*), kdy je člověk schopný rozumět jazyku, který se nikdy neučil. Toto lze aplikovat zejména u jazyků, které mají podobnou slovní zásobu, jako například čeština a slovenština. Za tímto

účelem se v průběhu porovnávací fáze využívají různé morfologické charakteristiky daných slov – například se porovnávají jejich podčásti nebo n-gramy. Alternativně se využívají určité psedo korektury, kdy se na druhý jazyk pohlíží jako na špatně napsaný původní jazyk. Ve výsledku tedy i v tomto případě stejně používáme určitou formu meta-překladů. [5]

5.1.2 Způsoby překladu

Samotné způsoby, jakými překládáme mohou být různé. Nejjednodušší variantou je využití strojově čitelných, člověkem vytvořených **slovníků**. V těchto slovnících je každý term nahrazen všemi možnými variantami, jak by mohl být přeložen. Postihnout ale všechny tyto možnosti je poměrně nejasné a takový seznam může postihovat zbytečně mnoho nebo naopak málo příkladů v závislosti na kontextu. Problém zde navíc nastává v momentě, kdy chceme překládat více slov. V takovém případě poskytuje určité zlepšení využití pseudo dotazovacího jazyka, ve kterém definujeme alternativní překlady pro jednotlivé části dotazu. V zásadě ale není prakticky možné postihnout všechny možné varianty překladu daného slova, které se mohou v různých textech vyskytnout a techniky založené na slovnících naráží na problémy s efektivitou.

Místo slovníků můžeme využít **statistický přístup**. Ten poté řeší problémy s nejasnostmi, dvojsmysly i s překladem více slov v dotazu. Tento přístup zahrnuje slovníky obohacené o statistické informace z kolekce – například frekvence výskytů v jednotlivých dokumentech – nebo přímé techniky, které transformují vektorovou reprezentaci slova do prostoru druhého jazyka.

Typickými představiteli statistického přístupu jsou *Latent Semantic Indexing (LSI)* a *Similarity Thesaurus*. **Similarity Thesaurus** mění proces porovnávaní – namísto dokumentů získáváme termy, které jsou v daném jazyce nejpodobnější. Většinou využívá váhovací schéma tf-idf. Samotná konstrukce slovníku pak probíhá velmi podobně jako u *LSI*, které bude blíže popsáno v sekci 5.2.

Tyto přístupy s výhodou využívají **paralelních textů**, což jsou kvalitně ručně přeložené texty ve více jazycích. Příklady takových textů jsou například články a nařízení různých mezinárodních organizací, jako je Evropská Unie nebo Organizace spojených národů. Tyto texty, většinou právnické povahy, bývají přeloženy velmi doslovně. A to hlavně za účelem prevence špatné interpretace. Důležité je ale brát v potaz, že tyto texty většinou postihují určitou specifickou doménu a v momentě, kdy potřebujeme přistoupit ke slovům, které s ní nesouvisí, se překlady mohou lišit a nemusí být natolik kvalitní. Většinou se u nich můžeme spolehnout na shodu až na úrovni odstavců, občas dokonce vět.

Pro domény, které nemají své paralelní texty můžeme využít alespoň **komparativní texty**. Příkladem může být Wikipedia corpus, který zahrnuje všechny přeložené články ze stránky `https://en.wikipedia.org/`. V těchto textech můžeme najít části postihující daný okruh témat ve více jazycích s určitými odkazy mezi sebou. Můžeme z nich využívat podčásti, které jsou mezi sebou propojené, ale nemůžeme se u nich spolehnout na přesnou shodu například na úrovni odstavců, jako tomu bylo u paralelních textů – jejich použití tedy ve výsledku není natolik robustní a je poměrně náchylné na chyby.

Posledním způsobem, jaký můžeme pro překlady použít, je speciální **systém** navržený pro překlad (*machine translation*). Ten může sice obecně dávat poměrně kvalitní překlady, ale u dotazů je problém, že dostáváme krátké a občas dokonce gramaticky nekorektní výrazy. Bez hlubšího kontextu a informacích o dané kolekci tedy nemusí být překlad vrácený daným strojem příliš relevantní. V tomto případě je navíc problémem fakt, že to, co v tomto případě děláme je nahrazení původního dotazu dotazem v jiném jazyce, což ale ve spoustě případů vůbec nemusí posloužit našim účelům. [5]

5.2 Latent Semantic Indexing

Technika jménem **Latent Semantic Indexing (LSI)** se snaží zabránit problémům klasických IR systémů podobně jako tomu bylo u word embeddingů. Využívá statistiky namísto přesné shody slov a snaží se najít určitou strukturu skrytou v jednotlivých slovech.

LSI využívá redukci dimenzionality – konkrétně pak techniku zvanou **Singular Value Decomposition**. Sestavuje *term-document matici*[1] a redukuje jí na matici v latentním prostoru menší dimenze. Vezměme si kolekci dokumentů $\mathcal{D}$ ($|\mathcal{D}| = d$) a slovník $\mathcal{T}$ ($|\mathcal{T}| = t$). Pro *TDM* matici $\boldsymbol{A} \in \mathbb{R}^{d,t}$ o m dimenzích provádí tato technika projekci do prostoru dimenze k, což je volitelný parametr, většinou nastavovaný kolem 100 – 150. Výsledkem této projekce je pak matice $\hat{\boldsymbol{A}} \in \mathbb{R}^{d,k}$, kterou se snažíme vytvořit co nejvíce podobnou původní matici v prostoru větší dimenze. Toho můžeme dosáhnou pomocí metody nejmenších čtverců, kde minimalizujeme následující výraz pomocí 2-normy[2].

$$\Delta = \|\boldsymbol{A} - \hat{\boldsymbol{A}}\|_2 \tag{5.1}$$

Toto mapování ve výsledku odpovídá dimenzím, které mají nevětší rozptyl podobně jako u techniky *PCA*[3], kterou zde ovšem nemůžeme použít vzhledem k tomu, že nemáme k dispozici čtvercovou matici. Konkrétně pak *SVD* dekomponuje matici $\boldsymbol{A}$ na tři dílčí matice následujícím způsobem.

$$\boldsymbol{A}_{t,d} = \boldsymbol{T}_{t,n} \cdot \boldsymbol{S}_{n,n} \cdot (\boldsymbol{D}_{d,n})^\intercal \tag{5.2}$$

V této rovnici značí $n = \min(d, t)$ a matice $\boldsymbol{T}$ a $\boldsymbol{D}$ reprezentují termy a dokumenty v novém prostoru. $\boldsymbol{S}$ reprezentuje singulární hodnoty z matice $\boldsymbol{A}$ seřazené sestupně na diagonále $(\sigma_1, ..., \sigma_n)$, kde i-tá hodnota odpovídá rozptylu na i-té ose. Vybráním prvních $k < n$ sloupců z jednotlivých matic dostaneme následující součin.

$$\hat{\boldsymbol{A}}_{t,k} = \boldsymbol{T}_{t,k} \cdot \boldsymbol{S}_{k,k} (\boldsymbol{D}_{d,k})^\intercal \tag{5.3}$$

V této rovnici pak $\hat{\boldsymbol{A}}$ reprezentuje hledanou matici v nižší dimenzi. Toto řešení je zároveň nejlepším řešením rovnice 5.1. Z důvodu výběru os s největším rozptylem, se SVD zbavuje nechtěného šumu, který zahlcuje klasické modely. Podobně jako tomu bylo u word embeddingů, pomocí této techniky docílíme toho, že slova, která se vyskytují pospolu, budou namapována blízko u sebe.

Pro účely IR systémů je potřeba do tohoto prostoru namapovat i dotazy poskytnuté uživatelem. Pro vektor dotazu $\boldsymbol{q}$ – vzniklým vynásobením vahami jednotlivých slov – toho docílíme tak, že ho vynásobíme následujícím způsobem.

$$\hat{\boldsymbol{q}} = \boldsymbol{q}^\intercal \cdot \boldsymbol{T}_{t,k} \cdot \boldsymbol{S}_{k,k}^{-1} \tag{5.4}$$

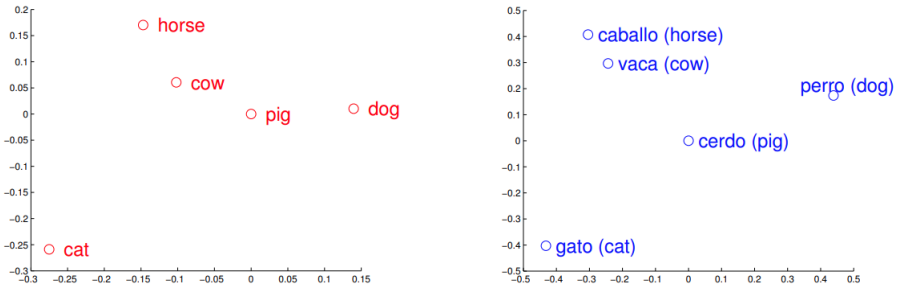
Výsledný vektor v latentním prostoru pak můžeme porovnávat s jednotlivými vektory dokumentů pomocí kosínové vzdálenosti. [26]

Aplikace této metody může být jak v klasických jednojazyčných IR systémech, tak i v CLIR systémech. V druhém případě hovoříme zkráceně o **CL-LSI**. Postupujeme zde podobně na dokumentech v jazycích, které máme k dispozici. Výsledná reprezentace v prostoru menší dimenze bude obsahovat slova z obou jazyků. Ta budou mít v momentě, kdy jsou vždy spojována spolu identickou reprezentaci a v případě, že se vyskytují společně alespoň ve většině případů, jejich reprezentace bude také podobná. [26, 27]

[1] Viz rovnice 3.7.
[2] Viz k-normy vyjádřené rovnicí 3.10.
[3] *Principal Component Analysis* je technika redukce dimenzionality pracující s vlastními čísly a projekcí do ortonormální báze.

5.3 Word2Vec pro vícejazyčné překlady

V této sekci je na místě uvést schopnosti modelu **Word2Vec** při práci s překlady. Ve své navazující práci *Exploiting Similarities among Languages for Machine Translation*[19] se Mikolov a kolektiv zabývají možnostmi, jak naučit model Word2Vec nad jednojazyčnými kolekcemi, čítajícími miliony slov, a mezi nimi vytvořit projekci, která bude fungovat jako „překladač". Při učení Word2Vec modelů – ať už je to Skip-Gram nebo CBOW – tvoříme vektorový prostor, ve kterém jsou podobná slova blízko u sebe. Tento prostor navíc zachycuje sémantické vztahy mezi nimi a můžeme si všimnout, že tyto závislosti budou velmi podobné v různých jazycích, neboť slova, která se dotýkají určitého tématu, mají tendenci být ve stejném kontextu. Z toho důvodu budou mít vektorové prostory různých jazyků podobné geometrické uspořádání. Takové chování se autoři článku snažili ukázat obrázkem 5.1, který zachycuje vektory v prostoru anglického a španělského jazyka, které jsou pomocí PCA redukovány na dvě dimenze. [19]



Obrázek 5.1 Vizualizace vektorové reprezentace vybraných slov v angličtině a ve španělštině pomocí PCA, obrázek převzat z [19].

Na základě této myšlenky hledáme překladovou matici, která je řešením problému optimalizace daným pro vektorové reprezentace slova ve dvou jazycích $\{\boldsymbol{x}_i, \boldsymbol{z}_i\}_{i=1}^{n}$ následovně. Samotná optimalizace je pak řešená gradientním sestupem.

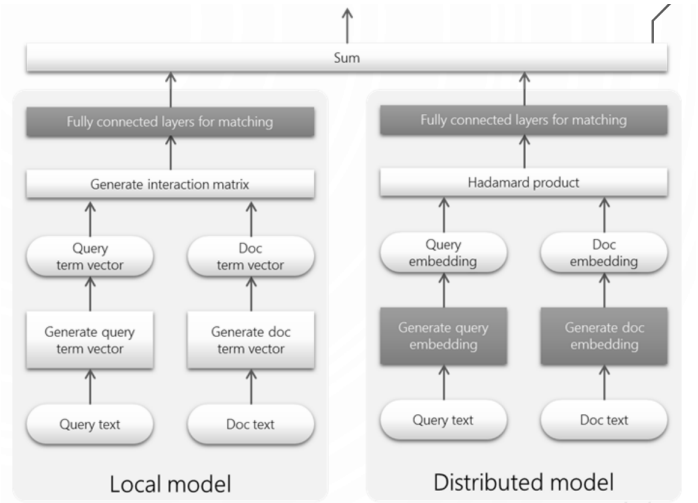
$$\min_{\boldsymbol{W}} \sum_{i=1}^{n} \|\boldsymbol{W}\boldsymbol{x}_i - \boldsymbol{z}_i\|^2 \tag{5.5}$$

Tato matice poté slouží pro překlad slova z vektorového prostoru jednoho jazyka do toho druhého. I přes svojí poměrnou jednoduchost měla tato technika velmi dobré výsledky. Pro slova z angličtiny a španělštiny dosahoval tento model P@5 až 90%. Pro formulování těchto experimentů používali autoři následující postup. Nejprve vytvořili vektory termů, které na jednotlivých indexech obsahovali počty slov. Pomocí slovníku tyto termy namapovali na druhý jazyk a pro každé testované slovo poté hledali nejpodobnější vektor ve výsledném jazyce.

Pro některé jazyky je navíc relevantní tuto transformaci vylepšovat pomocí editační vzdálenosti řetězců – například u jazyků, jako angličtina a španělština. Naopak u jazyků, jako je angličtina a čeština toto vylepšení nemá relevantní dopad. [19]

5.4 Duet

Tato architektura, předvedená v článku *Learning to Match using Local and Distributed Representations of Text for Web Search* [28], spojuje vlastnosti klasických IR systémů a word embedding modelů. Jedná se o spojení dvou dílčích hlubokých neuronových sítí – lokální a distribuovaný model – z jejichž výstupů se pak skládá finální reprezentace. Z důvodu spojení dvou přístupů, kdy obě části modelu spolupracují za dosažením stejného cíle, je tento model autory označován jako **Duet**. V článku je tento model představený přímo pro účely CLIR systémů.



Obrázek 5.2 Architektura modelu Duet, obrázek převzat z `https://www.microsoft.com/en-us/research/uploads/prod/2018/04/NeuralIR-Nov2017.pdf`.

Lokální model zachycuje vlastnosti slov, které většinou postihují klasické IR modely – tedy zejména přesnou shodu s termy v dokumentu nebo porovnávání na základě určitých vzorů. Každý term reprezentujeme jako *one-hot encoding* v m-dimenzionálním prostoru. Pro dotaz q složený z n_q termů pak dostáváme matici $\boldsymbol{Q} = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{n_q})$, obdobně pak pro dokument $d \in \mathcal{D}$ s n_d termy dostáváme matici $\boldsymbol{D} = (\boldsymbol{d}_1, ..., \boldsymbol{d}_{n_d})$. Tento model poté na základě těchto matic generuje tzv. *matici interakce* následovně.

$$\boldsymbol{X} = \boldsymbol{D}^\intercal \boldsymbol{Q} \in \mathbb{R}^{n_d, n_q} \tag{5.6}$$

Tato matice nezachycuje žádnou informaci o termech jako takových, pouze přesné shody mezi dotazem a dokumentem a jejich pozice. Takto zformovaná matice prochází přes různé konvoluční a plně propojené neuronové sítě. Zjednodušená architektura obou částí modelu je znázorněna obrázkem 5.2.

Druhou částí modelu je **distribuovaný model**. Ten se snaží simulovat chování modelů jako je Word2Vec nebo LSI, které ve své podstatě staví na distribuční struktuře textu. Tento podmodel se tedy učí zhuštěnou vektorovou reprezentaci daného slova, která má nižší dimenzi než ta, kterou jsme využívali u lokálního modelu. Místo one-hot encodingu tato metoda využívá n-gramů obsažených v jednotlivých slovech. N-gramy získané z dokumentu a dotazu pak prochází konvolučními a plně propojenými neuronovými sítěmi, které mají za cíl vytvořit hustší vektorovu reprezentaci. Výsledek těchto dvou matic, které mají ve výsledku stejné dimenze, je poté spojen pomocí násobení matic po složkách v jednu matici. Následně i tato matice prochází plně propojenými vrstvami, až se dostaneme k finálnímu skóre. Celý proces je poměrně komplexní a v rámci pouze povrchového výzkumu zde není dopodrobna představen.

Spojením těchto dvou podmodelů získáváme skóre, které zachycuje jak lokální, tak distribuovanou strukturu textu. Při označení lokálního modelu jako f_l a distribuovaného jako f_d dostáváme výsledné skóre jako součet těchto dvou modelů.

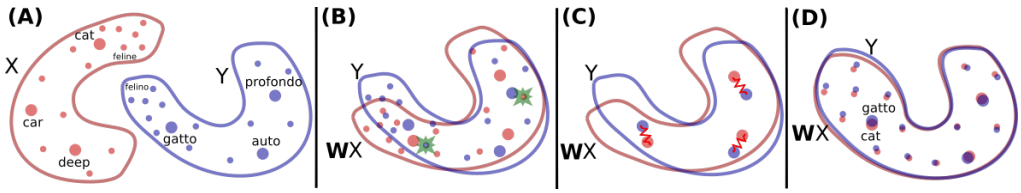
$$f(\boldsymbol{Q}, \boldsymbol{D}) = f_l(\boldsymbol{Q}, \boldsymbol{D}) + f_d(\boldsymbol{Q}, \boldsymbol{D}) \tag{5.7}$$

5.5 MUSE

MUSE je zkratka pro techniku *Multilingual Unsupervised and Supervised Embeddings*. Byla představena v práci *Word Translation Without Parallel Data*[4] [29]. Tato práce navazuje na techniku popsanou v sekci 5.3 a vylepšuje ji. Tedy stejně jako u vícejazyčného Word2Vec modelu i zde předpokládáme, že máme naučené reprezentace na jednojazyčných kolekcích. Navíc můžeme předpokládat, že tyto reprezentace nejsou žádným způsobem závislé na sobě – tedy pro naučení tohoto modelu nemusíme mít k dispozici paralelní texty. Toto je poměrně výhodný předpoklad, vzhledem k tomu, že v některých doménách takové texty nemusí být k dispozici. Navzdory tomuto ztížení je model schopný podávat podobné, ne-li lepší výsledky, než doposud představené modely pracující s paralelními texty.

Předpokládejme, že máme naučené vektorové reprezentace jednotlivých slov ve zdrojovém a cílovém jazyce – označme si je $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ a $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$. Model se ve výsledku skládá ze dvou částí. **Discriminator** se učí rozlišovat zdrojové reprezentace od cílových – tedy rozlišuje mezi náhodně vybranými vzorky z $\boldsymbol{W}\mathcal{X} = \{\boldsymbol{W}\boldsymbol{x}_1, ..., \boldsymbol{W}\boldsymbol{x}_n\}$ a $\mathcal{Y}$. Druhou částí je **mapovací matice** $\boldsymbol{W}$, která se učí spolu s první částí a jejím cílem je zabránit jí provádět přesné predikce. Toho dociluje tím, že dělá $\boldsymbol{W}\mathcal{X}$ a $\mathcal{Y}$ co možná nejpodobnější a vytváří tedy téměř přesné mapování distribucí obou jazyků. Ve výsledku se jedná o hru dvou hráčů – *discriminator* maximalizuje schopnost identifikovat zdroj reprezentace, zatímco *mapovací matice* se mu v tom snaží zabránit.

Pomocí právě naučené matice $\boldsymbol{W}$ je vytvořen slovník, který ale stále nedává nejlepší výsledky. Vzhledem k tomu, že některá slova, která se nevyskytují v kolekci tak často nejsou aktualizována dostatečně a ve výsledku mapování není přesné. Proto se využívají vysoce frekventovaná slova jako kotvy, které spojují vektorové prostory. Následně v okolí těchto „kotev“ vezmeme nejbližší sousedy, abychom zajistili co nejkvalitnější slovník. V poslední fázi provedeme minimalizaci výrazu podobného, jaký předvedl Mikolov a kolektiv (viz rovnice 5.5).



Obrázek 5.3 Názorná ukázka fungování modelu MUSE, obrázek převzat z `https://github.com/facebookresearch/MUSE/blob/main/outline_all.png`.

Grafické znázornění fungování tohoto modelu je viditelné na obrázku 5.3. V části **(A)** zde máme dva vektorové prostory vizualizované ve dvou dimenzích pomocí PCA – červený prostor zde vyjadřuje reprezentace anglických slov a modrý reprezentace španělských slov. Každý bod zde vyjadřuje jedno slovo v prostoru. Můžeme si povšimnout, že geometrické uspořádání těchto prostorů je velmi podobné, jak už bylo zmíněno v sekci 5.3. V **(B)** vidíme, že pomocí naučené matice $\boldsymbol{W}$ můžeme prostor $\mathcal{X}$ natočit tak, aby zhruba odpovídal prostoru $\mathcal{Y}$. Ve fázi **(C)** vidíme využití kotev, které dále vylaďují překladovou matici a finální mapování je vidět v části **(D)**.

[4] Tu doplňuje github repozitář s praktickou implementací na stránkách `https://github.com/facebookresearch/MUSE`.

Část II

Praktická část

Kapitola 6
Tvorba corpu

V rámci této práce se zpracovávaly dvě kolekce dokumentů. První z nich byla převzata z volně dostupného zdroje a upravena pro účely této práce (*ÚFAL*). Druhá kolekce, která byla podstatně objemnější byla vytvořena na základě databáze nařízení a vyhlášek Evropské Unie, volně dostupné v několika jazycích na stránkách `https://eur-lex.europa.eu/`.

K práci s daty byl zvolen programovací jazyk *Python*, který umožňuje díky svým poměrně robustním knihovnám jednoduchou a rychlou práci s většími kolekcemi. Navíc byla v tomto jazyce možnost využít knihovny uzpůsobené pro práci s přirozeným jazykem, které usnadňují implementaci – konkrétně knihovna *NLTK*[1], která obsahuje funkce pro stemming, tokenizaci a další techniky, které byly rozebírány v kapitole 2. Později pak pro samotnou práci s word embedding modely byla využívaná knihovna *Gensim*[2], která obsahuje implementaci většiny modelů, které byly doposud představeny – jako například Word2Vec, FastText nebo LSI.

6.1 Procesy pro předzpracování přirozeného jazyka

V rámci předzpracování textu byl zvolen jednotný způsob pro obě kolekce (až na určité speciální využití regulárních výrazů zmíněných u EUR-Lex kolekce). V zásadě jsou postupně procházeny všechny texty pomocí třech dílčích kroků – všechny byly již představeny a vysvětleny v kapitole 2. V první části probíhá samotná **tokenizace** po slovech. Pro tyto účely byla zvolena metoda `word_tokenize` z knihovny NLTK, která v základu předpokládá anglická slova a umí si poradit se zkrácenými tvary jako *„aren't“*, které byly diskutovány dříve. Zároveň má volitelný parametr *language*, který podporuje i český jazyk. Vzhledem k tomu, že český jazyk v tomto ohledu nemá výrazná specifika, která by nám ztěžovala tokenizaci je ovšem otázka, nakolik je tento parametr uplatnitelný. Jediný potenciální problém, který byl identifikován daných corpech, bylo použití spojovníků – například ve slově *„bude-li“*. V tomto případě se ale `word_tokenize` i v základu chová tak, jak bychom mohli předpokládat a slovo dále nedělí. Alternativou k `word_tokenize` je pak `sent_tokenize`, který slouží k tokenizaci po větách. Ten byl původně také v rámci této práce využíván k zarovnání kolekce po větách – tento způsob byl však nakonec zavržen vzhledem k tomu, že v rámci uvažovaných kolekcí nedával velký smysl. [30]

Určitý mezikrok pak představuje **normalizace**. V tomto případě se jedná zejména o odstranění čísel a symbolů, které nejsou znaky. Tento krok by mohl být nahrazen namapováním čísel na odpovídající slovní reprezentace, ale vzhledem k tématu obou kolekcí nebyl tento způsob natolik relevantní, aby byl implementován. Zároveň byl v rámci této fáze využit *case-folding*,

[1] Natural Language Toolkit – více informací o knihovně je možné nalézt přímo v dokumentaci na stránkách `https://www.nltk.org/`.

[2] Akronym pro „Generate Similar“. Původně český projekt, oficiální stránky jsou dostupné na adrese `https://radimrehurek.com/gensim/`.

tedy namapování všech slov na varianty s malým počátečním písmenem. Ten byl využit namísto metody *truecasing*, která by byla s ohledem na objem kolekce příliš pomalá.

Druhým krokem je **odstranění stop slov**. V případě angličtiny byla opět možnost využít knihovnu NLTK, která nabízí stažení seznamu stop slov a jeho následné využití. S tímto seznamem byly poté porovnávány jednotlivé tokeny. Ty pak byly vyřazovány na základě jejich výskytů v tomto seznamu. V případě českého jazyka tato knihovna nenabízela seznam stop slov, a proto byl využit jiný zdroj – `https://countwordsfree.com/stopwords/czech` – ze kterého byl tento seznam stažen. Až na slova v seznamu byl postup zde identický pro oba jazyky.

Poslední částí předzpracování je **stemming**. Ten byl vybrán oproti lemmatizaci, protože bylo v plánu pracovat s velkou kolekcí dokumentů a stemming je v tomto případě vhodnější variantou, vzhledem ke své rychlosti a jednoduchosti. Pro angličtinu posloužil už jednou zmiňovaný *Porterův* stemmovací algoritmus, který byl opět k dispozici v knihovně NLTK. Pro češtinu ovšem nebyl takto standardní algoritmus v knihovně, a proto byl využitý jiný, implementovaný v jazyce Python ze zdroje `https://research.variancia.com/czech_stemmer/`.

6.2 Techniky vytváření trénovacích dat

Finálním krokem v obou datasetech bylo vytvoření **trénovacích dat** pro vícejazyčný Word2Vec model. K tomu byly využívány dva přístupy. První z nich – dále označovaný jako „**jednoduché spojení**“ – spojuje české a anglické verze bez dalšího zpracování za sebe. Tedy v momentě, kdy máme dokument pro každý jazyk[3]:

- **Česká verze**: Datová sada pro plně trénovatelné generátory jazyka v hlasových dialogových systémech [...] byla použita metoda crowdsourcingu.
- **Anglická verze**: A dataset intended for fully trainable natural language generation (NLG) systems [...] Crowdsourcing has been used to obtain natural context user utterances as well as natural system responses to be generated.

Dostali bychom nejprve pomocí předzpracovacích procesů tokeny, které reprezentujeme formou sekvencí. Tyto sekvence bychom na sebe poté jednoduše napojili tak, jak můžeme vidět v tabulce 6.1.

	Sekvence tokenů
České tokeny	datov, sad, pln, ..., použit, metod, crowdsourcing
Anglické tokeny	dataset, intend, fulli, ..., system, respons, gener
Jednoduché spojení tokenů	datov, sad, pln, ..., použit, metod, crowdsourcing, dataset, intend, fulli, ..., system, respons, gener

Tabulka 6.1 Jednoduché spojení tokenů dvou jazyků, červeně jsou označený tokeny druhého jazyka.

Problém s tímto přístupem nastává v momentě, kdy chceme trénovat Word2Vec model – nemáme možnost, jak měnit velikost okna kontextu, protože jednotlivé věty mohou být různě velké. Navíc chceme do kontextu dostat druhý jazyk, a proto musíme v podstatě délkou okna obsáhnout minimálně délku věty – v předchozím případě bychom například chtěli, aby ve stejném kontextu skončily tokeny *„datov“* a *„dataset“*, které mají mezi sebou všechny tokeny z prvního jazyka. Možnost, jak tomuto předejít je nastavit kontext dostatečně velký. Tímto způsobem se ale do kontextu dostává zbytečné množství slov navíc, které mohou narušovat kvalitu výsledných reprezentací.

Alternativou k tomuto přístupu je **sekvenční prokládání tokenů**. Zde jsou vytvářeny nové věty tak, že je vždy vybrána delší z obou sekvencí. Střídavě jsou pak vybírány tokeny z této delší

[3] Jako příklad je zde uveden jeden z dokumentů obsažený v ÚFAL kolekci.

sekvence a ze sekvence druhého jazyka. Vzhledem k tomu, že druhá sekvence je kratší, v určitém momentě nastane situace, kdy skončí – zbytek výsledné sekvence je pak doplněn tou delší. Aby bylo dosaženo všech možných kontextů, postupně je posouván moment, kdy se začnou vybírat tokeny z druhé sekvence. Poslední věta, která je tímto způsobem vygenerována pak na začátku obsahuje tokeny z delší sekvence a až do konce se poté střídají tokeny z delší a kratší sekvence. Výsledkem pak je více vět v závislosti na rozdílu délky obou sekvencí. Pro předchozí příklad bychom dostali věty, které jsou pro jednotlivé indexy znázorněné v tabulce 6.2.

Index	Proložené tokeny
1	datov, dataset, sad, indend, pln, ..., metod, crowdsourcing
2	datov, sad, dataset, pln, intend ..., metod, crowdsourcing
⋮	⋮
$n-2$	datov, sad, ..., použit, respons, metod, gener, crowdsourcing
$n-1$	datov, sad, ..., použit, gener, metod, crowdsourcing

Tabulka 6.2 Sekvenční prokládání slov, červeně jsou označená tokeny z anglické části, n zde značí počet tokenů v kratší ze sekvencí.[4]

6.3 ÚFAL abstrakty

První kolekcí jsou zpracované abstrakty vědeckých prací vytvořených na Ústavu formální a aplikované lingvistiky na Matematicko-Fyzikální Fakultě Univerzity Karlovy (dále označované jako **ÚFAL**). Nejlépe můžeme tento corpus popsat citováním jeho autorů. „*This is a document-aligned parallel corpus of English and Czech abstracts of scientific papers published by authors from the Institute of Formal and Applied Linguistics, Charles University in Prague, as reported in the institute's system Biblio. For each publication, the authors are obliged to provide both the original abstract in Czech or English, and its translation into English or Czech, respectively. No filtering was performed, except for removing entries missing the Czech or English abstract, and replacing newline and tabulator characters by spaces.*“ [31]

Na stránkách Lindat[5] je volně dostupná část těchto záznamů ke stažení. Tento corpus byl zvolen, vzhledem k poměrně malému objemu slov, pro cvičné účely. Je zarovnaný na úrovni dokumentů, což k účelům této práce víceméně stačí. Původní myšlenka byla určitým způsobem docílit navíc i zarovnání na úrovni odstavců, posléze vět. Myšlenka zarovnat tuto kolekci na úrovni odstavců byla zavržena už jen z důvodu, že kolekce má již předzpracované a spojené odstavce, a navíc abstrakty vědeckých prací ve valné většině případů nepřesahují co do velikosti několik málo odstavců. Zarovnání na úrovni vět by se mohlo zdát jako poměrně lákavá myšlenka, která by mohla vést k lepší efektivitě modelů. Nicméně v obou jazycích jsou využívány různé zkratky a různá místa, kam se dávají interpunkce. Tento fakt je zřetelný zejména u druhé kolekce, kde se často vyskytovala slova jako v angličtině „*Article*“ a jeho český protiklad „*čl.*“. Tato a další slova způsobila, že zarovnání na úrovni vět nebylo jednoduše proveditelné.

Vstupní soubor byl ve formě jednoduché TSV[6] tabulky, kde každý záznam obsahuje anglickou a českou verzi jednoho abstraktu. Pro účely dalšího zpracování byly tyto abstrakty rozděleny na jednotlivé tokeny tak, jak bylo popsáno v předchozí sekci a uloženy spolu s původní tabulkou obsahující plné znění textu. Výsledná tabulka, kterou poté bylo možné použít pro potřeby vyhledávače byla tedy ve formátu (česká verze, anglická verze, české tokeny, anglické tokeny).

Při zpracování bylo zjištěno, že kolekce čítala 241 abstraktů pro oba jazyky, přičemž výsledný počet tokenů získaný z této kolekce byl 16 422 pro český jazyk a 10 539 pro anglický.

[5] Viz tehttps://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1731.

[6] Tab-separated values, datový formát pro ukládání tabulek využívající jako oddělovač tabulátor namísto čárky jako u CSV.

6.4 EUR-Lex

Stránka `https://eur-lex.europa.eu/` zahrnuje veřejné právní publikace Evropské Unie, které jsou k dispozici vždy v několika jazycích. Na tomto webu jsou pravidelně zveřejňovány dokumenty Úředního věstníku, které obsahují nařízení, smlouvy, mezinárodní dohody a právní předpisy. Většina těchto dokumentů je k dispozici ve formátech PDF a HTML.

Tato stránka byla použita pro stažení jednotlivých dokumentů pro vytvoření vstupní kolekce. Jednotlivé HTML stránky byly opět stahovány a zpracovány pomocí jazyka Python – konkrétně byly využívány knihovny *BeatifulSoup* pro zpracování HTML značek a *Requests* pro stahování obsahu ze stránek.

Adresy jednotlivých stránek mají určitý vzor, na základě kterého je lze automaticky stahovat. Konkrétně byl využíván následující vzor stránek: `https://eur-lex.europa.eu/legal-content/CS/TXT/HTML/?uri=OJ:C:YYYY:XXX:FULL`, kde `YYYY` mohlo být doplněno rokem vydání a `XXX` číslem v kolekci. Tímto způsobem byly poté stahovány stránky z let 2010 – 2020 s tím, že v každém roce se pracovalo s dokumenty z rozsahu 1 – 450. Neexistující dokumenty byly pak jednoduše přeskakovány. Tímto způsobem se poměrně efektivně dosáhlo získání relativně velkého objemu dat. Jediný problém z hlediska obsahu byla přítomnost jiných jazyků než angličtiny a češtiny. To bylo způsobené obsahem různých místních vyhlášek a mezinárodních smluv s konkrétními zeměmi. Na druhou stranu tato slova nebyla natolik frekventovaná, aby výrazně ovlivnila vektorové reprezentace jazyků primárního zájmu.

Z hlediska HTML struktury, stránky většinou nevykazovaly přílišný sémantický význam – například zde nebyly využívány specifické tagy pro číslované a řazené seznamy, což způsobovalo problémy při zpracování. Ty byly řešeny aplikací několika regulárních výrazů, které měli za cíl zbavit se všech typů seznamů (konkrétně se jednalo například o použití arabských a římských číslic, případně písmen). Formát těchto seznamů navíc nebyl jednotný napříč dokumenty a toto bylo také nutné brát v potaz. Jednotlivé odstavce článků pak byly alespoň děleny pomocí `<p>` tagů[7]. Zbylé informace z HTML struktury mimo tyto značky byly zanedbány, vzhledem k tomu, že se často jednalo pouze o obsah nebo tabulky, které obsahovaly irelevantní symboly.

Snahou zpracování bylo získat paralelní corpus, který bude zarovnaný na úrovni jednotlivých odstavců. K docílení tohoto cíle byly automaticky zanedbávány dokumenty, které neměly stejný počet odstavců, neboť nebylo možné je jednoduše automaticky zarovnat. S jednotlivými odstavci bylo pak pracováno podobně, jako v předchozí kolekci s tím rozdílem, že v rámci předzpracování bylo prováděno očištění od číselných seznamů a irelevantních symbolů, popsané v předchozím odstavci.

Vzhledem k většímu objemu dat a potřebě uchovávat textovou reprezentaci pro později používaný vyhledávač, byla vytvořena struktura složek, kde jedna složka obsahovala informace k jednomu dokumentu. V dané složce byly jednotlivé tokeny pro tento dokument v obou jazycích a také textové reprezentace dokumentu. Z této vytvořené struktury pak bylo možné stahovat jednotlivá data, aniž by bylo nutné znovu stahovat všechny dokumenty ze zdrojových stránek.

Po výsledném zpracování jednotlivých tokenů bylo dosaženo výsledných počtů shrnutých v tabulce 6.3. Pro představu je zde uveden i počet vytvořených vět pomocí sekvenčního prokládání slov oproti počtu odstavců, který byl stejný při použití jednoduchého spojení – můžeme zde vidět, že počet se zvedl zhruba šestinásobně.

[7] Párová HTML značka značící odstavec

	Počet
České tokeny	10 919 671
Anglické tokeny	9 209 719
Unikátní české tokeny	275 868
Unikátní anglické tokeny	261 872
Odstavce	359 628
Sekvenčně proložené odstavce	2 361 556

Tabulka 6.3 Statistika EUR-Lex kolekce

Kapitola 7

Tvorba dvojjazyčného Word2Vec modelu

Jak už bylo naznačeno v úvodu, cílem této práce je vytvořit dvojjazyčný Word2Vec model, který bude natrénovaný na paralelním corpu, jež byl představen v předchozí kapitole. Tento model mapuje vektorové reprezentace slov v prvním jazyce na slova ve druhém jazyce bez použití transformační matice. To je rozdíl oproti jiným přístupům, jako je například Mikolovův vícejazyčný Word2vec nebo MUSE[1]. Zde využíváme přímého zakódování vektoru do stejného prostoru bez jakékoliv potřeby mapování mezi jednotlivými jazyky. Tento fakt vychází z distribuční struktury textu, kdy v trénovacích datech vytváříme kontexty, kde jsou překlady slov blízko sebe.

K implementaci bylo opět využito jazyku Python a knihovny Gensim, která obsahuje implementaci obou Word2Vec modelů včetně optimalizací jako jsou hierarchický softmax a NEG sampling. V rámci práce s Word2Vec modelem bylo vytvořeno rozhraní pro kontinuální trénování, které umožňovalo modely průběžně ukládat a trénovat dávkovým režimem. Důležité parametry při trénování modelu jsou zejména tyto:

- *Velikost okna kontextu*
- *Volba typu modelu* – Na výběr byl Continuous Skip-Gram nebo CBOW.
- *Velikost vektorové reprezentace* – Určuje velikost výstupního vektoru a počet neuroné ve skryté vrstvě.
- *Použitý typ spojení vět* – Použity byly přístupy jednoduchého spojení nebo sekvenčního prokládání, popsané v předchozí kapitole.

7.1 Metriky použité k vyhodnocení úspěšnosti

K vyhodnocení úspěšnosti jednotlivých modelů byly navrženy dvě metriky. Obě stavěly na tom, že jsem vybral tokeny v kolekci, které byly velmi frekventované, v kolekci se vyskytoval jejich překlad ve druhém jazyce a z obou těchto tokenů bylo jednoznačně patrné, z jakého slova vychází. Pro tyto potřeby byl ručně sestaven seznam 30 dvojic – například zde byly přítomny dvojice jako *„access“* a *„přístup“* nebo *„geograph“* a *„zeměpisn“*.

První metrikou, která je poměrně jednodušší a vychází čistě z myšlenky porovnání vzdáleností mezi vektory, je **součet vzdáleností**. Ta byla počítána tak, že pro každý pár byla vypočtena ko-

[1] Viz sekce 5.3 a 5.5.

sinová vzdálenost a ta byla dále normována maximální vzdáleností pro danou dvojici. To můžeme vyjádřit pro vektorové reprezentace slov v jednotlivých jazycích $\boldsymbol{x} \in \mathcal{X}$ a $\boldsymbol{y} \in \mathcal{Y}$ následovně[2].

$$\mathrm{dist}_{\mathrm{sum}} = \sum_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{y}\in\mathcal{Y}} \frac{\mathrm{sim}_{\cos}(\boldsymbol{x},\boldsymbol{y})}{\max_{\boldsymbol{z}\in\mathcal{Y}} \mathrm{sim}_{\cos}(\boldsymbol{x},\boldsymbol{z})} \tag{7.1}$$

Při konstrukci druhé techniky bylo stavěno na metrikách z článků Mikolova [19] a Jansena [32], které upravovaly P@k metriku. Ta byla pro potřeby překladů mírně pozměněna. Konkrétně zde mapovali vektory reprezentující jednotlivá slova a počítali výskyty korektních překladů v daném okně. Pro účely této práce byla tato metrika zjednodušena, vzhledem k tomu, že nemusely být brány v potaz žádné transformace mezi jazyky. Jednoduše tedy stačilo vzít české slovo, vypočítat kosinové vzdálenosti do všech anglických slov a mezi nimi najít k nejbližších. Pokud se mezi těmito slovy vyskytoval překlad, test byl vyhodnocen jako úspěšný. Finální skóre pak bylo spočteno jako poměr úspěšných testů vůči počtu všech testů.

7.2 Výsledky

Vzhledem k poměrně nízkému objemu slov nebyl ÚFAL corpus příliš úspěšný a výsledky z něj zde nejsou ani prezentovány. Naproti tomu tokeny z Eur-LEXu se podařilo dobře namapovat na jejich překlady. V tomto corpu vyvstávaly určité problémy způsobené výskytem slov z jiných jazyků, než které byly předpokládány. Ty sem byly zavlečeny z různých vnitrostátních smluv v rámci EU, které jsou také součástí corpu. Bylo vidět, že například maďarská slova, která byla přítomna jak v české, tak v anglické části byla vyčleněna úplně zvlášť.

Dalším problémem byl samotný výskyt slov z druhého jazyka – tedy například v anglické části byla přítomná česká slova a naopak. Tento fakt působil šum, který ovlivňoval P@k metriku, vzhledem k tomu, že stejná slova se mapovala přesně na sebe. To je poměrně nechtěný fakt pro potřeby překladů, na druhou stranu pro indexaci je to ve výsledku očekávané chování – chceme mít ve výsledcích vyhledávání všechna slova, která jsou relevantní pro daný dotaz.

V rámci testů byly využity již výše popsané parametry. Pro potřeby Eur-LEX byly použity dvě techniky spojování odstavců a dva typy Word2Vec modelů. Zároveň byly vyzkoušeny různé velikosti výstupních reprezentací jednotlivých slov – konkrétně zde byly zvoleny dimenze 50, 100 a 200. Velikosti okna pak byla volena tak, aby v oknu byly vždy přítomny alespoň nějaké překlady daného slova. pro sekvenční prokládání pak byly zvoleny konkrétně hodnoty 4 – 8 a pro jednoduché spojení 50, 75, 100, 125 a 150. Důvod větších oken v případě jednoduchého spojení byl ten, že abychom do kontextu zahrnuli překlad, museli jsme v podstatě obsáhnout celou délku daného odstavce.

Pro test byly zvoleny jako metriky již popisovaná suma vzdáleností, P@5 a P@10. Suma vzdáleností ne úplně vždy odpovídala precission metrice, kdy v některých případech, kdy byla precission velmi nízká, bylo dosaženo téměř nejmenší sumy. Tento problém je přisuzován právě šumovým slovům ze stejného jazyka, které byly v corpu obsaženy.

Nejlepší výsledky pak byly zaznamenány při použití CBOW modelu a sekvenčního prokládání textů, kdy byla pro metriku P@5 naměřená úspěšnost až 90%. Lepší výsledky také poskytovaly menší vektorové reprezentace tedy 50 nebo 100, což je poměrně očekávané chování vzhledem k menšímu objemu dat. Podobně se dá argumentovat pro využití CBOW oproti Skip-Gram modelu, kdy druhý zmíněný většinou potřebuje více dat k tomu, aby poskytoval rozumné výsledky. Konkrétní hodnoty jsou představeny v tabulkách 7.1 a 7.2 pro sekvenční prokládání a poté v tabulkách 7.3 a 7.4 pro jednoduché spojení. V těchto tabulkách jsou znázorňují hodnoty l dimenzionalitu vektorové reprezentace slova a c velikost okna kontextu, ve zbytku sloupců jsou pak naměřené hodnoty pro jednotlivé metriky popsané výše. Ve výsledku si můžeme povšimnout, že sekvenční prokládání mělo poměrně lepší výsledky oproti jednoduchému spojení.

[2]Množiny $\mathcal{X}$ a $\mathcal{Y}$ zde označují tokeny jednotlivých jazyků zredukovaný na výše zmíněný výběr několika málo příkladů.

l	c	dist_{sum}	P@5 [%]	P@10 [%]
50	4	6.80	80	83.33
50	5	6.88	83.33	86.67
50	6	6.79	80	86.67
50	7	6.93	83.33	86.67
50	8	6.76	66.67	90
100	4	9.68	70	76.67
100	5	9.67	90	90
100	6	9.90	80	90
100	7	9.69	86.67	86.67
100	8	9.81	76.67	83.33
200	4	12.59	70	86.67
200	5	12.82	73.33	83.33
200	6	12.88	80	86.67
200	7	13.48	73.33	86.67
200	8	13.37	66.67	80

Tabulka 7.1 Výsledky sekvenčního prokládání pro CBOW model.

l	c	dist_{sum}	P@5 [%]	P@10 [%]
50	4	7.98	33.33	36.67
50	5	7.88	33.33	40
50	6	7.68	43.33	43.33
50	7	8.03	40	43.33
50	8	7.75	43.33	50
100	4	10.08	43.33	56.67
100	5	10.32	43.33	50
100	6	9.75	50	60
100	7	10.00	53.33	56.67
100	8	10.47	43.33	53.33
200	4	12.20	63.33	66.67
200	5	12.32	56.67	63.33
200	6	11.74	70	70
200	7	12.30	60	63.33
200	8	11.76	63.33	73.33

Tabulka 7.2 Výsledky sekvenčního prokládání pro Continuous Skip-Gram model.

Tyto hodnoty jsou samozřejmě výrazně ovlivněny výběrem slov, který nebyl čistě náhodný a byla brána v potaz více frekventovaná slova. Dokonce pro výraznou část slov v corpu by překlady nemusely dávat žádný smysl a vzorek pouze 30 slov nemůže být považován za dostatečně reprezentativní. Na druhou stranu je poměrně patrné, že model vyhovuje očekáváním a případné nesrovnalosti mohou být také dány nedostatečným objemem vstupních dat. Pro porovnání například v [19] Mikolov pracoval s corpem, který čítal 575 milionů anglických a 155 milionů českých slov, což je zhruba desetinásobek objemu corpu používaného v této práci.

7.3 Grafické reprezentace

K tomu, aby bylo možné graficky reprezentovat jednotlivá slova ve dvoudimenzionálním prostoru, bylo potřeba zredukovat dimenzi z původní velikosti na pouze dvě dimenze. K této redukci byly využity dvě technik – PCA a t-SNE. První z nich – **Principal Component Analysis** – využívá

l	c	dist_{sum}	P@5 [%]	P@10 [%]
50	50	6.99	30	33.33
50	75	6.96	40	43.33
50	100	5.26	36.67	43.33
50	125	4.50	33.33	36.67
50	150	5.44	33.33	36.67
100	50	8.12	50	56.67
100	75	7.90	66.67	73.33
100	100	7.87	70	76.67
100	125	7.90	76.67	80
100	150	7.87	73.33	86.67
200	50	8.35	53.33	63.33
200	75	7.91	70	73.33
200	100	7.90	76.67	83.33
200	125	7.97	80	80
200	150	7.60	83.33	90

Tabulka 7.3 Výsledky jednoduchého spojení pro CBOW model.

l	c	dist_{sum}	P@5 [%]	P@10 [%]
50	50	4.77	26.66	36.66
50	75	5.33	40	43.33
50	100	6.95	53.33	60
50	125	7.15	50	66.67
50	150	6.97	56.67	60
100	50	8.37	46.67	50
100	75	8.17	50	50
100	100	8.25	46.67	50
100	125	8.37	50	50
100	150	8.06	46.67	50
200	50	10.07	50	56.67
200	75	9.85	50	60
200	100	9.81	56.67	60
200	125	9.76	56.67	60
200	150	9.91	50	53.33

Tabulka 7.4 Výsledky jednoduchého spojení pro Continuous Skip-Gram model.

ortogonální projekce, kde v nové rovině máme v prvník k dimenzích (komponentách) největší rozptyl v datech a zároveň největší část informace. Tudíž redukcí na 2 dimenze neztrácíme příliš informací o datech. Na druhou stranu využívá pouze lineárních transformací, které nemusí být pro potřeby této práce dostatečné. Jak se také ukázalo, tak dostatečné nebyly, protože projekcí z dimenze 100 až 200 pouze do 2 nebylo možné reprezentovat data tak, aby z nich byl patrný jejich význam. [7]

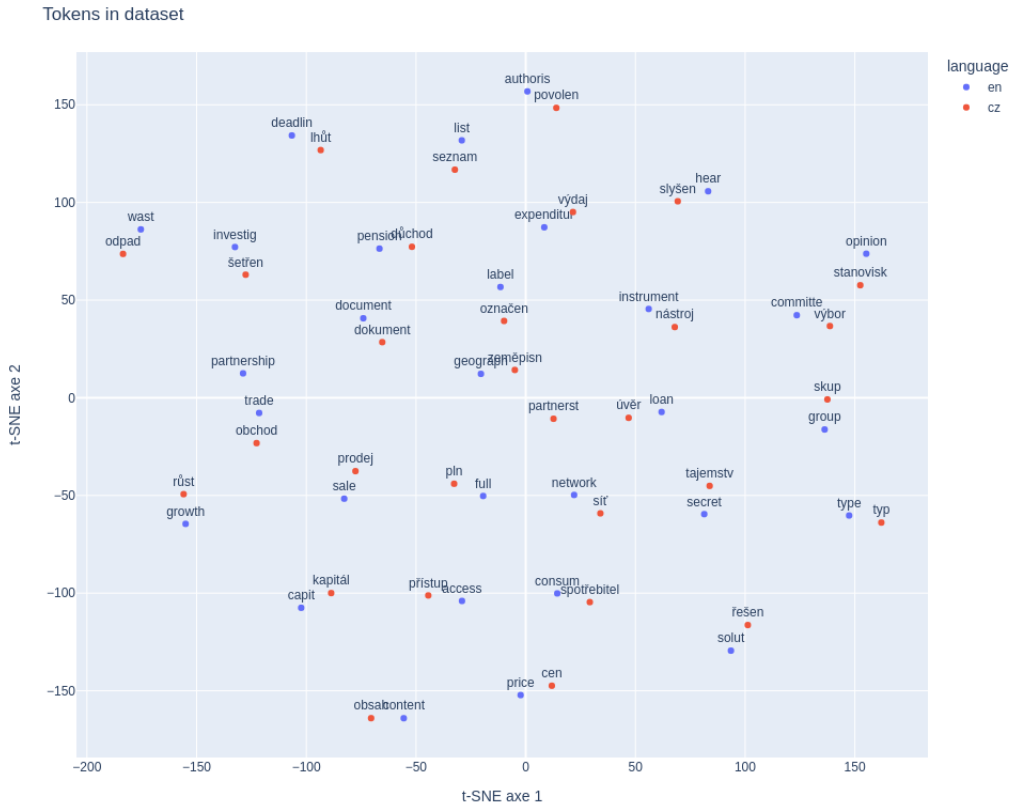
Alternativou k PCA je pak **t-distributed Stochastic Neighbor Embedding** (t-SNE). Tato technika pracuje s nelineárními transformacemi a pravděpodobnostními rozděleními, které lépe zachycují lokální strukturu. V originální verzi počítá s Eukleidovskými vzdálenosti, ale toto chování lze měnit v rámci *scikit-learn*[3] knihovny v Pythonu, která obsahuje implementace daného algoritmu. Konkrétně zde byla využívaná kosinová vzdálenost, která nejlépe reflektuje použité

[3]Obsahuje implementaci často využívaných technik strojového učení, jako jsou například různé regresní a klasifikační modely nebo právě techniky redukce dimenzionality, viz `https://scikit-learn.org/stable/`.

metriky. Důležitým parametrem při práci s t-SNE je *perplexity*. Tato hodnota blíže určuje sousedy a rozptyl v každém kroku. Typicky se pohybuje kolem hodnot 5 až 50. [33, 34]

Právě t-SNE pak vedlo na nejlepší výsledky a bylo také používáno jako hlavní technika redukce dimenzionality v rámci této práce. Samotné vykreslování grafů poté bylo zprostředkování pomocí knihovny *plotly*[4]. Tyto vizualize také umožňovaly částečně optometricky hodnotit úspěšnosti jednotlivých modelů.

Konkrétně byl reprezentován výše zmíněný vzorek slov, na kterém byly počítány jednotlivé metriky. Výsledkem pak byla vizualizace 7.1, ve které je jasně patrné, že překlady jednotlivých dvojic jsou mapovány blízko sebe. Toto bylo také částečným cílem této práce. Pro vizualizaci byly vybrány parametry, které nejlépe odpovídaly P@k metrice – tedy sekvenční prokládání, CBOW model, vektor velikosti 100 a velikost kontextového okna 5.



Obrázek 7.1 Vizualizace dvojic slov ve dvou dimenzích pomocí t-SNE redukce.

7.4 Diskuze

Výhodou tohoto přístupu je to, že můžeme zanedbat výpočetní části, které se vztahují k hledání transformačního zobrazení tak, jak tomu bylo u vícejazyčného Word2Vec modelu nebo MUSE. Tato přímost reprezentací může vést k rychlejšímu procesu učení.

Na druhou stranu poměrně velkou nevýhodou je potřeba existence paralelního corpu, což jak už bylo diskutováno dříve není častý případ. Komparativní corpy většinou obsahují nepoměrně

[4]Knihovna pro práci s vizualizacemi, viz `https://plotly.com/`.

více dat z různých oblastí. Možným zlepšením by bylo vyzkoušet tento přístup (konkrétně sekvenční prokládání) právě na komparativní kolekce dokumentů – například Wikipedia corpus. Jedinou otázkou zůstává nakolik by pak převládaly sémantické souvislosti v daném jazyce nad překladovými vztahy mezi jednotlivými slovy. Už pro případ, který zde byl prezentován bylo z grafické reprezentace patrné, že některé shluky, které se v rámci reprezentací vytvářely, kladly důraz právě na tuto sémantickou podobnost. Na druhou stranu, vzhledem k tomu, že se vždy díváme na druhý jazyk a v případě jiných corpů než je Eur-LEX by nemusela být přítomná slova z druhého jazyka v dané části, by možná stálo za to se touto problematikou dále zaobírat.

Dalším prostorem pro zlepšení je pak zarovnání jednotlivých vektorových reprezentací tak, jako tomu bylo u modelu MUSE s využitím slov, která slouží jako kotvy. S pomocí určité lineární transformace by pak mělo být možné vektorové reprezentace dále zlepšovat.

Kapitola 8

Implementace dvojjazyčného vyhledávače

Finální částí práce je implementace dvojjazyčného webového vyhledávače. Tato část byla implementována v jazyce *Ruby* a rozhraní (*framework*) pro tvorbu webových aplikací *Ruby on Rails*. Tato technologie byla zvolena kvůli autorovým osobním preferencím a s ohledem na poměrně rychlou možnost tvorby kompletní aplikace bez nutnosti využití další technologie pro grafickou část. Navíc byla využita možnost jednoduše oddělit tvorbu vektorových reprezentací vytvořených v Pythonu od samotné indexační a vyhledávací fáze. Jediný problém v tomto přechodu představoval proces stemmování a tokenizace. V průběhu zpracování ovšem bylo zjištěno, že samotné hledání překladů je také zapotřebí implementovat v Pythonu kvůli jeho možnostem rychle a efektivně zpracovávat velké objemy dat, kterými Ruby nedisponuje.

8.1 Architektura aplikace

Samotná aplikace obsahuje jednoduchou databázi v *PostgreSQL*, ve které jsou uloženy jednotlivé textové reprezentace dokumentů v obou jazycích, zpracované tokeny a vytvořený invertovaný index. V aplikaci jsou tedy přítomné dva hlavní modely – *Kolekce* a *Dokument*. V rámci modelu Kolekce jsou uloženy návaznosti na jednotlivé dokumenty a vygenerovaný invertovaný index, který může být díky možnostem PostgreSQL uložen v databázi ve formátu JSON. Navázané dokumenty obsahují identifikátor, textové reprezentace a tokeny. Pro přidání počátečních dat do aplikace na základě předvytvořené kolekce je v rámci aplikace k dispozici Ruby skript.

V této aplikaci jsou k dispozici obě kolekce – jak EUR-Lex, tak ÚFAL – nicméně pro vyhledávání je použitelná pouze první zmíněná kolekce, vzhledem k tomu, že ÚFAL kolekce nebyla v tomto ohledu přínosná.

Invertovaný index odpovídá struktuře, která byla představena dříve – obsahuje tedy jednotlivá slova namapována na jejich výskyty v dokumentech. Díky této struktuře pak bylo možné hledat dokumenty odpovídající daným dotazovaným termům. Pro účely vícejazyčného vyhledávání byl navíc tento index rozdělen na dva jazyky – ve výsledku tedy sestával z anglické a české části. V rámci aplikace je přítomná akce, která umožňuje invertovaný index znovu generovat.

Architektura aplikace je postavena na principu **MVC** (Model-View-Controller), kde jsou jasně oddělené jednotlivé části – tuto architekturu si do značné míry vynucuje samotný framework. **Model** zde představují již zmíněné třídy, které se mapují na databázové tabulky Kolekce a Dokument pomocí architektury *ActiveRecord*. V této architektuře instance třídy reprezentuje řádek v tabulce databáze a umožňuje nad tímto záznamem provádět jednotlivé operace jako jsou úpravy, mazání nebo vytváření.

Controller je zde zastoupen třídami `DocumentsController` a `CollectionsController`, které zprostředkovávají jednotlivé akce dostupné uživateli. Konkrétně se jedná o zobrazení indexu obsahujícího stránkovaně vypsané dokumenty a kolekce. V tomto indexu je možnost dostat se na zobrazení detailu jednotlivých dokumentů a kolekcí. Dále je také přítomná možnost vygenerovat a zobrazit invertovaný index a také nejdůležitější akce – samotné vyhledávání.

Poslední část **View** je v tomto případě zprostředkovaná pomocí HTML ERB (*Embedded Ruby*) souborů, které umožňují zakomponovat Ruby kód do HTML značek. Jednotlivé soubory se zde mapují na dostupné akce v `DocumentsController` a v `CollectionsController`.

Zvlášť jsou pak vyčleněny pomocné třídy v kontextu této aplikace označené jako **Services**. Ty zprostředkovávají komplexnější logiku a jsou vytvářeny a volány v rámci jednotlivých akcí. Konkrétně jsou zde implementovány třídy pro samotnou vyhledávací logiku – `SearchEngine` – a tvorbu invertovaného indexu – `IndexGenerator`.

8.2 Vyhledávaní

Pro účely dvojjazyčného vyhledávání byly využity vektorové reprezentace vytvořené postupem popsaným v minulé kapitole. Konkrétně se využívala opět kombinace parametrů, která nejlépe odpovídala P@k metrice.

Nad těmito vektorovými reprezentacemi byl vytvořený skript v Pythonu. Pro dané hledané slovo a jazyk pak vrací 5 nejbližších slov[1] v druhém jazyce. Jediným problémem tohoto přístupu je nutnost počítat kosinové vzdálenosti hledaného slova a všech slov v druhém jazyce. Tento faktor je poměrně rychlostně limitující a vzhledem k potřebě počítat tyto vzdálenosti ve fázi vyhledávání má výsledný vyhledávač poměrně značnou latenci.

Při dotazu $q = (t_1, \ldots, t_n)$, tedy pro jednotlivá slova dotazu $t_i \in q$ vrátil skript 5 překladů $\varphi_{i,1}, \ldots, \varphi_{i,5}$ a zpracovanou[2] variantu tokenu t_i. Ve výsledku se poté objevují dokumenty, které obsahují buď jeden z nejlepších překladů v jazyce překladu $\varphi_{i,j}$ nebo jedno z hledaných slov t_i.

Vyhledávač navíc pro výsledné dokumenty označuje nejrelevantnější slova a zobrazuje krátký úsek vybraný z okolí těchto slov pro texty z obou jazyků. Názorná ukázka fungování je viditelná na obrázku 8.1, kde byl v tomto případě zadán dotaz „zeměpisný obsah“ v českém jazyce. U jednotlivých dokumentů je viditelná i relevance. Na stránce jsou pak tímto způsobem stránkovaně zobrazeny všechny dokumenty, které obsahovaly část dotazu nebo jeho překlad. V detailu jednotlivých dokumentů je pak k dispozici celkový český a anglický text – nejenom jeho krátký úsek – a rozepsané unikátní anglické a české tokeny.

8.3 Řazení výsledků

Výsledný indexační model nelze jednoduše charakterizovat jako vektorový ani booleovský, protože používá vlastní způsoby řazení výsledků a hledání dokumentů. Metrika pro řazení byla navržena na základě jednoduchých pravidel.

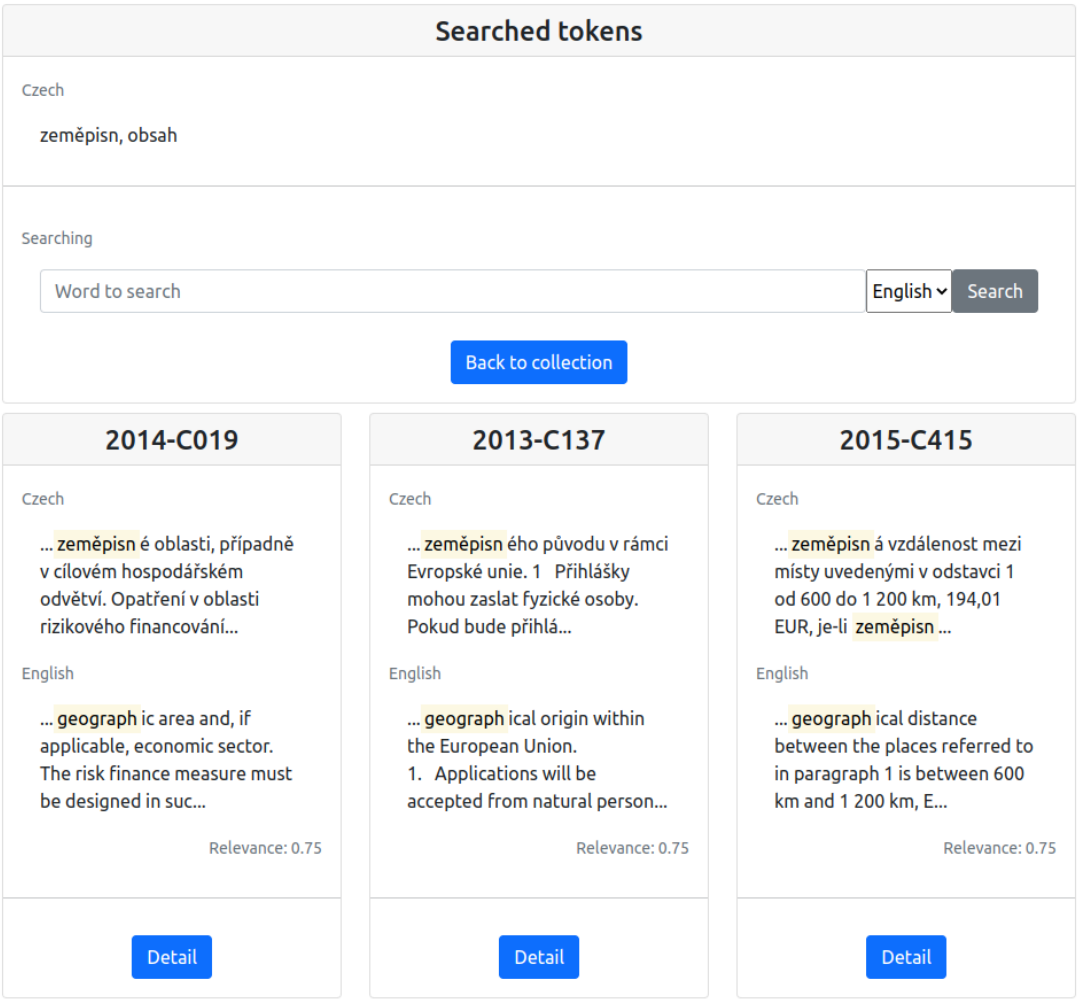
Při výpočtu metriky pro jazyk dotazu $\mathrm{w_q}$ určíme průnik slov z dotazu a velikost podělíme počtem slov v dotazu. Dokumenty, které mají větší průnik, tak mají větší váhu. Konkrétně tento fakt můžeme za předpokladu, že dokument obsahoval k z n termů v dotazu $q = (t_1, \ldots, t_n)$, vyjádřit následovně.

$$\mathrm{w_q} = \frac{k}{n} \tag{8.1}$$

Druhá dílčí metrika se vypočítá s pomocí pěti nejlepších překladů pro daná slova. Pro term t_i dostaneme pomocí skriptu, který tyto překlady hledá termy $\varphi_{i,1}, \ldots, \varphi_{i,5}$. Konkrétně $\varphi_{i,j}$ zde označuje j-tý nejlepší překlad pro term t_i. Konkrétní dokument ve výsledku pak v textu překladu

[1] Nejbližších ve smyslu kosinové vzdálenosti.

[2] Zpracováním je zde konkrétně myšlena tokenizace, která byla popsána v kapitole 6.



Obrázek 8.1 Ukázka práce webového vyhledávače.

obsahuje některé z překladů – pro naše účely bereme v potaz ten, který je z hlediska vzdálenosti nejbližší. Tedy pro zkoumaný dokument a term t_i dostaneme j_i-tý překlad (φ_{i,j_i}), který je v textu přítomen. Pro jeden term vypočteme dílčí skóre jako podíl $1/j_i$. Metrika $\mathrm{w_{tr}}$ je poté vypočtena jako součin dílčích výsledků pro jednotlivé termy.

$$\mathrm{w_{tr}} = \frac{1}{j_1} \cdot \frac{1}{j_2} \cdots \frac{1}{j_n} \tag{8.2}$$

Výsledná relevance nakonec vznikne spojením a normalizací těchto dílčích metrik dotazu a překladu.

$$\mathrm{w} = \frac{\mathrm{w_{tr}} + \mathrm{w_q}}{2} \tag{8.3}$$

V momentě, kdy nebyl přítomný překlad nebo dotazovaný term v dokumentu byla v momentální fázi řešení dílčí metrika nastavena na hodnotu 0. Z toho pak vychází, že i finální relevance pro daný dokument byla nulová. Do výsledku se pak tyto dokumenty s „nulovou“ relevancí i přes toto stále dostaly s ohledem na to.

Na základě takto spočítané relevance bylo možné jednotlivé dokumenty sestupně řadit. Nejrelevantnější dokumenty se tedy vyskytovaly nejdříve. Výhodou tohoto způsobu navíc je, že tato metrika zohledňuje kvalitu překladu – tedy v momentě, kdy dokument obsahoval hledaná slova, ale překlady v druhém jazyce byly buď vzdálené nebo nebyly vůbec přítomné, nebyl na něj brán takový ohled jako na dokument, který měl kvalitní překlady.

Na obrázku 8.1 je viditelná relevance, která pro první dokumenty vycházela stejně. Toto chování je ale očekávané vzhledem k výskytu slov ve stejném jazyce – konkrétně pro token „zeměpisn“, by jako nejbližší slovo ve druhém jazyce bylo opět to samé slovo. To ale logicky ve většině dokumentů, které nejsou zrovna konkrétní česká nařízení nebude přítomné. Toto chování je specifikem EUR-Lex corpu, jak už bylo diskutováno v předchozí kapitole.

8.4 Diskuze

Prostorů pro zlepšení je v tomto případě více – tento program také slouží do značné míry jako experimentální prototyp.

Hlavním problémem je rychlost odpovědí, která je daná zejména hledáním nejbližších překladů pomocí Python skriptu. Samotná Ruby on Rails část, která řešila označování dokumentů, načítání textů z databáze a řazení dle relevance už byla oproti diskutovanému skriptu rychlá. Možností, jak tento model optimalizovat je předpočítání nejbližších překladů pro všechna slova v obou jazycích. Tato možnost je ale výpočetně velmi náročná, protože má složitost $\mathcal{O}(n \cdot k)$, kde n je počet všech unikátních tokenů v corpu a k počet překladů, které zohledňujeme. V našem corpu se hodnota n pohybuje v řádu statisíců. Hodnota k je konstantní a výsledná asymptotická složitost tohoto algoritmu je tedy $\mathcal{O}(n)$.

V tomto momentě není jasné, v jaké technologii bude potřeba finální aplikaci vytvořit, a proto nedávají v tuto chvíli smysl další optimalizace. Vzhledem k tomuto faktu a výpočetní náročnosti tohoto úkonu bylo rozhodnuto, že tento krok nebude podstoupen a pro účely prototypu byla zanechána verze s vyšší latencí.

Další problém, který byl patrný už v předchozí kapitole je přítomnost slov z druhého jazyka v jazyce překladu. Tento problém je daný mezinárodnostní povahou corpu a v tomto momentě nemám konkrétní řešení, které by tento problém odstranily.

Posledním faktem je, že příklad, který byl v této kapitole nastíněn bere v potaz jedny z nejlepších vektorových reprezentací, které byly získány a tímto způsobem prototyp rozhodně nebude fungovat pro velkou část slov. Toto je dáno zejména nízkým objemem dat.

Kapitola 9
Závěr

Cílem práce bylo v rámci rešerše prozkoumat word embedding techniky, indexační nástroje a současné přístupy k vícejazyčnému vyhledávání. Dále v rámci implementace vytvořit dvojjazyčný Word2Vec model, který by přímo zachycoval vztahy mezi přeloženými slovy a bylo by možné ho využít pro indexaci ve dvojjazyčném vyhledávači. Poslední částí pak byla samotná tvorba prototypu webového vyhledávače vystavěného nad těmito naučenými vektorovými reprezentacemi.

Všechny tyto body byly v rámci práce uskutečněny. V rámci rešerše byly představeny indexační modely jako jsou vektorový a booleovský model, následně word embedding techniky – především pak Word2Vec a GloVe. Nakonec byly vyzkoumány přístupy k vícejazyčným překladům, jmenovitě pak CL-LSI, vícejazyčný Word2Vec, MUSE a Duet.

V rámci implementace byl úspěšně vytvořen model, který byl schopen reprezentovat překlady mezi slovy v poměrně jasné rovině. Díky vizualizačním technikám a různým metrikám bylo ukázáno, že daný model je schopen reprezentovat překlady tak, že vektorové reprezentace překladu a původního slova byly v konečném prostoru mapovány „blízko sebe“.

Nad těmito reprezentacemi pak byl implementován prototyp dvojjazyčného vyhledávače, který vracel pro vybraná slova relevantní výsledky a fungoval dle očekávání. Na druhou stranu vzhledem k poměrně nízkému objemu dat nebyly tyto reprezentace vždy přesné a často se stávalo, že výsledky relevantní nebyly vůbec. Pro svou poměrně značnou latenci ve vyhledávací fázi, slouží výsledný vyhledávač opravdu spíše jako prototyp.

Budoucí vývoj V rámci navazující práce je v plánu porovnat dosažené výsledky s dalšími nejmodernějšími přístupy k dané problematice, které byly v rámci této práce zkoumány – konkrétně se jedná o modely jako MUSE nebo vícejazyčný Word2Vec. Pro tyto potřeby je také nutné zvětšit objem kolekce, abychom dosáhli alespoň částečné porovnatelnosti.

Zároveň můžeme využít možnosti práce s více než dvěma jazyky, vzhledem k tomu, že portál EUR-Lex obsahuje nařízení a dokumenty, se kterými jsme doposud pracovali i v dalších jazycích. Na základě těchto dat, by tedy bylo možné zde vytvořený paralelní corpus dále rozšířit. S pomocí takto vytvořené kolekce bychom mohli porovnávat vektorové reprezentace v jednotlivých jazycích a vystavět nad nimi více než jen dvojjazyčný vyhledávač.

Bibliografie

1. BOKKA, Karthiek Reddy; HORA, Shubhangi; JAIN, Tanuj; WAMBUGU, Monicah. *Deep Learning for Natural Language Processing: Solve your natural language processing problems with smart deep neural networks*. Packt Publishing Ltd, 2019.
2. SCHÜTZE, Hinrich; MANNING, Christopher D; RAGHAVAN, Prabhakar. *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008.
3. BUTTCHER, Stefan; CLARKE, Charles LA; CORMACK, Gordon V. *Information retrieval: Implementing and evaluating search engines*. Mit Press, 2016.
4. LASHKARI, Arash Habibi; MAHDAVI, Fereshteh; GHOMI, Vahid. A Boolean Model in Information Retrieval for Search Engines. In: *2009 International Conference on Information Management and Engineering*. 2009, s. 385–389. Dostupné z DOI: 10.1109/ICIME.2009.101.
5. PETERS, Carol; BRASCHLER, Martin; CLOUGH, Paul. *Multilingual information retrieval: From research to practice*. Springer, 2012.
6. GARCIA, Dr. E. The Extended Boolean Model - Weighted Queries: Term Weights, p-Norm Queries and Multiconcept Types. Boolean OR Extended? AND that is the Query. 2006.
7. VAŠATA, Daniel; KLOUDA, Karel. *VZD Přednášky*. 2022. Dostupné také z: https://kam.fit.cvut.cz/bi-vzd/lectures/index.html.
8. CHEN, Lei. Curse of Dimensionality. In: *Encyclopedia of Database Systems*. Ed. LIU, LING; ÖZSU, M. TAMER. Boston, MA: Springer US, 2009, s. 545–546. ISBN 978-0-387-39940-9. Dostupné z DOI: 10.1007/978-0-387-39940-9_133.
9. HARRIS, Zellig S. Distributional structure. *Word*. 1954, roč. 10, č. 2-3, s. 146–162.
10. RUDER, Sebastian. *On word embeddings - Part 1* [http://ruder.io/word-embeddings-1/]. 2016.
11. D'ADDONA, Doriana Marilena. Neural Network. In: *CIRP Encyclopedia of Production Engineering*. Ed. LAPERRIÈRE, Luc; REINHART, Gunther. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, s. 911–918. ISBN 978-3-642-20617-7. Dostupné z DOI: 10.1007/978-3-642-20617-7_6563.
12. SHARMA, Sagar; SHARMA, Simone; ATHAIYA, Anidhya. Activation functions in neural networks. *towards data science*. 2017, roč. 6, č. 12, s. 310–316.
13. BENGIO, Yoshua; DUCHARME, Réjean; VINCENT, Pascal. A neural probabilistic language model. *Advances in Neural Information Processing Systems*. 2000, roč. 13.
14. MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 2013.

15. MIKOLOV, Tomas; SUTSKEVER, Ilya; CHEN, Kai; CORRADO, Greg S; DEAN, Jeff. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*. 2013, roč. 26.
16. AYYADEVARA, V Kishore. Pro machine learning algorithms. *Berkeley: Apress*. 2018.
17. CHAUBARD, Francois; MUNDRA, Rohit; SOCHER, Richard. *CS 224D: Deep learning for NLP - Stanford University*. 2016. Dostupné také z: `https://cs224d.stanford.edu/lecture_notes/notes1.pdf`.
18. STRAKA, Milan. *CRF, CTC, Word2Vec*. 2021. Dostupné také z: `https://ufal.mff.cuni.cz/~straka/courses/npfl114/2021/slides.pdf/npfl114-09.pdf`.
19. MIKOLOV, Tomas; LE, Quoc V; SUTSKEVER, Ilya. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*. 2013.
20. RUDER, Sebastian. *On word embeddings - Part 2: Approximating the Softmax* [`http://ruder.io/word-embeddings-softmax`]. 2016.
21. MORIN, Frederic; BENGIO, Yoshua. Hierarchical probabilistic neural network language model. In: *International workshop on artificial intelligence and statistics*. 2005, s. 246–252.
22. BOJANOWSKI, Piotr; GRAVE, Edouard; JOULIN, Armand; MIKOLOV, Tomas. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*. 2017, roč. 5, s. 135–146.
23. RUDER, Sebastian. *On word embeddings - Part 3: The secret ingredients of word2vec* [`http://ruder.io/secret-word2vec/`]. 2016.
24. PENNINGTON, Jeffrey; SOCHER, Richard; MANNING, Christopher D. Glove: Global vectors for word representation. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, s. 1532–1543.
25. PENNINGTON, Jeffrey. *GloVe: Global Vectors for Word Representation*. 2015. Dostupné také z: `https://nlp.stanford.edu/projects/glove/`.
26. ROSARIO, Barbara. Latent semantic indexing: An overview. *Techn. rep. INFOSYS*. 2000, roč. 240, s. 1–16.
27. LITTMAN, Michael L; DUMAIS, Susan T; LANDAUER, Thomas K. Automatic cross-language information retrieval using latent semantic indexing. In: *Cross-language information retrieval*. Springer, 1998, s. 51–62.
28. MITRA, Bhaskar; DIAZ, Fernando; CRASWELL, Nick. Learning to match using local and distributed representations of text for web search. In: *Proceedings of the 26th International Conference on World Wide Web*. 2017, s. 1291–1299.
29. CONNEAU, Alexis; LAMPLE, Guillaume; RANZATO, Marc'Aurelio; DENOYER, Ludovic; JÉGOU, Hervé. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*. 2017.
30. NLTK Project, 2022. Dostupné také z: `https://www.nltk.org/`.
31. ROSA, Rudolf. *Czech and English abstracts of ÚFAL papers*. 2016. Dostupné také z: `http://hdl.handle.net/11234/1-1731`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.
32. JANSEN, Stefan. Word and phrase translation with word2vec. *arXiv preprint arXiv:1705.03127*. 2017.
33. ERDEM, Kemal. *T-SNE clearly explained*. Towards Data Science, 2020. Dostupné také z: `https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a`.
34. VAN DER MAATEN, Laurens; HINTON, Geoffrey. Visualizing data using t-SNE. *Journal of machine learning research*. 2008, roč. 9, č. 11.

Obsah přiloženého média

- `dokument` — adresář se zdrojovými kódy textu bakalářské práce
  - `chapters` — jednotlivé kapitoly práce
  - `pictures` — přiložené obrázky
  - `thesis.pdf` — text práce ve formátu PDF
  - `ctufit-thesis.tex` — hlavní zdrojový kód textu práce ve formátu LaTeX
- `implementace`
  - `word2vec` — skripty pro učení Word2Vec modelu
    - `README.md` — instrukce pro inicializaci Python prostředí
    - `requirements.txt` — popis Python prostředí pro instalaci
    - `helpers` — pomocné metody pro tvorbu corpu a práci s Word2Vec modelem
    - `corpus` — vytvořené kolekce
      - `eurlex`
      - `ufal`
    - `trained_models` — naučené Word2Vec modely
  - `search_app` — webový vyhledávač v Ruby on Rails
    - `README.md` — instrukce pro spuštění vyhledávače
    - `app` — hlavní zdrojové kódy vyhledávače
      - `controllers` — kontrolery obsluhující jednotlivé akce uživatele
      - `helpers` — moduly obsahující pomocné metody pro View
      - `models` — ActiveRecord modely mapující databázové tabulky na třídy
      - `services` — pomocné třídy
    - `db`
      - `migrate` — databázové migrace
      - `schema.rb` — současné databázové schéma
      - `seeds.rb` — skript pro naplnění databáze počátečními daty
    - `python_search` — hledání nejbližších slov pomocí Pythonu