FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Vícejazyčné vyhledávání v textových dokumentech

Ondřej Matura

PLZEŇ

2023

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Vícejazyčné vyhledávání v textových dokumentech

Ondřej Matura

Vedoucí práce
Ing. Ladislav Lenc, Ph.D.

PLZEŇ **2023**

Citace v seznamu literatury:
MATURA, Ondřej. *Vícejazyčné vyhledávání v textových dokumentech*. Plzeň, 2023. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Ladislav Lenc, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Ondřej MATURA**
Osobní číslo:	**A19B0135P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Vícejazyčné vyhledávání v textových dokumentech**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se se systémem pro dvojjazyčné vyhledávání textových dokumentů aktuálně používaném v portálu Porta fontium.
2. Prostudujte relevantní metody pro vícejazyčné vyhledávání. Zaměřte se zejména na metody založené na kontextových vektorech a transformacích vektorových prostorů.
3. Vyberte alespoň jednu vhodnou metodu, implementujte ji a proveďte její srovnání se stávajícím systémem. Pro srovnání vyberte vhodná data.
4. Navrhněte a implementujte softwarový modul pro vícejazyčné vyhledávání, pomocí nehož začleníte implementovanou metodu do portálu Porta fontium.
5. Kriticky zhodnoťte dosažené výsledky a navrhněte možná rozšíření.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Ing. Ladislav Lenc, Ph.D.**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **3. října 2022**
Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2022

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

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

V Plzni dne 4. května 2023

. . . . . .
Ondřej Matura

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Tato bakalářská práce zkoumá potenciál pro zlepšení vícejazyčného vyhledávání informací (CLIR) v rámci portálu Porta fontium, který poskytuje přístup k historickým materiálům z Česko-Bavorského pohraničí. Je tedy důležité umožnit vyhledávání v českých i německých dokumentech zároveň. Práce nejprve poskytuje přehled metod CLIR, zahrnující jak tradiční, tak moderní přístupy. Tato analýza zdůrazňuje silné a slabé stránky každé metody, čímž připravuje cestu pro návrh řešení. Poté studie zkoumá dostupné CLIR datasety a hodnotící metriky. Následně jsou provedeny experimenty pro vyhodnocení výkonnosti vybraných CLIR metod s využitím identifikovaných datasetů a metrik. Tato analýza směřuje k vývoji softwarového modulu CLIR, který bude možné použít v systému Porta fontium.

Abstract

This bachelor's thesis investigates the potential for improving cross-language information retrieval (CLIR) within the Porta fontium portal, which provides access to historical materials from the Czech-Bavarian border region. It is therefore important to enable simultaneous search in both Czech and German documents. The thesis first provides an overview of CLIR methods, including both traditional and modern approaches. This analysis highlights the strengths and weaknesses of each method, paving the way for a solution proposal. Afterwards, the study examines available CLIR datasets and evaluation metrics. Experiments are then conducted to evaluate the performance of selected CLIR methods, using the identified datasets and metrics. This analysis leads to the development of a CLIR software module that can be integrated into the Porta fontium system.

Klíčová slova

CLIR • NLP • BERT model • slovní vektory

Poděkování

Chtěl bych upřímně poděkovat Ing. Ladislavu Lencovi, Ph.D., který mi byl velkou oporou a průvodcem během celého procesu psaní práce. Naše pravidelné schůzky a jeho ochota a trpělivost byly pro mě neocenitelné a pomohly mi zvládnout všechny výzvy spojené s realizací tohoto projektu. Dále bych chtěl vyjádřit svou vděčnost všem lidem spojeným s projektem Porta fontium, který umožnil vznik této bakalářské práce.

Obsah

1 Úvod **1**

2 CLIR **2**

2.1 Metody . . . 2

2.1.1 Překlad dokumentů vs. překlad dotazu . . . 2

2.1.2 Mezijazyčné analogie pomocí lineárních transformací mezi sémantickými prostory . . . 3

2.1.3 Slovní vektory pomocí syntaktické analýzy . . . 4

2.1.4 CLIR BERT . . . 5

2.1.5 Rozšíření dotazu . . . 7

2.2 Vyhledávací systém Porta fontium . . . 8

2.3 Datasety . . . 10

2.3.1 CLIRMatrix . . . 10

2.3.2 WikiCLIR . . . 11

2.3.3 Extended CLEF eHealth test collection . . . 11

2.4 Metriky . . . 12

2.4.1 Přesnost . . . 12

2.4.2 Úplnost . . . 13

2.4.3 Průměrná přesnost . . . 13

2.4.4 MAP . . . 13

2.4.5 CG . . . 13

2.4.6 DCG . . . 14

2.4.7 nDCG . . . 14

2.5 Závěr . . . 14

3 Experimenty **16**

3.1 Otestování Porta fontium . . . 16

3.2 Porovnání metod předzpracování a následného zpracování při vyhledávání pomocí slovních vektorů . . . 18

3.2.1 Návrh na vylepšení Porta fontium . . . 19

3.3 Vyhledávání pomocí dokumentových vektorů 20
3.4 ColBERT-X 21
3.5 Závěr 22

4 Modul pro vícejazyčné vyhledávání 23
4.1 ColBERT 24
4.2 ColBERT-X 25
4.3 Vlastní úpravy 26
4.4 REST API 27
4.5 Příkazy 29
4.5.1 `train` 29
4.5.2 `index` 31
4.5.3 `index_faiss` 32
4.5.4 `retrieve` 32
4.5.5 `retrieve_api` 33
4.6 Skripty 33
4.6.1 `passage_dataset_from_qrels.py` 33
4.6.2 `reduce_document_collection.py` 34

5 Závěr 35

A Uživatelská příručka 36
A.1 Modul pro vyhledávání 36

Bibliografie 37

Seznam obrázků 39

Seznam tabulek 40

Úvod 1

Vícejazyčné vyhledávání, nebo také cross-language information retrieval (CLIR) je specifické odvětví oboru získávání informací (information retrieval - IR), které se zabývá získáváním informací z kolekce dokumentů, která je psaná v jiném jazyce, než v jakém je příslušný dotaz. CLIR je významné, jelikož umožňuje uživatelům přistupovat k většímu množství zdrojů, což může být obzvláště užitečné v oborech, kde jsou informace publikovány v různých jazycích.

Konkrétním cílem této práce je seznámit se s CLIR funkčností již existujícího IR systému, který je součástí portálu „Porta fontium“ a pokusit se ji vylepšit. Porta fontium je rozsáhlá digitální síť historických dokumentů, která vznikla spoluprací západočeských a bavorských státních archivů. Porta fontium je dostupná veřejnosti a obsahuje matriky, kroniky, mapy, periodika a jiné historické záznamy. Navzdory tomu je však aktuální CLIR funkcionalita tohoto systému poněkud limitovaná a dle názoru uživatelů je zde prostor pro zlepšení.

Pro dosažení tohoto cíle budou v této práci prověřeny moderní CLIR přístupy a techniky a z toho vybrány ty nejvhodnější pro danou aplikaci. Účelem tedy není najít nejlepší možnou metodu pro řešení CLIR obecně, ale zohlednit i praktická hlediska pro daný případ, jako složitost implementace a integrace do systému. Dále bude vybrán vhodný dataset a metriky, pomocí kterých bude možné nakonec porovnat nově vybrané metody s těmi původními a celkově vyhodnotit dosažené výsledky.

CLIR 2

Úloha mezijazyčného vyhledávání informací (CLIR) řeší výzvu vyhledávání a získávání relevantních informací ze sbírky dokumentů napsaných v různých jazycích, a to na základě dotazu ve zdrojovém jazyce. Tento proces zahrnuje pochopení dotazu uživatele, překlad daného dotazu nebo dokumentů do společného jazyka nebo sdílené reprezentace a poté porovnání dotazu s odpovídajícími dokumenty na základě relevance. Pro dosažení tohoto cíle se používají různé techniky a přístupy, jako je překlad dotazů, překlad dokumentů, modely založené na strojovém učení nebo mezijazyčné rozšiřování dotazu. Tato kapitola se zaměřuje na analýzu různých technik a datasetů CLIR s cílem identifikovat nejúčinnější přístup pro implementaci softwarového modulu CLIR v reálném světě.

Následné otestování a případné trénování účinného CLIR systému vyžaduje rozmanitá a komplexní data, a pro tyto účely zde budou prozkoumány také různé CLIR datasety, které se liší svým rozsahem, pokrytím jazyků a specifičností oboru. Kapitola poskytne přehled každého datasetu a ohodnotí jejich potenciál pro následné experimenty. V rámci těchto experimentů bude také nutné použít metriky pro ohodnocení výkonnosti, které zde také budou blíže popsány.

2.1 Metody

2.1.1 Překlad dokumentů vs. překlad dotazu

Pravděpodobně nejpřímočařejším způsobem, jak implementovat CLIR, je kombinace již existujících metod IR a strojového překladu (machine translation - MT). Tedy nejdříve strojově přeložíme buď dotaz při samotném vyhledávání, a nebo všechny dokumenty předem a dále již můžeme považovat úlohu za obyčejný IR problém. Nastává však otázka, zda je při takovém přístupu lepší překládat dotaz nebo dokumenty. O této konkrétní problematice pojednává článek [SP20]. Pro samotné MT autoři použili a také porovnali statistické a neuronové metody. U statistických byly navíc srovnány různé způsoby předzpracování a následného zpracování.

Dříve se předpokládalo, že pro CLIR poskytuje překlad dokumentů lepší výsledky, než překlad dotazu. Toto tvrzení je založeno na tom, že překlad celých dokumentů by měl být kvalitnější, kvůli mnohem většímu kontextu, než u jednotlivých dotazů. I přes to se však v praxi častěji používal překlad dotazu, protože je jednodušší a nevyžaduje zdlouhavé překládání všech dokumentů. Tato domněnka je už však relativně stará a nebere v potaz dnešní nejmodernější metody strojového překladu.

Kvalita překladů byla vyhodnocena pomocí BLEU a PER metriky. K vyhodnocení vyhledávání byly použity metriky MAP a P@10. Pro trénování, ladění a testování strojového překladu autoři použili datovou sadu UFAL Medical Corpus: `https://ufal.mff.cuni.cz/ufal_medical_corpus`. Jako testovací data byla použita autory vytvořená data z jejich předchozích prací: `https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2925`.

Samotné překládání všech dokumentů (1,1 miliónu) z angličtiny do sedmi jazyků trvalo statistickému překladači 21 dní s využitím 200 CPU jader a 20 GB RAM pro každé jádro. Neuronovému překladači trval překlad 20 dní za pomoci 20 grafických karet s pamětí 10 GB. Pro druhou metodu trvalo přeložení 66 dotazů statistickému překladači 15 minut, zatímco neuronovému 3 minuty.

V těchto experimentech bylo zjištěno, že s pomocí nejmodernějších strojových překladačů jsou metody překladu dotazu obecně lepší, než metody překladu dokumentů a zároveň neuronové strojové překlady jsou kvalitnější, než ty statistické. Je ale třeba brát na vědomí, že experimenty byly prováděny na datech výhradně z lékařské domény.

2.1.2 Mezijazyčné analogie pomocí lineárních transformací mezi sémantickými prostory

Dalším způsobem, jakým lze řešit CLIR, je využitím několika vlastností slovních vektorů uvedených v článku [BTS18]. Každé slovo z přirozeného jazyka lze převést na vektor pomocí různých algoritmů (např. Word2Vec, GloVe, FastText). Počet složek těchto vektorů je poté dán použitým algoritmem. Prostor těchto slovních vektorů se někdy nazývá sémantický.

Tato reprezentace (pokud je kvalitní) s sebou přináší různé výhody. Jednou z nich je možnost mezi těmito vektory používat jednoduchou aritmetiku, která pak i významově platí pro příslušná slova. Například analogie *„muž je vůči ženě to, co je král vůči královně“* lze vyjádřit vektorově jako $V_{muž} - V_{žena} \approx V_{král} - V_{královna}$. Díky této vlastnosti je pak možné aproximovat pozici slovního vektoru čistě na základě vztahů jiných slov. Dále také platí to, že vektory podobných slov se zároveň nachází blízko sebe, proto se také slovní vektory často používají pro hledání synonym.

Pro účely mezijazyčných úloh je další výhodou to, že je možné tyto prostory

transformovat mezi různými jazyky pomocí lineárních transformací. Díky tomu je možné porovnávat a spojovat sémantické prostory více jazyků, což usnadňuje analýzu a modelování významu slov nebo konceptů napříč jazyky. Existuje několik metod, které se zabývají nalezením těchto transformací. Tyto metody byly porovnány v článku [BTS18].

Tyto vlastnosti můžeme dále kombinovat. Například pokud bychom chtěli zjistit ženskou verzi anglického slova *king* (resp. mužskou verzi slova *queen*), pak můžeme využít znalosti českého jazyka a použít vztah slov *muž* a *žena*.

Autoři otestovali 3 různé způsoby pro nalezení lineární transformace mezi dvěma prostory:

- metoda nejmenších čtverců,
- ortogonální transformace,
- kanonická korelační analýza.

A déle 3 způsoby hledání podobných slov:

- jednojazyčné - jako kontrolní
- dvojjazyčné - transformace přímo mezi dvěma jazyky
- vícejazyčné - společná transformace do angličtiny

Otestovány také byly metody následného zpracování sémantických prostorů - vyrovnání těžiště na střed a normalizace vektorů. Zde se ukázalo, že nejlepší výsledky byly dosaženy vždy při použití obou metod zároveň.

Pro dvojjazyčné vyhledávání měla nejlepší výsledky ortogonální transformace, pro vícejazyčné byla lepší kanonická korelační analýza. Metoda nejmenších čtverců byla ve všech případech nejhorší.

Napříč šesti jazyky bylo dosaženo přesnosti 38,2% pro vícejazyčné vyhledávání a 43,1% pro dvojjazyčné oproti 51,1% při „obyčejném“ jednojazyčném.

2.1.3 Slovní vektory pomocí syntaktické analýzy

Často používaný algoritmus pro získání slovních vektorů je word2vec. Jak již bylo popsáno, vektory slov, které spolu nějak souvisí, by měly mezi sebou mít kratší vzdálenost v sémantickém prostoru. Tímto způsobem ale nelze jednoduše získat opravdu podobná slova, či synonyma. Pro vytvoření prostoru slovních vektorů používá word2vec velké množství textu, ve kterém se slova považují za související, pokud jsou blízko u sebe. Tento způsob však nepočítá s opravdovými závislostmi mezi slovy v daném jazyce. To může mít za následek to, že pokud použijeme reprezentaci vytvořenou ve word2vec pro hledání podobných slov, můžou nám vyjít slova,

která spolu sice souvisí, ale nejsou si vůbec podobná ve významu - např. „object-oriented“ a „objective-c“. Autoři článku [LG14] proto navrhují úpravu word2vec tak, aby k vyhodnocení, zda spolu slova souvisí, použil syntaktickou analýzu (určení větné skladby). Tímto způsobem bychom měli dostat lepší výsledky - např. „object-oriented“ a „data-driven“.

Článek pojednává pouze o jednojazyčném modelu. Zda by tento způsob mohl dobře fungovat i mezijazyčně, by bylo potřeba vyhodnotit vlastními experimenty. Pro syntaktickou analýzu v „libovolném“ jazyce by bylo možné použít UDPipe. Pro samotnou mezijazyčnost pak vytvořit prostor slov pro každý požadovaný jazyk a ty pak mezi sebou transformovat pomocí lineárních transformací.

2.1.4 CLIR BERT

Dalším způsobem, jakým se lze postavit k CLIR, je s využitím strojového učení sestavit model, který bude řešit tuto konkrétní úlohu. Jedním z modelů, ze kterých lze vycházet je předtrénovaný model BERT.

BERT, což je zkratka pro *Bidirectional Encoder Representations from Transformers* (volně přeloženo - Obousměrně kódované řetězce z transformer modelů), je sofistikovaná a výkonná architektura jazykového modelu, která byla před-trénovaná nad rozsáhlým korpusem textových dat. Používá enkodér, založený na transformer modelu, který zpracovává vstupní text v obou směrech (vpřed i vzad), a tím zachytává kontextové informace z předcházejících i následujících slov. Tento obousměrný enkodér umožňuje BERTu lépe porozumět složitým vztahům a závislostem mezi slovy ve větě. Kromě toho BERT využívá metodu trénování s maskovaným jazykovým modelem (masked language model - MLM), kde jsou náhodně zakryta (maskovaná symbolem `[Mask]`) některá slova ve vstupním textu během předtrénování a model se učí tato slova předpovědět na základě jejich kontextu. Tato MLM úloha zlepšuje schopnost BERTu zachytit složitější významy a skladby slov ve větách. Kromě MLM se BERT také během před-trénování učí klasifikovat, zda dvě věty mohou být významově po sobě. Tato úloha na druhou stranu napomáhá BERTu porozumět textu jako celku napříč větami. Kombinace těchto vlastností dělají z BERTu state-of-the-art model pro různé úlohy zpracování přirozeného jazyka.

Článek [Jia+20] navrhuje modifikovanou verzi pro vícejazyčné vyhledávaní - na vstupu je jednoslovný dotaz v angličtině q a věta v jiném jazyce s. Na výstupu pak modifikovaný BERT predikuje pravděpodobnost $p(q|s)$, že se dotaz q vyskytuje ve větě s. Při vyhledávání pak používáme víceslovné dotazy Q, jenž se skládají z jednoslovných dotazů q a dokumenty D, které se skládají z vět s. Jako skóre relevance mezi Q a D vypočítáme pravděpodobnost, že jsou všechna slova dotazu relevantní k alespoň jedné větě dokumentu. Tedy: $1 - \prod_{s \in D}(1 - \prod_{q \in Q} p(q|s))$

Pro modifikaci se vytvoří „jednoduchá“ trénovací data získaná z překladů vět.

Vezmeme-li větu v cizím jazyce, pak je anglické slovo vůči této větě relevantní, pokud se vyskytuje v jednom z možných překladů této věty. Naopak jako nerelevantní anglické slovo vůči dané větě lze použít náhodné anglické slovo, které se nevyskytuje v překladu věty. Tyto příklady se pak použijí jako pozitivní a negativní trénovací data.

Autoři trénovali model po jednu epochu s využitím grafické karty Nvidia Tesla V100. Tréování trvalo celkem jeden týden.

Modifikovaný BERT byl porovnán s těmito modely:

1. Pravděpodobnostní CLIR model: Jedná se o generativní pravděpodobnostní model, který vyžaduje pravděpodobnostní překladový slovník. Tento slovník byl vygenerován ze zarovnaných paralelních dat pomocí nástrojů GIZA++ a Berkeley aligner.

2. Pravděpodobnostní model výskytu: Tento model vypočítává relevance dokumentu jako pravděpodobnost, že každý termín z dotazu se v dokumentu vyskytuje alespoň jednou.

3. Model neuronové sítě s attention mechanismem pro zjištění relevance dotazu (Query Relevance Attentional Neural Network Model - QRANN): Tento model používá attention mechanismus k výpočtu kontextového vektoru odvozeného ze slovních vektorů z cizích vět, následovaného feedforward vrstvou k zachycení vztahů mezi dotazovými slovy. Model byl trénován na víceslovných i jednoslovných dotazech.

4. Model skalárního součinu (Dot-Product): Jedná se o zjednodušenou verzi modelu QRANN, který vypočítává kontextový vektor ze slovních vektorů dotazu pomocí multiplikativního attention mechanismu a následovaného skalárního součinu s původními slovními vektory. Tento model byl trénován pouze na jednoslovných dotazech.

Tabulka 2.1: Porovnáné MAP skóre výsledků pro CLIR BERT [Jia+20] nad datasetem z IARPA MATERIAL programu

model	**jednoslovné dotazy**	**všechny dotazy**
Pravděpodobnostní CLIR	57,4	61,2
Pravděpodobnostní model výskytu	51,4	56,9
BERT	61,3	56,8
Dot-Product	50,8	39,2
QRANN	55,8	45,5

Ve všech testech (2.1) byl BERT výkonnější než ostatní modely založené na neuronových sítích (QRANN a Dot-Product), avšak výsledky byly srovnatelné s pravděpodobnostními modely.

2.1.5 Rozšíření dotazu

Jedním ze způsobů, jak vylepšit výsledky IR, je rozšíření dotazu (query expansion - QE). O rozsáhlém průkumu QE pojednává [AD19]. Jde o postup, který nám původní dotaz nějakým způsobem rozšíří a upraví, aby byl efektivnější pro následné použití v IR systému. Tento postup se skládá ze čtyř základních kroků:

1. Předzpracování a extrakce termínů z datových zdrojů
    - Před jakýmkoliv rozšiřováním je potřeba získat samotné termíny, kterými budeme dotaz rozšiřovat. Ty se získají z předem daných dat rozdělením na jednotlivá slova, odstraněním často používaných slov (předložky, spojky, ...), a nakonec stematizací (nalezení kmene) těchto slov.
2. Ohodnocení termínů vůči dotazu
    - V dalším kroku už použijeme jako vstup dotaz a předem získané termíny. Tyto termíny pak nějakým způsobem ohodnotíme, zda jsou relevantní vůči danému dotazu.
3. Výběr termínů
    - Z ohodnocených termínů musíme vybrat prvních *n* nejlepších, abychom je mohli přidat do rozšířeného dotazu. Autoři článku poznamenali, že na jejich počtu zdaleka nezáleží tolik, jako na jejich kvalitě.
4. Přeformulování dotazu
    - Tento krok je již volitelný. Zde můžeme nový dotaz změnit například úpravou vah jednotlivých termínů na základě vzájemných vztahů.

Metody QE lze rozdělit na dvě hlavní kategorie - lokální a globální. Lokální QE pracuje s předpokladem, že pokud nějaký termín dobře vystihuje dotaz, pak by ho měl termín podobný také dobře vystihovat. U lokálních QE tedy přiřazujeme nové termíny k těm původním z dotazu. Oproti tomu globalní QE pracuje s dotazem jako s jedním celkem.

Lokální QE lze dále rozdělit na dvě podkategorie - relevance feedback a pseudo-relevance feedback. Relevance feedback využívá zpětnou vazbu od uživatele, který po zadání dotazu a vyhodnocení výsledku určuje, jak jsou tyto výsledky relevantní. S touto informací navíc se pak aplikuje QE a vylepší se výsledek IR. Pseudo-relevance

feedback v podstatě jen vybere prvních *n* výsledků místo uživatele jakožto relevantní a dále funguje stejným způsobem. Relevance feedback by tedy měl být intuitivně kvalitnější, avšak vyžaduje interakci od uživatele navíc, což někdy může být nežádoucí.

Globální QE lze také rozdělit, a to na čtyři podkategorie s využitím:

- lingvistiky - relevantní termíny se určí na základě vztahů mezi slovy,
- jazykového korpusu - využití textových dat, kde se mohou také nacházet relevantní termíny blízko u sebe,
- historie vyhledávání - uživateli lze personalizovat QE dle jeho vyhledávací historie,
- webových dat - například hypertextové odkazy.

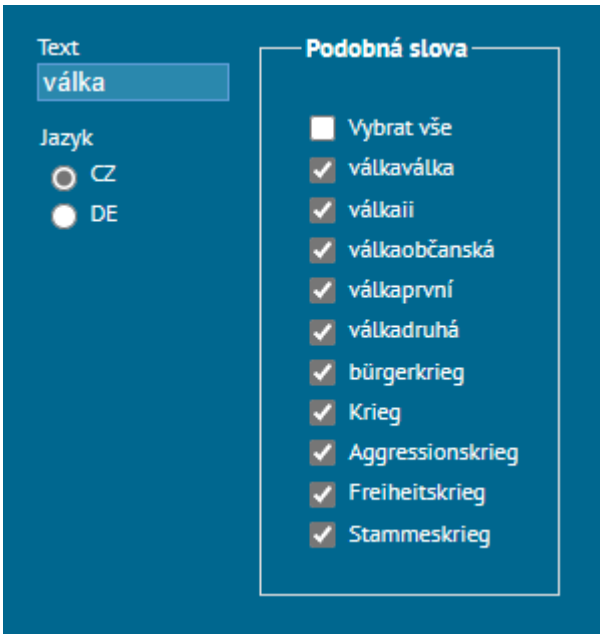
Pro účely CLIR lze také použít QE k vylepšení výsledků. Například u metody překladu dotazu lze aplikovat QE na dotaz před, nebo po překladu. Z experimentů bylo zjištěno, že lepší výsledky poskytuje aplikace QE před překladem. Avšak i aplikace po překladu dává lepší výsledky, než použití samotného překládání dotazu bez žádného QE.

2.2 Vyhledávací systém Porta fontium

Projekt Porta fontium je společný česko-bavorský archivní projekt, jehož hlavním cílem je digitalizace a zpřístupnění rozsáhlých archivních fondů souvisejících s historií a kulturou českého a bavorského pohraničí. Projekt koordinují Generální ředitelství bavorských státních archivů v Mnichově a Státní oblastní archiv v Plzni. Součástí archivních fondů jsou historické dokumenty v češtině a němčině, mezi kterými lze vyhledávat pomocí webového CLIR systému, který je součástí projektu.

Funkcionalita tohoto CLIR systému je primárně založena na integraci mezijazyčných slovních vektorů a vyhledávače Apache Solr. Mezijazyčné vektory umožňují reprezentaci slov z různých jazyků ve sdíleném vektorovém prostoru, díky čemuž lze měřit sémantickou podobnost mezi slovy jiných jazyků. V kontextu Porta fontium je tato technika použita pro hledání podobných slov v němčině a češtině na základě hledaného dotazu od uživatele.

Vyhledávač Solr je použitý k indexování a vyhledávání v historických dokumentech, které byly digitalizované pomocí optického rozpoznávání znaků (Optical Character Recognition - OCR). Když uživatel zadá dotaz například v češtině, tak systém nejdříve vygeneruje seznam podobných slov jak v češtině, tak v němčině, pomocí mezijazyčných slovních vektorů. Seznam těchto slov se vypíše v uživatelském rozhraní (2.1), a je zde možnost konkrétní slova z vyhledávání vynechat. Vyhledávač Solr



Obrázek 2.1: Příklad použití UI vyhledávacího systému Porta fontium

poté použije tato podobná slova k vyhledávání v indexovaných textech a nalezení relevantních dokumentů. Nejedná se tedy čistě o CLIR úlohu, ale také o „obyčejný" IR, jelikož systém nevyhledává pouze mezijazyčně (např. čeština → němčina), ale zároveň jednojazyčně (čeština → čeština)

Tento systém má aktuálně svá určitá omezení, která nepříznivě ovlivňují jeho výkon:

1. Nedostatečné zpracování víceslovných dotazů: současný design systému Porta fontium nedokáže dostatečně zpracovat víceslovné dotazy. Pokud uživatel zadá dotaz se dvěma nebo více slovy, systém vrátí pouze podobná slova ve stejném jazyce pro první dvě slova v dotazu a nesestaví podobná slova v druhém jazyce. Toto chování plyne z omezení systému, kdy je použito 10 podobných slov na jeden dotaz. Toto omezení může vést k neúplným nebo nerelevantním výsledkům vyhledávání.

2. Suboptimální kvalita podobných slov: Dalším významným omezením systému Porta fontium je nižší kvalita podobných slov, která generuje. Systém často vrací slova obsahující znaky, které nejsou součástí slov (např. "**,** **-**„") nebo slova, která jsou spojena nebo nemají význam (např. "válkaválka", "válkaii", "válkaobčanská", "válkaprvní", "válkadruhá"). Navíc v některých případech vrací různé tvary téhož slova, což pouze zabírá místo v omezeném seznamu podobných slov, aniž by přidávalo hodnotu k výsledkům hledání (např.

Tabulka 2.2: Příklady generovaných podobných slov pomocí systému Porta fontium

dotaz	podobná slova			
král	král, protikrál Normannenherzog	exkrál König	spolukrál Konigs Normannenkönig	kukrál Prinzen
pole	pole, position Rasterfeld	polí Feldes	poli feldern Getreidefeld	magnerické -Felder
Stadt	města metropole Stadtverwaltung	město_vesnice Metropole	předměstím Stadt.In Landeshauptstadt	městská Großstadt
Kirche	starokatolický církevpravoslavná Kirchengemeinde	kostel Kirchen	přesvěcen Pfarrkirche Dorfkirche	kostel_sv Kapelle

"pole", "polí", "poli"). Další příklady těchto problematických výsledků lze vidět v tabulce 2.2.

Pro řešení těchto problémů se nabízí dva způsoby - vylepšení stávajícího systému hledání podobných slov, nebo návrh jiného způsobu implementace CLIR. Hledání podobných slov by se dalo vylepšit například přidáním různých metod předzpracování a následného zpracování, použitím jiných slovních vektorů, nebo vylepšením mezijazyčné transformační matice. Celkový CLIR systém by bylo možné nahradit například neuronovou sítí natrénovanou přímo pro tento účel.

2.3 Datasety

2.3.1 CLIRMatrix

CLIRMatrix [SD20] je rozsáhlý dataset pro mezijazyčné vyhledávání v dokumentech. Obsahuje 139x138 jazykových párů pro dvojjazyčné vyhledávání a dataset s 8 jazyky pro vícejazyčné vyhledávání. Dataset byl shromážděn z článků z Wikipedie a dat z wikidata. Každá jazyková kombinace dat je rozdělena do tří množin - trénovací, validační a testovací, pro možnost přesnějšího srovnání různých metod CLIR.

Celý dataset lze v podstatě považovat za jednu velkou množinu trojic - dotaz v jazyce A, dokument v jazyce B a label. Label je v tomto případě skóre relevance daného dokumentu vůči danému dotazu. Jako dotazy byly jednoduše použity tituly článků. Ke každému dotazu v jazyce A bylo poté nalezeno 100 dokumentů v jazyce A pomocí Elasticsearch, které zároveň každému výsledku přiřadí BM25 skóre relevance. Toto skóre pracuje s dotazem a dokumenty pouze jako s množinami

slov/termínů (bag-of-words) a podobnost vypočítavá na základě četnosti termínů mezi dokumentem a dotazem a inverzní četností termínů napříč všemi dokumenty (inverzní - čím vzácnější je termín napříč všemi dokumenty, tím je významnější pro vyhledávání). Pro nalezení dokumentu v jazyce B byla použita wikidata, která poskytují odkazy na články v různých jazycích pro daný dokument. Samotné BM25 skóre je dále normalizováno a diskretizováno pomocí Jenks algoritmu na interval [1, 5] a výsledek byl použit jako label. Všechny dokumenty samozřejmě vždy nemají ve wikidatech odkazy pro všechny ostatní jazyky, proto je do trojic doplněno několik náhodných dokumentů, aby pro každý dotaz bylo v datech 100 trojic. Těmto náhodným dokumentům je pak přiřazen label 0. Label 6 je pak vyhrazen pro ten dokument, jehož titulek přímo odpovídá danému dotazu. Celkově je tedy label celé číslo na intervalu [0, 6]

2.3.2 WikiCLIR

WikiCLIR [Sch+14] je relativně menší CLIR dataset zkonstruovaný pomocí dat z článků Wikipedie. Obsahuje pouze jeden pár jazyků - dotazy v němčině (DE) a dokumenty v angličtině (EN). Jako dotazy zde byly použity první věty z DE článků. Tuto první větu lze ve většině případech považovat za dobře utvořenou definici tématu, o němž daný článek píše. Ze všech dotazů byla také odstraněna slova z nadpisu. Průměrná délka těchto dotazů pak byla 1,8 slov. Dataset přiřazuje míru relevance (3) příslušným EN článkům, které jsou odkázány k původnímu DE článku (ten, ze kterého byl získán dotaz) pomocí mezijazyčných odkazů Wikipedie a Wikidat. Je zde pak dále použita relevace (2), která je přiřazena těm EN článkům, které na sebe navzájem odkazují s EN článkem s relevancí (3) obecnými odkazy. Všechny ostatní EN články jsou pak považovány za nerelevantní.

2.3.3 Extended CLEF eHealth test collection

Tento dataset [SP19] nemá jednoznačný vlastní název jako ostatní, protože se jedná o rozšíření testovacích dat z lékařské domény vzniklých v rámci CLEF eHealh IR úloh z let 2013, 2014 a 2015. Jde tedy původně o jednojazyčné datasety, které byly rozšířeny překladem na mezijazyčné. Stejně jako ostatní je tento dataset složený z dotazů, dokumentů a ohodnocení relevancí dokumentů k dotazům.

Dokumenty byly získány z lékařských webových stránek v angličtině pomocí crawlerů. Výsledná kolekce obsahuje přibližné milion dokumentů, přičemž rozšířená verze používá tuto kolekci bez úprav. Tedy dokumenty jsou stále pouze v angličtině. Relevance těchto dokumentů vůči dotazům pak byla ručně ohodnocena znalci v lékařském oboru. Každý účastník dostal na výběr, zda je daný dokument k danému dotazu relevantní hodně, trochu, nebo vůbec.

Tabulka 2.3: Porovnání CLIR datasetů

dataset	počet				
	jazyků		dokumentů	dotazů	trojic
	dokumentů	dotazů			
WikiCLIR	1	1	1,2M	1,2K	1,2M
CLEF	1	8	1M	166	1,1K
BI-139	139	139	1,5M	1,3M	1,1B
MULTI-8	8	8	1,4M	1,4K	1,8M

Samotné dotazy byly v původních datech vytvořeny různými způsoby. V letech 2013 a 2014 byly formulovány lékařskými experty, kterým byly prezentovány skutečné propouštěcí zprávy a následně byli instruováni náhodně vybrat jednu chorobu a popsat ji. Tím přirozeně vznikla kolekce dotazů, která obsahuje více odborných termínů. Naopak v roce 2015, byly dotazy vytvořeny nekvalifikovanými dobrovolníky z řad studentů. Dobrovolníkům byly ukázány obrázky zdravotních problémů a poté byli instruováni vytvořit na jejich základě dotaz. Tím, v kontrastu s předchozími lety, vznikly dotazy které by mohl zadat i obyčejný laik. Rozšíření datasetu tedy spočívalo v tom, že se všechny dotazy sloučily do jedné kolekce a nechaly se ručně přeložit experty do sedmi evropských jazyků včetně češtiny a němčiny.

2.4 Metriky

Pro vyhodnocení CLIR metod lze použít stejné metriky, jako pro IR, jelikož jediný praktický rozdíl je v tom, že dotaz je v jiném jazyce, než dokument. Důležité je mít k dispozici testovací dataset, který každému páru dotazů a dokumentu přiřadí skóre relevance. Pro účely vyhodnocení metriky pak záleží pouze na relevantnosti výsledných dokumentů a ne na tom, v jakém jazyce jsou dotazy či dokumenty.

2.4.1 Přesnost

V kontextu IR je přesnost podílem počtu dokumentů ve výsledku, které jsou relevantní a počtu všech dokumentů ve výsledku. Lze zapsat množinově:

$$přesnost = \frac{|\{relevantní\ dokumenty\} \cap \{dokumenty\ ve\ výsledku\}|}{|\{dokumenty\ ve\ výsledku\}|} \tag{2.1}$$

Přesnost tedy pracuje s binárním skóre relevance - dokument buď je a nebo není relevantní k danému dotazu. Pro lepší srovnatelnost ji lze také vyhodnocovat vždy pouze nad prvními k dokumenty z výsledku jakožto *přesnost@k*, resp. *P@k*.

2.4.2 Úplnost

Protože přesnost nedokáže vyjádřit kolik ze všech relevantních dokumentů bylo úspěšně nalezeno, tak se používá také *úplnost*:

$$úplnost = \frac{|\{relevantní\ dokumenty\} \cap \{dokumenty\ ve\ výsledku\}|}{|\{relevantní\ dokumenty\}|} \tag{2.2}$$

Úplnost ale nelze použít samostatně, jelikož by bylo triviální dosáhnout 100% tím, že by se použily všechny dokumenty jakožto výsledek.

2.4.3 Průměrná přesnost

Průměrná přesnost (average precision - AP) je v podstatě kombinací přesnosti a úplnosti. Jedná se o průměr přesností v rámci všech relevantních dokumentů. Vypočte se následovně:

$$AP@n = \frac{\sum_{k=1}^{n} P@k \times rel_k}{|\{relevantní\ dokumenty\}|} \tag{2.3}$$

Kde $P@k$ je již zmíněná *přesnost@k* a $rel_k \in \{0, 1\}$ je binární skóre relevance k-tého dokumentu ve výsledku vůči vyhodnocovanému dotazu.

2.4.4 MAP

Střední průměrná přesnost (mean average precision - MAP) je metrika, pomocí které lze vyhodnotit průměrnou přesnost pro všechny dotazy a tím pádem pro celý dataset najednou. Vypočítá se jako jednoduchý průměr:

$$MAP = \frac{1}{Q} \sum_{q=1}^{Q} AP(q) \tag{2.4}$$

Kde $AP(q)$ je průměrná přesnost pro dotaz s indexem q a Q je počet všech dotazů.

2.4.5 CG

Cumulative gain je velmi jednoduchá metrika, která pracuje s určitým počtem p dokumentů z výsledku IR a jejich relevancí rel dle datasetu, která už nemusí být binární, ale může být jakékoliv číslo. Samotný CG je pak součet těchto relevancí:

$$CG@p = \sum_{i=1}^{p} rel_i \tag{2.5}$$

CG tedy nijak nepočítá s pořadím dokumentů a volba p je zde velmi důležitá.

2.4.6 DCG

Pokud budeme předpokládat, že relevantnější dokumenty se nacházejí dříve ve výsledku IR, pak můžeme vylepšit CG zakomponováním pořadí do výpočtu. Tím získáme discounted cumulative gain (DCG), která logaritmicky penalizuje dokumenty, které se nachází později ve výsledku:

$$DCG@p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)} \tag{2.6}$$

Existuje také často používaná, alternativní forma, která dává vetší váhu relevanci:

$$DCG@p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{2.7}$$

2.4.7 nDCG

Samotné DCG nelze dobře použít pro porovnávání výsledků, protože interval hodnoty této metriky je ovlivněn jak počtem vybraných dokumentů z výsledku p, tak hodnotami, které může nabývat relevantnost rel (což může být obecně jakékoliv reálné číslo). Nejčastěji se tedy používá normalizovaná varianta DCG (nDCG).

Pro výpočet je potřeba zjistit samotnou hodnotu $DCG@p$, která se bude normalizovat. Dále se vypočítá optimální (ideální) DCG v rámci datasetu - $IDCG@p$. Tu lze zjistit tak, že seřadíme všechny možné dokumenty k danému dotazu podle relevance a vybereme prvních p. Na těchto relevancích spočítáme $DCG@p$ stejným způsobem a tím získáme $IDCG@p$. Finální výsledek pak vypadá následovně:

$$nDCG@p = \frac{DCG@p}{IDCG@p} \tag{2.8}$$

Pro vyhodnocení nDCG napříč všemi dotazy z datasetu už stačí jen vypočítat průměr hodnot nDCG všech dotazů.

2.5 Závěr

Z analýzy různých metod a přístupů CLIR lze vidět, že je jich opravdu mnoho a v této práci se podařilo prozkoumat některé obecné a nějaké konkrétní, které by potenciálně mohly být užitečně pro následnou implementaci v Porta fontium. Samotný výběr tedy není zdaleka lehký. Efektivita těchto metod může záležet na konkrétním využití a na párech potřebných jazyků. Celkově se jedná o aktivní a stále velmi rychle se vyvíjející odvětví a je tedy možné, že během relativně krátké doby bude opět publikována nová metoda, která ostatní učiní zastaralé. Mohlo by

tedy být výhodnější snažit se zvolit spíše nějaké robustnější a dlouhodobější řešení něž to, které je zrovna v danou dobu považováno za state-of-the-art.

Dataset pro vyhodnocení byl zvolen CLIRMatrix hlavně díky velkému množství různých jazyků. Není však ale nutné použít všechny dostupné jazyky. Pomocí webového rozhraní lze vybrat jen konkrétní potřebné jazykové páry a ty využít, díky čemuž je dataset modulární. Také je zde průměrná délka dotazu 2 slova, což je nejblíže aktuálnímu používání CLIR systému v Porta fontium.

Hlavní metrikou byla zvolena nDCG. Z prozkoumaných publikací se zdá být tou nejpoužívanější, což indikuje kvalitu a zároveň dává možnost srovnat dosažené výsledky s více pracemi. Za druhou nejpoužívanější lze považovat MAP, kterou není problém také vyhodnotit pro lepší celistvost, i když pro porovnání bude primárně použita nDCG.

Experimenty 3

3.1 Otestování Porta fontium

V portálu Porta fontium je CLIR funkčnost integrovaná pomocí externího REST API. Toto API poskytuje funkce mezijazyčného rozšíření dotazu pomocí lineárních transformací slovních vektorů (popsáno v 2.1.2). Samotné rozšíření dotazu zde funguje tak, že se dotaz rozdělí na jednotlivá slova a ke každému slovu se poté najde určitý počet podobných slov v jiném jazyce. Všechna tato slova se pak spojí dohromady zpátky do jednoho (rozšířeného) dotazu. O část IR se stará vyhledávač Apache Solr. Solr umí vyhledávat nad přidanými dokumenty pomocí své vlastní dotazovací syntaxe, která například umožňuje logické operátory „a“ a „nebo“ mezi dotazovanými výrazy. Pro účely testování bude celý rozšířený dotaz rozdělen na slova a ta budou předána vyhledávači Solr jako jednotlivé výrazy s implicitním operátorem „nebo“ mezi nimi.

Testování je rozdělené na dvě části (CL a IR) - rozšíření dotazů z datasetu pomocí lokálně spuštěného Porta fontium QE (query expansion) API a následné dotazování pomocí lokální instance Apache Solr. Každá část má své jednotlivé kroky s vlastními mezivýsledky. Ukládat tyto mezivýsledky pouze do paměti by bylo nevýhodné, jelikož by takový způsob neumožnil jejich pozdější prohlédnutí a také by mohl způsobit ztrátu výpočetně drahých výsledků při selhání procesu. Je tedy lepší si tyto mezivýsledky ukládat nějakým způsobem perzistentně. Pro toto úložiště byla zvolena databáze sqlite. Tato databáze funguje bez serveru a k uložení dat používá pouze jeden soubor. I přes tato zjednodušení lze však stále používat užitečný dotazovací jazyk SQL.

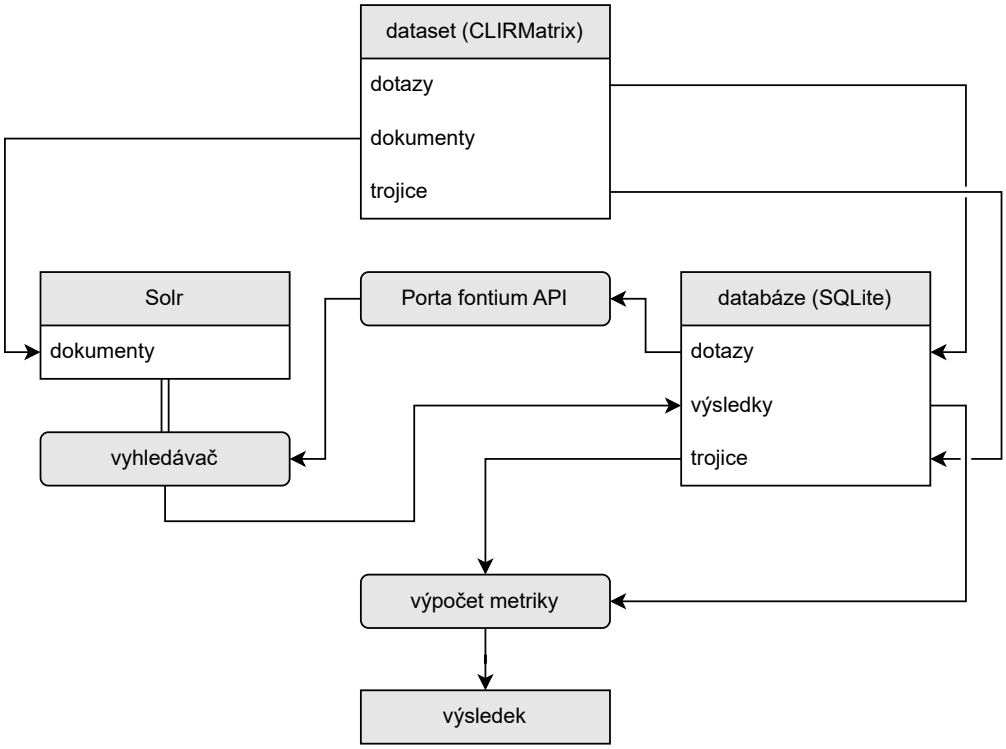
Celkový postup (3.1) je následující:

1. Založit sqlite databázi.
2. Nahrát dotazy z datasetů do databáze.
3. Načíst dokumenty z datasetu do lokální instance Apache Solr.

4. Přeložit a rozšířit datasetové dotazy pomocí lokální instance Porta fontium QE API a uložit výsledky. (Cross-lingual - CL)

5. Načíst přeložené dotazy z databáze, nad každým spustit vyhledávání Solr, a uložit výsledky. (Information retrieval - IR)

6. Nad výsledky vyhledávání vypočítat metriku nDCG.



Obrázek 3.1: Diagram postupu otestování Porta fontium

Tento postup má velkou výhodu v tom, že pro otestování jiných metod, které jen modifikují dotaz, stačí zaměnit pouze krok 4, přičemž kroky 1-3 stačí udělat pouze jednou při prvním testování a kroky 5-6 se pokaždé opakují. Pokud bychom testovali metodu, která implementuje i své vlastní vyhledávání (např. 2.1.4), pak bychom museli změnit také 5. krok.

Na závěr, metrika nDCG@100 pro Porta fontium vyšla 0,022, což je poměrně málo (rozsah nDCG je $[0, 1]$). Pro porovnání, pouhé přeložení dotazů pomocí překladače Google dosáhlo nDCG 0,218.

3.2 Porovnání metod předzpracování a následného zpracování při vyhledávání pomocí slovních vektorů

Pro mezijazyčné vyhledávání s využitím slovních vektorů je možné využít postup, který spočívá v nahrazení každého slova v dotazu N nejbližšími slovy, nalezenými jako vektory v transformovaných prostorech mezi jazykem dotazu a cílovým jazykem dokumentů. Výsledný dotaz může být pak použitý pro vyhledávání.

V rámci experimentu jsem použil slovní vektory z modelu fastText, před-trénované na článcích z Wikipedie, které jsou transformovány do společného prostoru angličtiny. Pro ověření jsem použil postup popsaný v sekci 3.1 na datasetu CLIRMatrix pro vyhledávání mezi češtinou a němčinou oběma směry.

Pro tento způsob vyhledávání lze využít různé metody předzpracování a následného zpracování jednotlivých slov dotazu. Jednou z těchto metod, kterou jsem použil v každém případě, je odstranění nežádoucích znaků (znaky které nejsou součástí slov přirozeného jazyka - podtržítka, čárky, pomlčky, závorky, apod.), jelikož dotazy v použitém datasetu jsou názvy článků z Wikipedie, které nemusí nutně obsahovat pouze slova a použité fastText vektory jsou trénované bez učitele, takže se v nich mohou vyskytovat nežádoucí znaky (např. slovo z modelu "`}>související`"). Dále jsem otestoval různé kombinace metod - lemmatizace (převod do základního tvaru) a stematizace (hledání kmene slova). Pro předzpracování jsem stematizaci vyřadil, protože kmen slova často není slovem sám o sobě a pravděpodobně se nebude nacházet v prostoru slovních vektorů. Dále jsem vyzkoušel varianty, kdy jsou všechna písmena převedena na malá nebo ponechána v původní kapitalizaci (case sensitivity). Kombinace těchto parametrů jsem nakonec testoval při hledání pomocí různých počtů podobných slov N (2, 5 a 10).

Dosažené výsledky lze porovnat v obrázku 3.2. První, zdánlivě hlavní věc, která z výsledků plyne je, že case-sensitive vyhledávání je obecně mnohem lepší, než vyhledávání s převodem na malá písmena. Bohužel, po pozdější analýze se ukázalo, že použité vektory obsahují všechna slova převedená na malá písmena, tudíž case-sensitive vyhledávání by v tomto případě nemělo ani dávat smysl. Otázkou tedy ale je, proč i přesto mělo case-sensitive vyhledávání lepší výsledky. Použitý algoritmus fungoval tak, že pokud ve vektorovém prostoru slovo nenašel (a nemohl tedy ani hledat podobná slova), pak použil pro výsledný „přeložený" dotaz alespoň ono původní slovo. Zároveň dataset CLIRMatrix používá názvy článků jakožto dotazy, jenomže tyto názvy jsou vždy s velkým písmenem na začátku prvního slova i v případě, pokud toto slovo není ve skutečnosti název. Například dotaz v tomto datasetu ke článku o jablku by byl název článku, tedy „Jablko" s velkým „J". Mimo to, je také použita pro porovnání němčina, kde se všechna podstatná jména píší s vel-

kým písmenem. Nelze tedy nijak jednoduše obecně rozlišit mezi obyčejnými slovy a názvy v rámci dotazů. Tím pádem může často nastat případ, kdy k nějakému názvu vyhledáme podobná slova v jiném jazyce, která jsou ale nežádoucí a nedávají smysl pro daný kontext, jelikož se jedná o název. V tomto datasetu tedy primárně v dotazech převažují názvy, u kterých jejich překlad může vyhledávání naopak uškodit a lepší výsledek dostaneme, pokud použijeme původní dotaz i v mezijazyčném vyhledávání. Výsledky z case-sensitive vyhledávání nejsou tedy z pohledu praktického CLIR natolik zajímavé, i když jsou lepší.

U parametru počtu podobných slov, se výsledky příliš neliší. I při porovnání velkého rozdílu parametru (2 a 10 podobných slov) jsou výsledky téměř stejné. Dokonce jsou výsledky tím lepší čím méně podobných slov hledáme. Tento rozdíl je však velmi malý a je možné, že je opět dán spíše problematickým datasetem. Pro průkaznější výsledky by bylo potřeba tento parametr porovnat nad jinými daty.

Nejzajímavější výsledky jsou v porovnání předzpracování a následného zpracování v rámci case-insensitive vyhledávání. Použití lemmatizace v předzpracování je zde v každém případě lepší než žádné. To naznačuje, že v rámci vektorů jsou slova zastoupena hlavně v základních tvarech a že se vyplatí převod do základního tvaru, i když ztratíme ten původní, pokud nám to umožní najít více, nebo lepší podobná slova. U následného zpracování je nejhorší stematizace, to může být dáno tím, že se vyhledávač Solr neumí vypořádat s kořeny slov a vyhledává lépe, pokud má k dispozici slova, která mají samy o sobě význam. Lemmatizace je zde opět o něco lepší než nic, avšak rozdíl už není zdaleka tak velký jako u předzpracování.

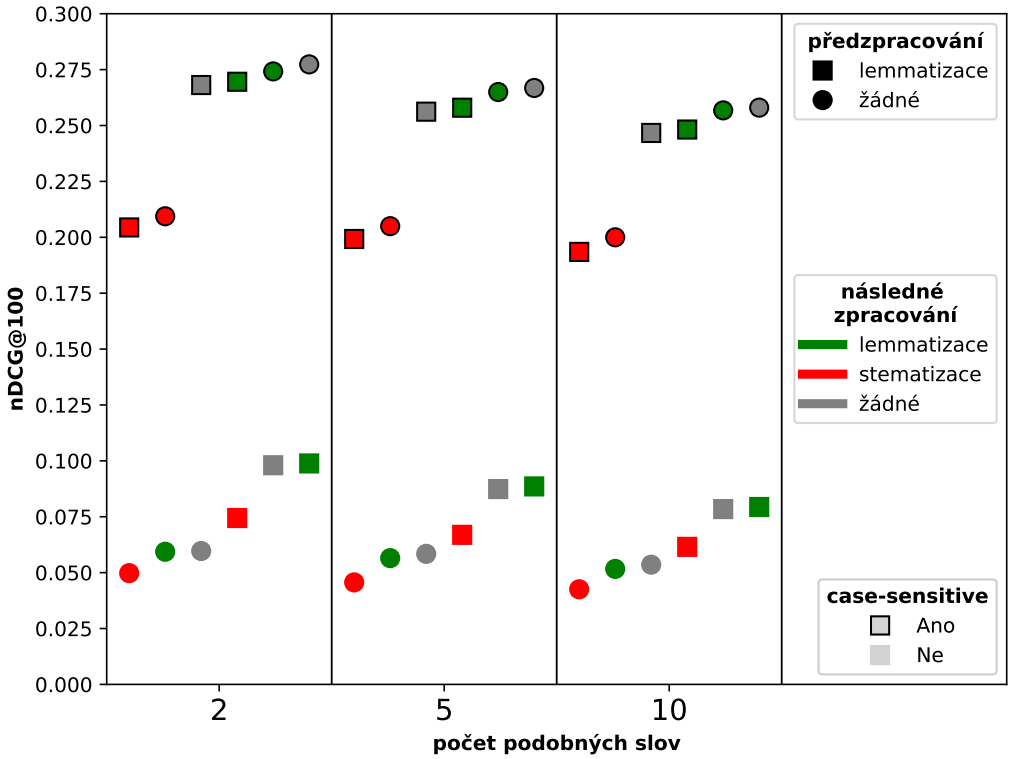
3.2.1 Návrh na vylepšení Porta fontium

Porta fontium aktuálně používá v zásadě stejnou metodu, jejíž varianty byly testované v tomto experimentu. Díky tomu lze na základě těchto výsledků vyvodit návrhy pro implementačně jednoduchá vylepšení.

Vzhledem k nedostatkům popsaným v sekci 2.2, by se například vyplatilo použít stejnou metodu odstranění nežádoucích znaků (tedy ponechat ve slovech pouze písmena), kterou jsem použil zde při zpracování slov před i po generování podobných slov. Tím se odstraní různé nedostatky vygenerovaných slov, způsobené použitím vektorů natrénovaných nad daty z internetu.

Dalším hlavním přínosem by bylo přidání lemmatizace, která dle výsledků u předzpracování zlepšuje kvalitu podobných slov a u následného zpracování eliminuje různé varianty stejného slova. Tyto varianty jinak zabírají místo ve výpisu podobných slov a dle výsledků experimentu nepřidávají samotnému vyhledávání pomocí Solr žádnou hodnotu.

Nakonec to, zda použít case-insensive vyhledávání záleží výhradně na tom, zda použité slovní vektory obsahují pouze malá, nebo i velká písmena.



Obrázek 3.2: Vizualizace porovnání předzpracování a následného zpracování metod při vyhledávání pomocí slovních vektorů nad CLIRMatrix. Tvar bodů značí metody předzpracování, barva značí metodu následného zpracování a obrys značí zda bylo vyhledávání case-sensitive. Osa X udává rozřaření do kategorií podle počtu vyhledáváných podobných slov a v rámci jedné kategorie vyjadřuje seřazení podle metriky nDCG@100 pro vyšší přehlednost grafu. Osa Y vyjadřuje metriku nDCG@100.

Mimo tyto metody by jako složitější vylepšení bylo možné natrénovat vlastní slovní vektory nad stejnými dokumenty, nad kterými probíhá vyhledávání. Tím by se také zlepšila kvalita podobných slov, jelikož by všechna tato slova pocházela z vyhledávaných dat. Je však nutné mít na paměti, že textové verze dokumentů v Porta fontium jsou vytvořené pomocí optického rozpoznávání znaků a nemusí být vždy správně převedené na text. Před trénováním by tedy bylo nejspíš potřeba zvolené dokumenty ručně opravit.

3.3 Vyhledávání pomocí dokumentových vektorů

V tomto experimentu jsem se pokusil pro CLIR využít dokumentové vektory, založené na modelu BERT. Velkou výhodou tohoto způsobu, a také důvodem pro tento

experiment, je jeho jednoduchost. Metodu jsem otestoval opět nad CLIRMatrix, konkrétně mezi jazykovými páry čeština-němčina za pomocí metriky nDCG@100. Postup testování opět vycházel z experimentu 3.1.

Použil jsem „bert-base-multilingual-cased“ model, což je předtrénovaný BERT model, který podporuje více jazyků (včetně češtiny a němčiny) a používá case-sensitive slovní zásobu. Nejdříve byly všechny dokumenty najednou tokenizovány a zakódovány pomocí BERT tokenizeru, a výsledné sekvence byly zpracovány modelem. Výsledkem byly kontextové vektory pro jednotlivé tokeny, tyto vektory byly následně zprůměrovány a konečný vektor s konstantní dimenzí by měl teoreticky reprezentovat význam vstupního textu. Tímto postupem vznikl vektorový prostor dokumentů, ve kterém lze následně vyhledávat. Pro konečné testování vyhledávání jsem na jednotlivé dotazy použil stejný postup jako na dokumenty a výstupem byl vektor reprezentující význam daného dotazu. Pomocí tohoto vektoru jsem pak vyhledával nad vytvořeným prostorem dokumentů pomocí nejbližší kosinové vzdálenosti.

Výsledky tohoto experimentu byly špatné, s nDCG@100 hodnotou 0,01. Je však možné, že toto nízké skóre může být dáno povahou použitého CLIRMatrix datasetu, který obsahuje velice krátké dotazy, které teoreticky často nemusí dobře zachytit význam cílového dokumentu a nemusí být tím pádem vhodné pro vyhodnocování významových vektorů na úrovni dokumentů. Pro otestování této hypotézy by však bylo potřeba hlubšího průzkumu.

I přes špatné výsledky v tomto případě, je tento způsob potenciálně slibný pro jiný typ dat. Je to jednoduchá a efektivní metoda, která nevyžaduje další trénovací data a komplexní optimalizační postupy, díky čemuž je snadná na použití a případné nasazení v reálných aplikacích. Budoucí práce by mohla prozkoumat alternativní před-trénované modely, fine-tuning techniky a podobnostní metriky pro zlepšení výsledků této metody pro CLIR.

3.4 ColBERT-X

V posledním experimentu jsem otestoval model ColBERT-X, na kterém je založen implementovaný softwarový modul (4). Tento model je blíže popsán v sekcích 4.1 a 4.2. Stručně jde o model, který je založen na neuronové síti, která řeší CLIR úlohu jako celek, na rozdíl od jiných metod, které úlohu rozdělovaly na dvě části. V tomto experimentu tedy není potřeba externího vyhledávače a je strukturou velmi podobný 3.3.

Tuto metodu se bohužel nepovedlo porovnat za přesně stejných podmínek, jako ostatní. ColBERT-X totiž pro vyhledávání vytváří index, který je velice náročný na úložiště a bylo tedy nutné zredukovat kolekci vstupních dokumentů, aby obsahovala pouze dokumenty použité v testovacích dotazech. Tato změna metodu zvýhodňuje,

Tabulka 3.1: Souhrn výsledků vyzkoušených metod v rámci experimentů nad datasetem CLIRMatrix. *ColBERT-X byl porovnán za jiných podmínek, viz 3.4

metoda	**nDCG@100**
**ColBERT-X*	0,3643
fasttext podobná slova s lematizací	0,2289
Porta fontium retrieval API	0,0267
dokumentové vektory	0,0099

jelikož má k dispozici méně dokumentů, ze kterých musí vyhledávat. Model je také potřeba trénovat na jiném typu dat, kde je ke každému dotazu přiřazen relevantní a nerelevantní dokument. Musel jsem tedy převést trénovací data CLIRMatrix (kde je ke každému dotazu a dokumentu přiřazena relevance 0 až 6) tak, že jsem zvolil práh relevance 3. Tato změna naopak metodu znevýhodňuje, protože model není naučený určit správné pořadí relevantních dokumentů.

Výsledné skóre nDCG@100 tohoto experimentu je 0,36, což je nejvyšší dosažené skóre. Kvůli zmíněným změnám se však nejedná o přímé porovnání, ale i přes to jde o slibný výsledek vzhledem k použití problematického datasetu.

3.5 Závěr

V rámci experimentů bylo potvrzeno, že CLIR výkon systému Porta fontium není moc vysoký. Zároveň byly prozkoumány potenciální vylepšení pomocí různých předzpracování a následných zpracování podobných slov. Zde se ukázalo, že nejlepší přístup přináší lemmatizace. Z experimentů s neuronovými modely vyplynulo, že pro smysluplné výsledky nestačí použít obecný jazykový model, ale je potřeba neuronovou síť natrénovat přímo pro úlohu CLIR. Celkové srovnání výsledků nad společným datasetem CLIRMatrix je vypsané v tabulce 3.1.

Modul pro vícejazyčné vyhledávání

4

Pro reálnou implementaci softwarového modulu jsem vybral přístup řešení celého problému v jednom kroku pomocí neuronové sítě. Oproti metodám rozšiřování/překladu dotazu s využitím mezijazyčných slovních vektorů v kombinaci s vyhledávačem má tento přístup několik výhod:

- Vyšší míra porozumění významu: neuronové sítě mohou zachytit složitější významové vztahy mezi jednotlivými slovy, nežli slovní vektory, které se zaměřují pouze na jednotlivá slova. Toto umožňuje lepší porozumění kontextu dotazu a může vést k relevantnějším výsledkům.
- Zpracování dvojích významů: neuronové sítě jsou lépe vybaveny pro porozumění dvojím významům, jelikož mají možnost zpracovat celý kontext dotazu a tím pádem i rozeznat, o který konkrétní význam se jedná. Slovní vektory oproti tomu každé slovo přiřazují pouze k jednomu bodu v prostoru, což nemusí vždy zachytit všechny významy daného slova.
- Nezávislost na jazyce: pokud využijeme jeden z mezijazyčných předtrénovaných modelů BERT (např. `bert-base-multilingual-cased`, nebo `xlm-roberta-base`, které jsou trénované na 100 a více jazycích), nemusíme se zabývat přesně, které jazykové kombinace chceme použít a model nám poskytne jednotnou reprezentaci textu napříč více jazyky.
- Snazší zpracování víceslovných dotazů: náročnost vyhledávání kvalitních podobných slov pomocí slovních vektorů je tím vyšší, čím delší máme dotaz. Je větší pravděpodobnost, že se vygeneruje slovo, které s dotazem nesouvisí a které může negativně ovlivnit relevantnost vyhledávání. Naopak při zpracování dotazu neuronovou sítí platí, že čím více kontextu je k dispozici, tím je dosažena vyšší úroveň porozumění dotazu.

I přesto má přístup s využitím mezijazyčných vektorů své výhody, jako jsou nižší výpočetní nároky a jednodušší implementace. Avšak vzhledem k aktuálním nevýhodám v systému Porta fontium jsem pro účely této práce vybral přístup využívající neuronovou síť, konkrétně předtrénovaného modelu BERT.

Implementovat model BERT pro CLIR od nuly by bylo velice náročné a mimo rozsah této práce, proto jsem se rozhodl jít cestou modifikace již existujícího modelu. Model navrhovaný v [Jia+20] bohužel nemá k dispozici žádné zdrojové kódy, které by se daly použít jako základ. Podařilo se mi však najít jediný model, který využívá BERT pro CLIR - ColBERT-X [Law+22]. Samo o sobě se jedná o modifikovaný IR model ColBERT [KZ20]. V této kapitole představím softwarový modul založený na modelu ColBERT-X, který lze integrovat do stávajícího systému pro vyhledávání informací, jako je Porta fontium.

4.1 ColBERT

Model ColBERT implementuje přístup hustého vyhledávání (dense retrieval), které reprezentuje vyhledávané dokumenty jako husté vektory. Husté vektory mají zpravidla méně dimenzí a obsahují menší počet nulových prvků na rozdíl od řídkých vektorů. Tento model byl navržený za účelem řešení kompromisu mezi efektivitou a účinností v IR úlohách. ColBERT je postaven na architektuře BERT a využívá mechanismus pozdní interakce, který má za cíl minimalizovat množství výpočtů přes BERT, bez významného ovlivnění IR výkonu.

Klíčovou inovací modelu ColBERT je strategie pozdní interakce, která v počáteční fázi odděluje zpracování dotazů a dokumentů. Model nejdříve najednou vypočítá vektorové reprezentace všech dokumentů, čímž značně snižuje výpočetní zatížení při vyhledávání. Interakce mezi vektory dotazů a dokumentů je odložena až na konečnou fázi procesu, což umožňuje efektivní vyhledávání v rozsáhlé kolekci dokumentů.

Model ColBERT se skládá ze dvou hlavních komponent: komponenta pro tvorbu vektorů a komponenta pro interakci. První komponenta je zodpovědná za generování hustých kontextových vektorů pro každý token v dotazu, či dokumentu. Toho je dosaženo aplikací modelu na tokeny dotazu a dokumentu nezávisle. Výsledné vektory jsou pak agregovány pomocí operace max-pooling, která vybere maximální hodnotu pro každý rozměr vektorů.

V komponentě interakce jsou agregované vektory dotazu a dokumentu kombinovány k výpočtu podobnostního skóre. Tento krok se nazývá *pozdní* interakce, jelikož se uskutečňuje až poté, co byly tyto vektory nezávisle vypočítány a agregovány. Podobnostní skóre je poté použito pro řazení dokumentů při trénování a vyhledávání.

Na povrchu se může tento způsob zdát podobný, jako v experimentu 3.3. Avšak ColBERT má dva zásadní rozdíly. První rozdíl je v tom, že přidává navíc krok trénování, který modifikuje model BERT přímo pro účely IR. Dále ColBERT využívá knihovnu FAISS, která je určená pro efektivní vyhledávání v hustých vektorech.

Proces trénování ColBERT spočívá v modifikaci modelu BERT na specifickém IR datasetu. Model používá vypočítanou podobnost mezi dokumentem a dotazem jako zástupnou hodnotu pro relevanci a je trénován na základě nákladové funkce, která řadí relevantní dokumenty nad nerelevantní. Optimalizací této funkce se ColBERT učí generovat takové vektory, pomocí kterých lze efektivně vyhledávat relevantní dokumenty na základě zadaného dotazu.

ColBERT také využívá FAISS (Facebook AI Similarity Search) knihovnu k efektivnímu vyhledávání na základě podobnostního skóre mezi agregovanými vektory dotazu a dokumentů. Po procesu trénování se vypočítají vektory dokumentů, které jsou pak indexovány pomocí FAISS knihovny. Po zadání dotazu se nejdříve vypočítá vektor reprezentující tento dotaz a následně se použije FAISS, který vykoná přibližné vyhledávání nejbližších sousedů v indexovaných dokumentech. Toto vyhledávání identifikuje dokumenty s nejvyšším podobnostním skóre k dotazu, které jsou zároveň považovány za nejrelevantnější. Využití této metody umožňuje modelu ColBERT vyhledávání ve velkém měřítku.

ColBERT prokázal svoji účinnost v různých testech pro získávání informací, jako je sada dat pro řazení dokumentů MS MARCO, kde překonává tradiční IR modely, jako je BM25 a zároveň snižuje výpočetní nároky.

4.2 ColBERT-X

Model ColBERT-X je přizpůsobením původního modelu hustého vyhledávání ColBERT pro účely vyhledávání ve vícejazyčném prostředí (CLIR). Zatímco původní model používá jednojazyčný enkodér BERT pro reprezentaci dotazů i dokumentů, ColBERT-X vylepšuje model zahrnutím vícejazyčného enkodéru, jako je XLM-R. Toto vylepšení umožňuje modelu zvládnout úkoly vyhledávání informací v různých jazycích. Mimo to si model ColBERT-X zachovává základní strukturu vyhledávání v hustých vektorech s pozdní interakcí.

Důležitým faktorem pro fázi trénování je to, že model nepoužívá klasickou strukturu CLIR trénovacích dat, jako v sekci 2.3.3, tedy trojice dokument - dotaz - relevance. Model využívá jiný typ trojic – dotaz – pozitivní výsledek – negativní výsledek, kde se ke každému dotazu přiřazuje jeden text, který je relevantní a reprezentuje tedy dokument, který by se měl objevit ve výsledku a jeden nerelevantní text, který by naopak ve výsledku být neměl.

4.3 Vlastní úpravy

Softwarový modul založený na zdrojovém kódu modelu ColBERT-X, je navržen tak, aby mohl být použit v reálných aplikacích. Pro dosažení tohoto cíle byly provedeny následující modifikace původního zdrojového kódu ColBERT-X:

Přidání seznamu použitých balíčků a kompilačního skriptu.
Byl přidán seznam použitých balíčků a skript, který je schopen kompilovat tyto balíčky do souboru `requirements.txt`, obsahujícího přesné verze všech závislostí. S použitím tohoto skriptu je také možné kompilovat balíčky pro zařízení bez GPU.

Opravy nutné pro běh modelu na lokálním stroji.
Původní zdrojový kód ColBERT-X byl v podstatě ve verzi alpha, což si vyžádalo několik oprav, aby bylo možné model spustit na lokálním stroji.

Úpravy pro podporu spuštěni na CPU.
Model nebylo možné spustit na počítači bez GPU s podporou CUDA. Tuto funkcionalitu tedy bylo potřeba dodělat. Pro proces trénování by tato funkce byla vcelku zbytečná, jelikož trénování bez GPU je příliš pomalé. Ale pro vyhledávání stačí spustit model na CPU, protože není tak výpočetně náročné a lze tím potenciálně ušetřit náklady za server.

Parametrizace předtrénovaného modelu.
Byla přidána parametrizace předtrénovaného modelu XLM-R, což umožnilo použití základního modelu `xlm-roberta-base`, jelikož jsem neměl k dispozici stroj s dostatečnou kapacitou VRAM pro velkou verzi modelu `xlm-roberta-large`.

Oprava načítání datasetu.
V původní implementaci byly trojice, dotazy a dokumenty načítány v různých formátech. Byly provedeny úpravy, aby byly všechny načítané datasety ve formátu `.tsv` pro lepší konzistenci. Navíc jsem přidal funkčnost pro načítání dotazů a dokumentů s vlastními ID. ColBERT-X totiž sám o sobě umí pracovat pouze s ID, které se shodují s pořadím od nuly. Toto usnadňuje použití různorodých dat bez nutnosti předzpracování.

Dokumentace důležitých parametrů.
Byla vytvořena dokumentace všech důležitých parametrů používaných v různých příkazech a jejich vlivu na procesy trénování, indexování a vyhledávání.

Přidání více metadat k vyhledávacímu indexu.

Do vyhledávacího indexu jsem přidal více metadat, což umožňuje snazší použití indexu, bez znalosti parametrů použitých pro trénování.

Přidání REST API.
Bylo přidáno REST API, které umožňuje použití funkce vyhledávání v reálném čase.

4.4 REST API

Důležitou součástí modulu je REST API, které s využitím indexu vytvořeného pomocí ColBERT-X dokáže vyhledávat v dokumentech v reálném čase. Díky architektuře REST lze tento modul snadno zakomponovat do již existující aplikace.

API je zabezpečené pomocí *basic access authentication*. Pokud chce uživatel API používat, musí do HTTP hlavičky `Authorization` vložit řetězec "`Basic <encoded_auth>`", kde `<encoded_auth>` je řetězec "`<user>:<password>`" zakódovaný metodou Base64.

API má k dispozici dva koncové body k ověření dostupnosti:

- `/ping` - není zabezpečený a lze tedy pomocí něj ověřit dostupnost i bez znalosti uživatelského jména a hesla
- `/ping/secured` - zabezpečená verze `/ping`, sloužící k ověření správného nastavení

API dokáže vyhledávat ve více indexech. Pokud si jich tedy uživatel vytvořil více (například pro různé jazyky dokumentů) a chce vyhledávat v každém z nich zvlášť, pak není nutné API spouštět vícekrát. K této hlavní funkci slouží koncový bod `/retrieve/<index_name>`, jehož parametry jsou popsané v tabulce 4.1

Tabulka 4.1: Seznam parametrů pro REST API koncový bod `/retrieve/<index_name>`

parametr	vysvětlení
`index_name`	název indexu ve kterém bude probíhat vyhledávání - prakticky se jedná o název složky
`query`	text dotazu
`depth`	hloubka vyhledávání - kolik dokumentů bude koncový bod vracet ve výsledku (výchozí hodnota: 100)

(tabulka pokračuje na další stránce)

Tabulka 4.1 *(pokračování z předchozí stránky)*

parametr	**vysvětlení**
`include_documents`	zda má koncový bod ve výsledcích vracet text dokumentů, pro použití tohoto parametru je nutné mít ve složce s indexem soubor s kolekcí dokumentů, možné hodnoty: 0, 1 (výchozí hodnota: 0)

Parametry jsou zadávány pomocí URL parametrů, tedy například: `/retrieve/index?query=test&include_documents=1&depth=5`

Výstup je vracen ve formátu JSON v následujícím tvaru:

```
[
    {
        "documentId": <id dokumentu>,
        "score": <podobnostní skóre>,
        "rank": <pořadí>,
        "text": <text dokumentu (include_documents=1)>"
    },
    {
        ...
    },
    ...
]
```

Před spuštěním API serveru je nutné upravit nastavení. Toto nastavení je uložené ve formátu JSON v souboru `settings.json`. Soubor má následující strukuturu:

```
{
    "server_host": <hostname serveru>,
    "server_port": <port serveru>,

    "basic_auth_user": <uživatelské jméno zabezpečení>,
    "basic_auth_password": <heslo zabezpečení>,

    "retrieve_index_root":
        <cesta ke kořenovému adresáři s indexy>
}
```

Pokud chce uživatel využívat možnost vracení textů dokumentů pomocí parametru `include_documents` , je nutné vložit do složky s daným indexem (`<retrieve_index_root>/<index_name>`) kolekci s dokumenty jako soubor `collection.tsv`. Použití tohoto parametru ale významně nezvyšuje paměťovou náročnost serveru, jelikož se texty dokumentů načítají až ve chvíli, kdy je vyhledávání

hotové a pouze ty, které byly nalezeny. Server tedy nenačítá do paměti celou kolekci dokumentů, ale pouze jejich pozice v souboru pro pozdější načtení.

4.5 Příkazy

V rámci jednotlivých kroků trénování, indexování a vyhledávání se používají příkazy spouštěné pomocí interpretu Python. V této sekci budou popsány jak se tyto příkazy spouští, k čemu slouží a jejich vstupy a výstupy.

4.5.1 `train`

```
$> python -m xlmr_colbert.train
```

Tento příkaz slouží ke spuštění trénování nad vstupním datasetem.

Tabulka 4.2: Seznam argumentů pro příkaz `train`

argument	vysvětlení
`--root`	kořenová složka, kam se budou ukládat data ze všech použitých příkazů (výchozí hodnota: `"experiments"`)
`--experiment`	složka, kam se budou ukládat data pro všechny příkazy při aktuálním kontextu (výchozí hodnota: `"dirty"`)
`--run`	složka, kam se budou ukládat výsledná data a metadata pro konkrétní spuštění aktuálního příkazu (výchozí hodnota: časové razítko)
`--similarity`	metrika, podle které se počítá podobnostní skóre, možné hodnoty - `"cosine"` (kosínová) a `"l2"` (euklidovská) (výchozí hodnota: `"cosine"`)
`--dim`	rozměr výsledných agregovaných vektorů dokumentů a dotazů, nižší hodnota ušetří paměť, ale může vést ke zhoršení kvality výsledků (výchozí hodnota: 128)
`--query_maxlen`	maximální délka dotazu, nižší hodnota ušetří paměť, ale může vést ke zhoršení kvality výsledků u delších dotazů (výchozí hodnota: 32)

(tabulka pokračuje na další stránce)

Tabulka 4.2 (*pokračování z předchozí stránky*)

argument	**vysvětlení**
`--doc_maxlen`	maximální délka dokumentu, nižší hodnota ušetří paměť, ale může vést ke zhoršení kvality výsledků u delších dokumentů (výchozí hodnota: 180)
`--bsize`	velikost batche - kolik dokumentů se v rámci trénování zpracovává najednou, vyšší hodnota může urychlit trénování, ale je náročnější na paměť (výchozí hodnota: 32)
`--base_model`	použitý vícejazyčný BERT model, možné hodnoty - `xlm_roberta_large` (větší, kvalitnější, paměťově náročnější) a `xlm_roberta_base` (výchozí hodnota: `"xlm_roberta_base"`)
`--queries`	cesta k souboru s dotazy z trénovacího datasetu ve formátu tsv - `<id>\t<text>`
`--collection`	cesta k souboru s dokumenty z trénovacího datasetu ve formátu tsv - `<id>\t<text>`
`--triples`	cesta k souboru s trojicemi z trénovacího datasetu ve formátu tsv - `<q_id>\t<pd_id>\t<nd_id>`, kde `q_id` je `id` dotazu z `queries` a `pd_id` a `nd_id` jsou `id` pozitivního (relevantního) dokumentu a negativního (nerelevantního) dokumentu z `collection`

Metadata pro tento i ostatní příkazy se ukládají do složky `<root>/<experiment>/<název příkazu>.py/<run>`

Může se stát, že trénování selže kvůli nedostatku video RAM. V takovém případě je potřeba poupravit parametry `dim`, `query_maxlen`, `doc_maxlen`, `bsize` a `base_model`.

Výsledkem tohoto procesu je natrénovaný model, který se uloží do souboru `<root>/<experiment>/train.py/<run>/colbert.dnn`

Pro natrénování na vlastních dokumentech, například pro účely vyhledávání historických dokumentů v Porta fontium, by bylo nutné si tyto dokumenty anotovat pomocí ručně psaných trénovacích dotazů. Nejdříve je tedy potřeba vymyslet dotazy, které se informační hodnotou hodí k dané kolekci dokumentů a které se zároveň budou podobat stylem a délkou takovým dotazům, které se budou reálně používat

při vyhledávání. Ke každému dotazu se poté vybere co nejvíce dvojic relevantních a nerelevantních dokumentů a tím vznikne vhodná datová sada pro trénování, díky které pak bude model potenciálně lépe vyhledávat pro vlastní, specifické účely.

4.5.2 index

```
$> python -m xlmr_colbert.index
```

Tímto příkazem vytvoříme index dokumentů, ve kterém bude možné později vyhledávat. Vstupem je natrénovaný model a kolekce dokumentů. Prakticky to znamená, že se zpracují všechny dokumenty pomocí modelu a výsledky se uloží do několika souborů po částech, podle velikosti kolekce.

Tabulka 4.3: Seznam argumentů pro příkaz `index`

argument	**vysvětlení**

`--root`
`--experiment`
`--run`
`--bsize`
`--similarity`
`--query_maxlen`
`--doc_maxlen`
`--collection`
	viz 4.2

`--checkpoint`	výsledný natrénovaný model pomocí příkazu `train`
`--chunksize`	samotný index bude rozdělen do více souborů po částech, toto je velikost jedné této části (v GiB) (výchozí hodnota: 6.0)
`--index_root`	složka pro ukládání všech vytvořených indexů
`--index_name`	název vytvářeného index (bude odpovídat názvu složky uvnitř `index_root`)

Výsledkem jsou všechny soubory, které jsou potřebné pro další krok, uloženy ve složce `<index_root>/<index_name>`.

4.5.3 index_faiss

```
$> python -m xlmr_colbert.index_faiss
```

Aby byl vytvořený index reálně použitelný, je nutné natrénovat a uložit FAISS index pro rychlé vyhledávání. Tento příkaz je nejvíce přímočarý ze všech. Vstupem je pouze cesta k vytvořenému indexu a výsledný FAISS index se uloží na stejné místo. Tím je index hotový a lze ho využít k plnohodnotnému vyhledávání pomocí vytvořeného API, nebo příkazu `retrieve`.

Tabulka 4.4: Seznam argumentů pro příkaz `index_faiss`

argument	vysvětlení

`--root`
`--experiment`
`--run`
	viz 4.2

`--index_root` `--index_name`	viz 4.3
`--sample`	číslo mezi 0.0 a 1.0, určuje na jak velké části dokumentů z indexu bude FAISS index trénován (výchozí hodnota: 1.0)

4.5.4 retrieve

```
$> python -m xlmr_colbert.retrieve
```

Tento příkaz slouží k offline vyhledávání pomocí více dotazů najednou. Vstupem je vytvořený index a soubor z dotazy. Výstupem je `.tsv` soubor s vyhledávacími výsledky ve formátu `<id dotazu>\t<id dokumentu>\t<pořadí ve vyhledávání>`

Tabulka 4.5: Seznam argumentů pro příkaz `retrieve`

argument	vysvětlení

`--root`
`--experiment`
`--run`
	viz 4.2

(tabulka pokračuje na další stránce)

Tabulka 4.5 (*pokračování z předchozí stránky*)

argument	**vysvětlení**

`--bsize`
`--similarity`
`--query_maxlen`
`--doc_maxlen`	

`--checkpoint`
`--index_root`
`--index_name`
	viz 4.3

`--queries`	dotazy, nad kterými se spustí vyhledávání, formát viz 4.2
`--faiss_depth`	počet výsledných dokumentů vyhledávaný algoritmem FAISS, mocnina dvou (výchozí hodnota: 1024)
`--depth`	konečný počet vyhledaných dokumentů, musí být ≤ `faiss_depth` (výchozí hodnota: 1000)
`--log_scores`	zda se do výsledků budou zapisovat i podobnostní skóre

4.5.5 retrieve_api

```
$> python -m retrieve_api
```

Vytvořené REST API lze jednoduše spustit pomocí tohoto příkazu. Žádné argumenty zde nejsou, jelikož veškeré nastavení je uložené v souboru `settings.json` popsaném v sekci 4.4.

4.6 Skripty

4.6.1 passage_dataset_from_qrels.py

```
$> python scripts/passage_dataset_from_qrels.py \\
    --qrels <qrels_path> \\
    --treshold <treshold>
```

Pomocí tohoto skriptu lze převést dataset CLIRMatrix typu trojice „dotaz - dokument - relevance“ na dataset typu trojice „dotaz - relevantní dokument - nerelevantní dokument“. Při převodu je nutné určit, zda je dokument relevantní, nebo ne -

k tomu slouží parametr `treshold`, hodnoty relevance větší než `treshold` jsou považovány za relevantní, a ostatní hodnoty jsou považovány za nerelevantní. Na vstupu je cesta k původnímu datasetu - `qrels_path` a výsledek je zapsán do stejné složky pod názvy `<puvodni_nazev>.triples.tsv` a `<puvodni_nazev>.queries.tsv`.

4.6.2 reduce_document_collection.py

```
$> python scripts/passage_dataset_from_qrels.py \\
    --qrels <qrels_path> \\
    --documents <documents_path>
```

Tento skript slouží k zjednodušení kolekce dokumentů v datasetu typu CLIR-Matrix podle vstupních trojic. Můžeme tím zredukovat dokumenty ze souboru `documents_path` tak, aby zde byly pouze ty, které jsou použité v trojicích `qrels`. Výstup se zapíše vedle vstupní kolekce s názvem odvozeným od vstupních souborů.

Závěr 5

V rámci této bakalářské práce jsem se zaměřil na analýzu a porovnání různých metod CLIR, aby bylo možné identifikovat nejvhodnější přístupy pro zlepšení vyhledávání v rámci projektu Porta fontium. Při hodnocení těchto metod byly provedeny experimenty, které přinesly užitečné poznatky pro následnou implementaci vyhledávacího modulu.

Pro vylepšení Porta fontium by bylo možné přímo použít implementovaný vyhledávací modul, a nebo využít poznatky z experimentů pro zlepšení stávajícího systému. Z experimentů plyne, že výkon systému by mohl benefitovat z implementace lemmatizace a odstranění nežádoucích znaků při vyhledávání podobných slov.

V této práci se podařilo úspěšně porovnat a analyzovat různé metody CLIR, což přispělo k lepšímu pochopení jejich výhod a nevýhod. Následné experimenty poskytly dostatečný základ pro hodnocení těchto metod, i když by bylo ideální provést více experimentů s různými přístupy a na kvalitnějším datasetu, jehož výběr byl jedním z omezení této práce. Dataset byl zvolen z důvodu splnění kritérií pro testování kombinace češtiny a němčiny, avšak nebyl vytvořen ručně, což mohlo ovlivnit výsledky experimentů. Přesto se i s tímto omezením podařilo porovnat vybrané metody na stejných datech a získat tím relevantní informace.

Pro budoucí výzkum je doporučeno zaměřit se na výběr datasetu na základě jeho kvality a ručně vytvořených dat. V případě potřeby lze použít jinou kombinaci testovaných jazyků nebo vytvořit vlastní dataset.

Vytvořený CLIR modul pro vyhledávání, který je založený na vícejazyčném modelu ColBERT-X, je funkční a splnil zadané cíle, přičemž má prostor pro další vylepšení. V budoucí práci by bylo možné začlenit více zkoumaných metod, z nichž by si uživatel mohl vybírat podle konkrétních potřeb. Práce s modulem by se dala vylepšit také prostřednictvím dalších funkcí a optimalizací.

Celkově tato práce přispěla k pochopení a vývoji CLIR metod a jejich aplikaci v rámci projektu Porta fontium. Přestože byla identifikována některá omezení, poskytuje tato práce cenné informace pro další výzkum a zlepšení vyhledávacího modulu.

Uživatelská příručka A

A.1 Modul pro vyhledávání

Pro spuštění modulu pro vyhledávání je potřeba (ideálně virtuální) prostředí Python na stroji s grafickou kartou s podporou NVIDIA CUDA.

Ke spuštění na operačním systému Windows je nutné mít Python maximálně verze 3.7, jelikož použitá knihovna Pytorch není na Windows podporována pro vyšší verze. U operačního prostředí Linux takové omezení není.

Pro vytvoření virtuálního prostředí, které izoluje nainstalované package, je nutné buď mít instalaci Python, jejíž součástí je modul `venv`, nebo nainstalovanou package `virtualenv` pomocí příkazu

```
$> python -m pip install virtualenv
```

virtuální prostředí lze vytvořit příkazem:

```
$> python -m <venv|virtualenv> .venv
```

a na Linuxu aktivovat:

```
$> source .venv/bin/activate
```

nebo ve Windows:

```
$> .venv/Scripts/Activate
```

Po skončení lze prostředí jednoduše deaktivovat příkazem `deactivate`

Před spouštěním jednotlivých příkazu je nutné zkompilovat a nainstalovat závislosti. K tomu slouží skripty `requirements/compile.py` a `requirements/sync.py`. Po instalaci je možné používat modul za pomocí příkazů popsaných v 4.5

Pro vyzkoušení REST API je nejjednodušší použít API klient jako je Insomnia nebo Postman.

Bibliografie

[AD19] AZAD, Hiteshwar Kumar; DEEPAK, Akshay. Query expansion techniques for information retrieval: A survey. *Information Processing & Management*. 2019, roč. 56, č. 5, s. 1698–1735. ISSN 0306-4573. Dostupné z DOI: https://doi.org/10.1016/j.ipm.2019.05.009.

[BTS18] BRYCHCÍN, Tomáš; TAYLOR, Stephen Eugene; SVOBODA, Lukáš. *Cross-lingual Word Analogies using Linear Transformations between Semantic Spaces*. 2018. Dostupné z arXiv: 1807.04175.

[Jia+20] JIANG, Zhuolin; EL-JAROUDI, Amro; HARTMANN, William; KARAKOS, Damianos; ZHAO, Lingjun. *Cross-lingual Information Retrieval with BERT*. 2020. Dostupné z arXiv: 2004.13005.

[KZ20] KHATTAB, Omar; ZAHARIA, Matei. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. 2020. Dostupné z arXiv: 2004.12832 [cs.IR].

[Law+22] LAWRIE, Dawn; YANG, Eugene; OARD, Douglas; MAYFIELD, James. Multilingual ColBERT-X. 2022. Dostupné z DOI: 10.48550/arXiv.2209.01335.

[LG14] LEVY, Omer; GOLDBERG, Yoav. Dependency-Based Word Embeddings. In: *Annual Meeting of the Association for Computational Linguistics*. 2014.

[SP19] SALEH, Shadi; PECINA, Pavel. An Extended CLEF eHealth Test Collection for Cross-Lingual Information Retrieval in the Medical Domain. In: AZZOPARDI, Leif et al. (ed.). *Advances in Information Retrieval*. Cham: Springer International Publishing, 2019, s. 188–195. ISBN 978-3-030-15719-7.

[SP20] SALEH, Shadi; PECINA, Pavel. Document Translation vs. Query Translation for Cross-Lingual Information Retrieval in the Medical Domain. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Lin-

guistics, 2020, s. 6849–6860. Dostupné z DOI: 10.18653/v1/2020.acl-main.613.

[Sch+14] SCHAMONI, Shigehiko; HIEBER, Felix; SOKOLOV, Artem; RIEZLER, Stefan. Learning Translational and Knowledge-based Similarities from Relevance Rankings for Cross-Language Retrieval. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, 2014, s. 488–494. Dostupné z DOI: 10.3115/v1/P14-2080.

[SD20] SUN, Shuo; DUH, Kevin. CLIRMatrix: A massively large collection of bilingual and multilingual datasets for Cross-Lingual Information Retrieval. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, 2020, s. 4160–4170. Dostupné z DOI: 10.18653/v1/2020.emnlp-main.340.

Seznam obrázků

2.1 Příklad použití UI vyhledávacího systému Porta fontium 9

3.1 Diagram postupu otestování Porta fontium 17
3.2 Vizualizace porovnání předzpracování a následného zpracování metod při vyhledávání pomocí slovních vektorů nad CLIRMatrix. Tvar bodů značí metody předzpracování, barva značí metodu následného zpracování a obrys značí zda bylo vyhledávání case-sensitive. Osa X udává rozřaření do kategorií podle počtu vyhledáváných podobných slov a v rámci jedné kategorie vyjadřuje seřazení podle metriky nDCG@100 pro vyšší přehlednost grafu. Osa Y vyjadřuje metriku nDCG@100. . . 20

Seznam tabulek

2.1 Porovnáné MAP skóre výsledků pro CLIR BERT [Jia+20] nad datasetem z IARPA MATERIAL programu 6
2.2 Příklady generovaných podobných slov pomocí systému Porta fontium 10
2.3 Porovnání CLIR datasetů 12

3.1 Souhrn výsledků vyzkoušených metod v rámci experimentů nad datasetem CLIRMatrix. *ColBERT-X byl porovnán za jiných podmínek, viz 3.4 22

4.1 Seznam parametrů pro REST API koncový bod `/retrieve/<index_name>` 27
4.2 Seznam argumentů pro příkaz `train` 29
4.3 Seznam argumentů pro příkaz `index` 31
4.4 Seznam argumentů pro příkaz `index_faiss` 32
4.5 Seznam argumentů pro příkaz `retrieve` 32