Entity Retrieval through Entity Annotated Corpus

Tino Hakim Lazreg

June 1, 2016

Abstract

Entity retrieval is an important part of information retrieval, and a large number of search queries are entity oriented. This thesis focuses on performing entity retrieval using an entity annotated corpus. For this purpose we will index the ClueWeb12 corpora, and use the FACC1 to create an index of documents that are linked to entities. Related research were reviewed, and existing test collections were used in the experimentation. We adapt an expert retrieval model to entity retrieval, and we provide results of the model for three different test collections. Model 2 achieved a varying score depending on the test collection. A detailed discussion and analysis of the datasets, and the model used are presented in the thesis. The best score was achieved for the REWQ ClueWeb dataset, with Model 2 achieving MAP score of 0.2637, P@10 score of 0.3676, NDCG score of 0.4674 and NDCG@10 score of 0.4738.

Sammendrag

Entitetsgjenfinning er en viktig del av informasjonsgjenfinning, og et stort antall søk på søkemotorer er orientert rundt entiteter. Denne oppgaven undersøker mulighetene for entitetsgjenfinning på entitet annotert korpus. For å oppnå dette vil vi indeksere ClueWeb12 korpus, og bruke FACC1 til å lage en indeks av dokumentene som er lenket til entiteter. Relatert forskning ble vurdert, og eksisterende testsamlinger ble brukt i eksperimentene. I denne oppgaven, vil vi tilpasse en ekspertgjenfinningsmodel, Model 2, og evaluere modellen for tre forskjellige testsamlinger. Model 2 oppnådde en variende score avhengig av testsamlingen. En detaljert diskusjon og analyse ble utført av datasettene, og modellen som ble brukt. Den beste scoren ble oppnådd for REWQ ClueWeb datasettet, hvor Model 2 fikk en MAP score på 0.2637 og en P@10 score på 0.3676, NDCG score på 0.4674 og NDCG@10 score på 0.4738.

Preface

This is a Master thesis that has been carried out at the Department of Computer and Information Science, Faculty of Information Technology, Mathematics, and Electrical Engineering at Norwegian University of Science and Technology (NTNU). The work has been supervised by Professor Svein Erik Bratsberg, and co-supervised by PhD Candidate Faegheh Hasibi.

I would like to express my gratitude towards my supervisors, Svein Erik and Faegheh, for their continuous advice and assistance while writing the thesis.

I would also like to thank my brother, Sofien Lazreg, for advice regarding the implementation.

Finally, I would like to thank my family and friends for their endless support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet has quickly become the primary source for information regarding most topics. According to Bounie and Gille [7] the world generated 14.7 exabytes of new information in 2008. The enormous amounts of data being produced is growing for each year. Information Retrieval (IR) is a broad term that can be defined as the activity of obtaining information from an information source. In the context of computer science, the common approach is a user that states an information need, and the IR system provides the user with the requested information. Manning et al. [25] defined information retrieval as:

> "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)."

Information retrieval is used by hundreds of millions of people every day. People use Web search engines, that employ information retrieval, several times a day. The classic approach to information retrieval in Web search engines is that the user provides a query, and the system returns a ranked list of documents. Figure 1.1 shows a simplified approach to a Web search engine.

One difficult task with the Web is the amount of new information generated each year. It is estimated that 90 % of the data in the world has been created in the past two years[1]. This makes it increasingly difficult for

[1] http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

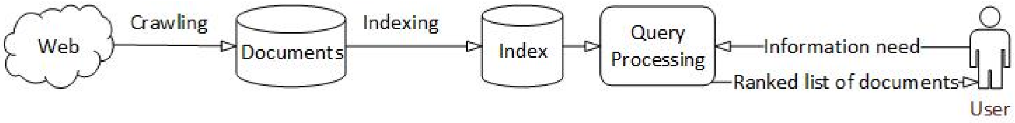
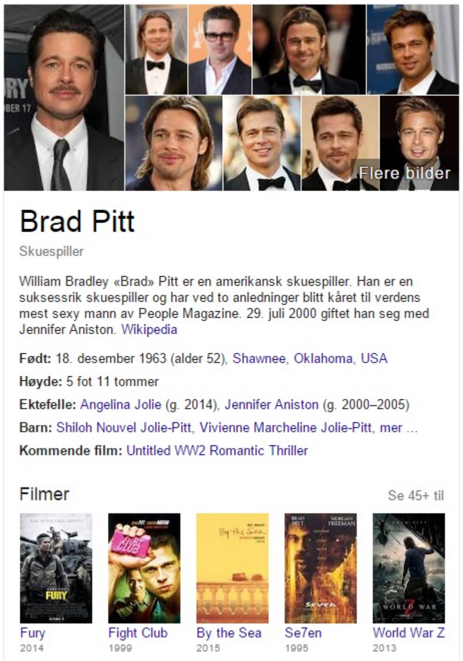


Figure 1.1: Simplified flow in a Web search engine

the user to filter out irrelevant or unwanted data. Information need is also dependent on the specific user, e.g. the same query from two different users can give different satisfaction. Semantic search attempts to tackle this task by understanding user intent and the context of the query. The goal is to find the actual answers that meet the user's information need, by combining text and structure.

Entities are unique objects or things. Examples of entities can be a person, a location, a service, etc. Figure 1.2 shows an entity that is returned when entering the query string "Brad Pitt" in Google [2].



Brad Pitt

Skuespiller

William Bradley «Brad» Pitt er en amerikansk skuespiller. Han er en suksessrik skuespiller og har ved to anledninger blitt kåret til verdens mest sexy mann av People Magazine. 29. juli 2000 giftet han seg med Jennifer Aniston. Wikipedia

Født: 18. desember 1963 (alder 52), Shawnee, Oklahoma, USA

Høyde: 5 fot 11 tommer

Ektefelle: Angelina Jolie (g. 2014), Jennifer Aniston (g. 2000–2005)

Barn: Shiloh Nouvel Jolie-Pitt, Vivienne Marcheline Jolie-Pitt, mer ...

Kommende film: Untitled WW2 Romantic Thriller

Filmer

Se 45+ til

Fury
2014

Fight Club
1999

By the Sea
2015

Se7en
1995

World War Z
2013

Figure 1.2: Example of an entity

[2] https://www.google.com/?q=Brad+Pitt

Search for entities makes up a large portion of all Web search queries. Lin et al. [24] collected a frequency-weighted query sample of 200 queries, and examined each query to determine if it contains an entity. Queries were divided into six groups:

- Contains an entity
- Contains an entity and refiner words
- Contains an entity category (e.g., "car battery")
- Contains an entity category and refiner words
- Contains a website entity (URL or website name)
- All other queries

To illustrate what constitutes a refiner word, we can look at the example query "download PDF". In this query, "download" is the refiner word and "PDF" is the entity.

Figure 1.3 shows the distribution of the Web search queries. 43 % of the queries contain an entity, 14 % contain an entity category, 28 % contain a website name or url, and 15 % do not contain any entity. An important assessment that can be made about the distribution, is that users know what they are looking for, but not where to find the information. This is where entity retrieval plays an important role in semantic search.

Entities play a central role in information retrieval, and have seen an increase in attention from the research community. Linked Data is essential for semantic search and entity retrieval. Knowledge bases as DBpedia[3] and Freebase[4] are used to link documents or queries to entities.

The important role of entities in IR is a huge motivation for continued research into entity retrieval. In this thesis we will take a further look at entity retrieval. Information retrieval is an important asset that people use every day, and there are clearly optimizations that can be done to achieve better results to queries. Entity retrieval is also gaining traction in the public domain. Services like Siri, Google Now and Wolfram Alpha are examples of commercial usages of entity linking and entity retrieval.

[3] http://wiki.dbpedia.org/

[4] https://www.freebase.com/

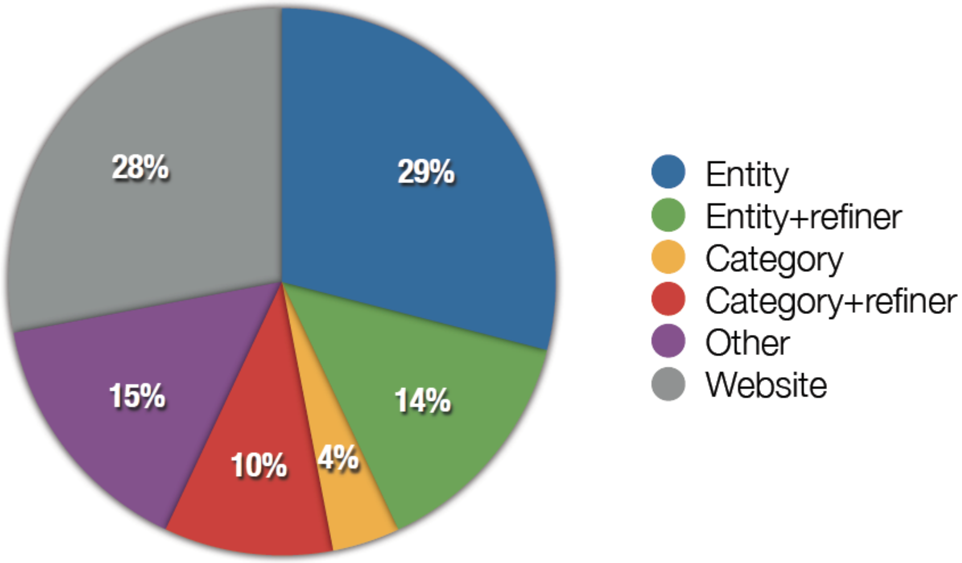


Figure 1.3: Distribution of Web search queries [24]

1.1 Context

This thesis is part of Faegheh Hasibi and Krisztian Balog's current work in the field of entity-oriented search, which includes entity retrieval and entity linking. Table 1.1 shows the context for the thesis. Hasibi and Balog have already implemented generative models using DBpedia as the data source. We will implement generative models using an annotated corpus as the data source.

Data Source	**Model**	
	Generative	Discriminative
KB DBpedia	[5], [42]	-
Annotated Corpus	this thesis	[36]

Table 1.1: Overview of context

1.2 Project Goals

The problem that we want to address in this thesis is how to perform entity retrieval on annotated corpus using generative models. To achieve this we want to develop a system that is able to preprocess the ClueWeb corpus [34], and replace entity mentions in ClueWeb with the corresponding annotations from FACC [15]. The system will then index the preprocessed data. We will also implement Model 2 from the "Formal Models for Expert Finding in Enterprise Corpora" paper [5], and evaluate the results.

The problem can be divided into four main tasks.

1. Preprocess ClueWeb data
2. Index the preprocessed data
3. Implement Model 2 from the "Formal Models for Expert Finding in Enterprise Corpora" paper
4. Evaluate Model 2, and compare it to some baselines

The input to the first task will be both the ClueWeb dataset and the FACC dataset. After the index is created, and the models are implemented, we will use the queries from the DBpedia entity test collection [4], REWQ Robust04 and REWQ ClueWeb12 [36] test collections as the input. The system will perform entity retrieval using Model 2 on the queries, and the output will be a ranked list of entities.

1.3 Thesis Outline

The thesis is organized as follows:

- Chapter 1: Introduction to the thesis and project goals.
- Chapter 2: Introduces the background theory and literature review of the thesis.
- Chapter 3: Describes in detail our approach to the problem.
- Chapter 4: Presents details on the implementation of the system.

- Chapter 5: Presents the results, evaluation and a discussion of the results obtained.
- Chapter 6: The final conclusion of the thesis, and recommendations for future work.

Chapter 2

Background

This chapter will provide the background information needed to understand the thesis. The following topics will be introduced and explained: information retrieval, entity linking, entity retrieval and language models.

2.1 Retrieval Model

In information retrieval, there are a variety of different models that can be used to accomplish your task. The models can be divided into three different categories: set-theoretic models, algebraic models and probabilistic models [1]. Set-theoretic models represents documents as a set of words or phrases. Boolean model is a well-known set-theoretic model. Algebraic models represents documents and queries as vectors, matrices or tuples. An example of an algebraic model is the vector space models, that represents documents and queries as vectors. The last category is probabilistic models, where a user's information needs are translated into query representations, and documents are converted into document representations. Probabilistic models try to estimate the probability of a document being relevant to an information need. Language models are a type of probabilistic models [25].

2.1.1 Language Model

Language models originally came from probabilistic models developed for automatic speech recognition systems, where the goal was to predict the next term based on the terms already spoken [35]. The approach of applying language models to ad-hoc document retrieval was proposed by Ponte and Croft [32] in 1998. Since then, there have been proposed many variations to the LM approach.

A language model is a probability distribution of a sequence of words. Given a sequence, the language model will assign a probability to the whole sequence. When used in information retrieval, each document d is used to estimate a language model. The model is then used to calculate the probability of any word sequence. This is used to rank documents according to their probability of generating the query q: $P(d|q)$. There are three common steps for retrieval based on LM:

1. Infer a LM for each document
2. Estimate the probability of generating the query according to each of these language models
3. Rank the documents according to the probabilities

An important issue with language models is smoothing, where you adjust the probability of terms to compensate for data sparseness. Smoothing in models is used to avoid zero probabilities, and as a part of the term weight component. There are a wide number of approaches to smoothing probability distributions, however we will focus on two approaches, Jelinek-Mercer and Dirichlet [41].

Standard language model approach:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \propto P(q|d)P(d) \tag{2.1}$$

$P(d)$ is the probability of the document being relevant to any query.
$P(q|d)$ is the query likelihood, i.e. the probability that a query q is produced by document d.

$$P(q|d) = \prod_{t \in q} P(t|\theta_d)^{n(t,q)} \tag{2.2}$$

$$P(t|\theta_d) = (1 - \lambda) \cdot P(t|d) + \lambda P(t|C) \tag{2.3}$$

$n(t, q)$ is the number of times term t appears in query q.
$P(t|d)$ estimates a multinominal probability distribution from the text.

$P(t|C)$ smooths the distribution by using probability from the entire collection.
λ is the smoothing parameter.

Figure 2.1 depicts how the document language model is calculated. The first square is a document, and the colored circles are different terms. A multinomial probability is estimated based on the terms in the document, and is then smoothed by using probability from the entire collection. This gives us the final box with the probability distribution of the term.

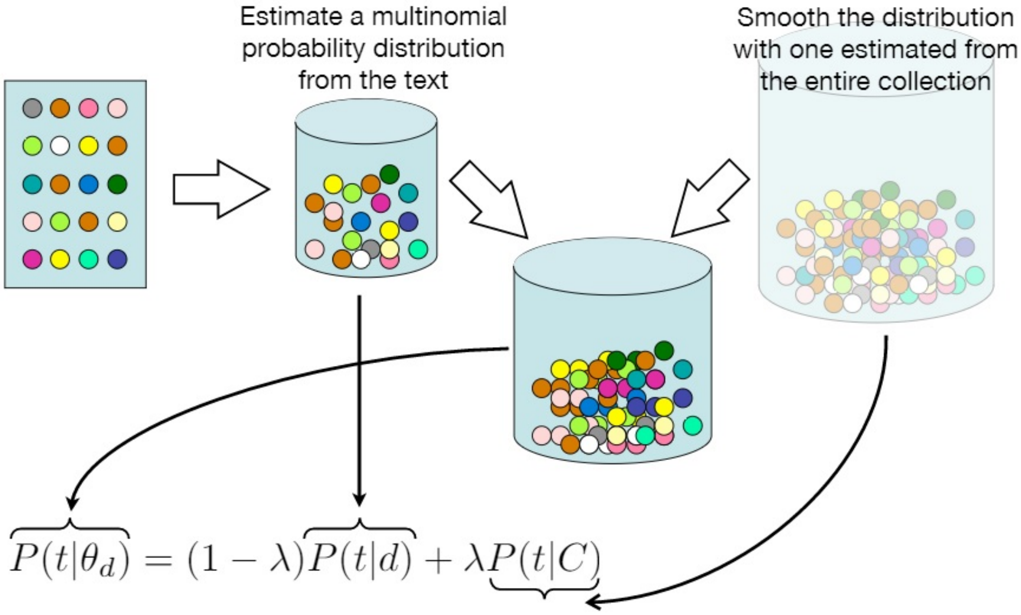

Estimate a multinomial probability distribution from the text
Smooth the distribution with one estimated from the entire collection
$\overbrace{P(t|\theta_d)} = (1-\lambda)\overbrace{P(t|d)} + \lambda\underbrace{P(t|C)}$

Figure 2.1: Document language model

Ogilvie and Callan [31] present a paper that investigates combining document representations for known-item search. Known-item search is an important task where the user knows about a particular document, but not the location of the document. The paper experimented with two different approaches, a mixture-based language model and meta-search algorithms. Equation 2.4 shows the mixture-based language model that is proposed in the paper.

$$P(w|\theta_D) = \sum_{i=1}^{k} \lambda_i P(w|\theta_{D(i)}) \tag{2.4}$$

k is the number of language models, D(i) is the document's i^{th} presentation, and λ_i is the weight of the model $\theta_{D(i)}$. The probability distribution of each model $\theta_{D(i)}$ is estimated by taking a linear interpolation of the maximum likelihood estimate of the text observed in the i^{th} document representation, and a collection language model is estimated from all the document representations of the same type in the collection.

The authors came to the conclusion that the best meta-search algorithms did not perform as well as the mixture-based language model. We will implement language models and this paper supports the notion that language models are a good approach to entity retrieval.

2.2 Entity Linking

Entity linking is the process of recognizing entity mentions in a text, and linking them to a reference in a knowledge base [21]. The entities are usually gathered from a knowledge base, e.g. Wikipedia or Freebase. Entity linking provides the text with a context, which makes it possible to understand the text. We differentiate between entity linking where the text are documents or queries. Figure 2.2 shows an example text with entities being linked to a knowledge base. In this example we have used Freebase as the knowledge base, and we can see that "Brad Pitt", "Actor", and "Golden Globe Award" has been linked to entities in Freebase.

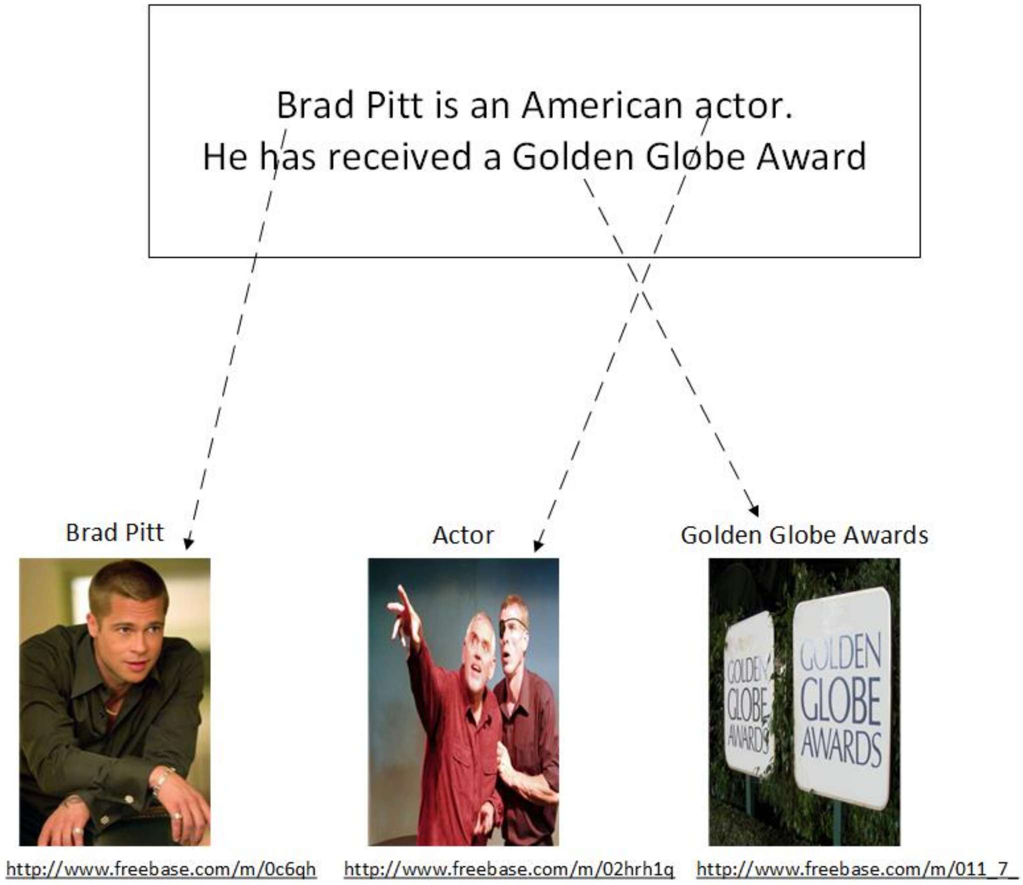

Brad Pitt is an American actor.
He has received a Golden Globe Award
Brad Pitt
Actor
Golden Globe Awards
http://www.freebase.com/m/0c6qh
http://www.freebase.com/m/02hrh1q
http://www.freebase.com/m/011_7_

Figure 2.2: Example of entity linking

2.2.1 Entity Linking in Documents

The Wikify! System [28] is one of the earliest approaches of entity linking in documents. The system identifies the important concepts in an input document, and automatically links the concepts found to the corresponding Wikipedia articles. The concept detection is done by extracting all n-grams that match Wikipedia concepts and filtering them. The entity disambiguation is accomplished by a combination of knowledge-based and feature-based learning approaches. The evaluation of the system showed that automatic annotations are reliable, and almost indistinguishable from manual annotations.

Milne and Witten [29] proposes a machine learning approach to entity linking in documents, with Wikipedia as the knowledge base. The link detector and disambiguator used in the system, provides recall and precision of almost 75 %. Wikipedia was used as a training set. The approach was tested on new documents from a different text corpus, and the results indicated that the algorithm works on other text corpora than Wikipedia.

The research in entity linking in documents proves that automatic annotations are reliable. It is also evident that it is possible to perform entity linking on documents from other sources than Wikipedia.

2.2.2 Entity Linking in Queries

Hasibi et al. [21] presents a paper where they discuss entity linking in queries. Two major tasks are discussed, which are semantic mapping and interpretation finding. There is an important distinction between the tasks. Semantic mapping is used to give users suggestions that could benefit navigation or contextualization. Interpretation finding is used to find semantically related entities mentioned in a query, and return several sets if the query has multiple interpretations. An interpretation is defined as a set of entities. This paper provided us with domain understanding of entity linking, and the difference between semantic mapping and interpretation finding.

Meij et al. [26] proposed an approach to adding semantics to microblog posts, and automatically link the tweets to corresponding Wikipedia articles. The paper proposes a novel method based on machine learning, and also shows that recently proposed approaches for semantic linking does not perform well on microblog posts.

While this research provides us with knowledge on entity linking, our research will be focused on entity retrieval and use a different knowledge base than Wikipedia. It was still useful to gather valuable information on entity linking, and the issues with heterogeneous datasets.

The Entity Recognition and Disambiguation (ERD) challenge [8] main goal was to promote research in recognition and disambiguation of entities in unstructured text. The task is that given an input text and a knowledge base, the system should recognize entity mentions in the input text, and link them to entities in the knowledge base. The challenge included two tracks, one long-text track for documents, and one short-text track for Web search queries. The challenge requested the output from the short-text track to be a set of valid entity linking interpretations of the query. The ERD challenge garnered a lot of attention, and 27 teams submitted final results. We will present some of the approaches to the challenge.

Hasibi et al. [20] proposed an approach that uses a three-stage pipeline, where the system first detect candidate entities, score the entities, and finally finds the interpretation sets for the input query. The focus for the first stage, candidate detection, is to obtain high recall. This is accomplished by generating all n-grams of the query, and performing lexical matching between n-grams and entity names. The second stage, entity scoring, was accomplished by scoring the identified candidate entities using a learning-to-rank approach. Two datasets were used as the ground truth, the query annotations from the ERD challenge [8] and the Yahoo! webscope dataset [1]. After getting a score for each candidate entity, the last stage involves finding all the valid interpretations of the query. The paper proposed a greedy algorithm for the interpretation finding, which takes a list of ranked entities as the input, and outputs zero or more interpretation sets associated with the query.

Chiu et al. [10] presented a system, NTUNLP, for both tracks in the challenge. The NTUNLP team was the winner of the first prize in the ERD challenge 2014 for the short track. This was accomplished by first creating an alias dictionary with the surface forms of Freebase IDs. The query text was then scanned from left to right with string matching between the dictionary and the substrings of the query. Inappropriate surface forms were eliminated using a stop word list based on the commonness of the words. Finally, the system was integrated with TAGME Ferragina and Scaiella [16] and Wikipedia article titles were used to improve the system performance. NTUNLP did not handle multiple interpretations, but only generated the top interpretation.

Cornolti et al. [11] presented a paper where they propose the SMAPH system, which uses a pipeline of four main steps. First, search results are fetched from a search engine given the query to be annotated. Then search result snippets are parsed by looking at the bold parts of the search snippets to identify candidate mentions. Following is the candidate generation step, were candidate entities are generated from the Wikipedia pages occurring in the search results, and from an existing annotator. The last step is pruning, were the SMAPH system takes into use a binary SVM classifier to decide which entities to keep or discard.

[1] http://webscope.sandbox.yahoo.com/

The TAGME system [16] is used in both the NTUNLP and SMAPH systems. The TAGME system takes plain-text and annotates it with hyperlinks to Wikipedia pages. TAGME is able to annotate short-texts such as search-engine results, tweets, etc. Hasibi et al. [22] tested the reproducibility of the TAGME system, and showed that TAGME could in fact be extended to the task of entity linking in queries.

2.3 Entity Retrieval

Entity retrieval is a way to address information needs that are better answered by returning specific objects, also called entities, instead of just any type of document. Entity search is the most popular type of Web search [33], and has attracted interest in the research community.

Figure 2.3 shows an example of entity retrieval used in a commercial search engine. The query input in Google is “museums in amsterdam”. The entities in this query is museum and amsterdam. The search engine interprets the query, and understands based on the entities that the information need is museums located in Amsterdam. As can be seen in Figure 2.3, a list of museums located in Amsterdam is returned as the top result.

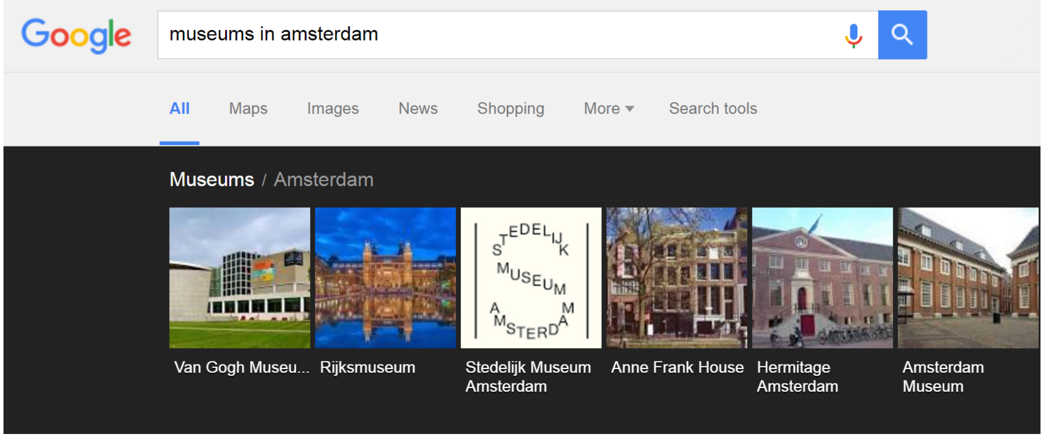

museums in amsterdam

Figure 2.3: Example of entity retrieval

Zhiltsov et al. [42] proposes a novel retrieval model that incorporates term dependencies into structured document retrieval, and use it in the task of Entity Retrieval in the Web of Data (ERWD). The proposed model, FSDM, replaces a single document language model, with a mixture of language models for each document field. The model is an extension of the dependence model, with respect to multi-fielded entity descriptions.

Balog et al. [5] paper on "Formal Models for Expert Finding in Enterprise Corpora" introduces two general models for expert finding. Expert finding is similar to entity retrieval. Experts are another type of entities, with the same qualities. The two different approaches were compared in the evaluation. The paper found that Model 2 performs better than Model 1 in almost all measures. The measures used in the paper is Mean average precision, R-precision, mean reciprocal rank, precision@10 and precision@20.

This paper provided us with Model 2, which proved to have excellent performance compared to other unsupervised techniques. Model 2 will be implemented and used in this research paper. Balog et al. [5] used TREC Enterprise corpora to evaluate the models. However we want to evaluate the model on the ClueWeb12 dataset with the corresponding Freebase annotations.

There has been an effort to create test collections for entity retrieval. Vries et al. [39] created a test collection for entity retrieval in Wikipedia. The goal is to evaluate how well systems are able to rank entities based on a query. The track consists of triples of type <category, query, entity>. "category" specifies the type of entities, "query" contains the information need, and "entity" contains a list of example entities. A new version of the INEX-XER track was made available in 2009, using the new Wikipedia 2009 XML data, based on a Wikipedia dump taken in 2008 [14].

Balog et al. [2] presented the TREC 2010 entity track, which was another effort to create a test collection for entity retrieval. However, the TREC entity track uses a different corpora than the INEX-XER track. ClueWeb is used as the knowledge base. Entities are represented by their homepages, and the homepage URL is used as the unique identifier. Wikipedia pages were not accepted as entity homepages, which also makes the test collection different from the INEX-XER track. The main task was that given an input in free text consisting of an entity, the type of the target entity or location, and the relation. Then the output should be homepages of related entities, and optionally the name of the entity.

Balog et al. [3] presented a few changes to the TREC 2010 entity track. The TREC 2011 entity track's main change is the usage of the Sindice-2011 corpus, a semantic Web crawl, instead of the BTC-2009 collection that was used in the 2010 entity track. The reasons given for this change is that the new corpus is a larger and more representative linked open data (LOD) crawl, and that examples are not mapped manually to LOD, but rather given as ClueWeb document identifiers. A new pilot task was introduced, REF-LOD, which wanted to explore the differences between ranking Web data and semantic Web data. REF-LOD is very similar to the main task in the TREC 2010 entity track, with the difference being that results should be identified by their LOD URIs instead of their homepages.

Balog and Neumayer [4] presented another test collection for entity search based on the DBpedia knowledge base. The test collection is explored in more detail in Chapter 2.4.3.

2.4 Data

We will be using five datasets for this thesis. The first is the ClueWeb dataset that contains English Web pages. The second dataset is the Freebase Annotations of the ClueWeb Corpora (FACC), which contains annotations of the ClueWeb dataset. The three remaining datasets are test collections, namely the DBpedia entity test collection, REWQ Robust04 and REWQ ClueWeb12.

2.4.1 ClueWeb

The ClueWeb12 dataset [34] consists of 733,019,372 English Web pages. The pages have been collected between February 10, 2012 and May 10, 2012. Most of the documents were collected by using the Internet's Archive Heritrix Web crawler [2]. The crawlers were configured to capture page text, css, html, javascript, images, and http response header. Multimedia files, i.e. audio and video files, were left out of the crawling. The documents were then processed by a series of requirements. The documents that did not pass the requirements were removed. The requirements included removing non-English documents, documents from domains that appear to contain adult content, and documents larger than 10 megabytes. The dataset were then divided into segments of approximately 50 million documents, and each segment was divided into warc files that are approximately 1 GB in size uncompressed. WARC is an archive format specified to contain different

[2] https://webarchive.jira.com/wiki/display/Heritrix

digital resources in the same archive [3]. Each WARC file consists of a series of records, that have a WARC-TREC-ID as the document identifier.

Size compressed	5.54 TB
Size uncompressed	27.3 TB
Number of WARC files	33,447
Number of documents	733,019,372

Table 2.1: Statistics for ClueWeb12

We will be using the ClueWeb12 B13 dataset, which is a 7 % uniform sample of the full dataset. ClueWeb12 B13 was created by taking every 14th document from each of the WARC files in the complete dataset. Table 2.4.1 and Table 2.4.1 show the statistics for the datasets.

Size compressed	389 GB
Size uncompressed	1.95 TB
Number of WARC files	33,447
Number of documents	52,343,021

Table 2.2: Statistics for ClueWeb12 B13

Preprocessing ClueWeb

The ARSC team [30] indexed the Category "B" subset of the ClueWeb09 collection , using a divide and conquer method, across the separate ClueWeb subsets for 1, 2 and 3-grams . The ClueWeb data was preprocessed to remove HTML-tags, since Lucene only processes plain text documents. Markup languages must be removed before indexing. The preprocessing of the ClueWeb data took approximately 1705 CPU hours to compute. The paper states that the team utilized several systems to accomplish the work. The processing of the ClueWeb dataset and Lucene indexing, was computed using ARSC's "Midnight". "Midnight" is a supercomputer cluster with 2312 processors and a peak performance of 12.8 teraflops. A teraflop is a measure of a computer's speed, and can be expressed as a trillion floating point operations per second. Midnight consists of 358 "small" nodes, with two dual-core Opteron processors, and 55 "large" nodes with eight dual-core Opteron processors. Nodes provide 4 GB of memory per processor core.

[3] http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml

Multiple jobs, each with multiple Lucene threads, were needed to complete the indexing. The paper used index merging, which takes multiple indexes and merges them into one index. The indexer was multithreaded, and created a set of indexes in parallel during the program's run. The indexes were then merged into a single final index.

The research paper provides us with knowledge about ClueWeb preprocessing. It is time-consuming, and computationally expensive to preprocess the dataset. We will be working on a newer version of the ClueWeb collection, the ClueWeb12 collection. The dataset is larger, which has to be taken into consideration when comparing the findings of the ARSC team. The research also made us aware of the importance of parallelizing the preprocessing and indexing.

2.4.2 FACC1

The FACC1, Freebase Annotations of the ClueWeb Corpora v1, [15] consists of annotations of the ClueWeb12 data. Researchers at Google were responsible for annotating the ClueWeb12 dataset, and made the annotations publicly available. The annotations are stored with the extension tsv, tab separated values.

Table 2.3 shows an excerpt from a FACC file. The first column is the WARC-TREC-ID, which is unique for each annotation. The second column is the original encoding, that was used to find the annotation. The third column is the entity mention that has been detected in the ClueWeb data. The fourth and fifth columns are the beginning and end byte offsets for the entity mention. This is where the entity mention is located in the given ClueWeb record. The beginning byte offset's zero location is at the beginning of the HTTP header. The sixth and seventh columns are two confidence levels. However, they are calculated differently. The first confidence level is the posterior probability of an entity given both the mention and the context of the mention. The second confidence level is the posterior probability given just the context of the mention, excluding the mention string. Finally the last column is the Freebase identifier for the entity mention. The Freebase identifier can be used to look up the entity mention in Freebase, just prepend each ID with "http://www.freebase.com".

WARC ID	Encoding	Entity	Begin	End	Confidence	Confidence	Freebase ID
clueweb12-0000tw-00-00000	UTF-8	Flash Player	12770	12782	1.000000	0.024096	/m/05qh6g
clueweb12-0000tw-00-00001	UTF-8	Flash Player	12815	12827	1.000000	0.024096	/m/05qh6g
clueweb12-0000tw-00-00002	UTF-8	Flash Player	12848	12860	1.000000	0.024096	/m/05qh6g
clueweb12-0000tw-00-00003	UTF-8	Flash Player	12912	12924	1.000000	0.024096	/m/05qh6g

Table 2.3: Excerpt from FACC file 0000tw.anns.tsv

2.4.3 DBpedia Entity Test Collection

Balog and Neumayer [4] presents a paper that develops an entity search test collection based on the DBpedia knowledge base. The collection is made publicly available, and consists of 485 queries, and corresponding relevance judgements. The paper presented baseline results and showed limitations of methods based on language models and BM25.

This research paper provides us with an entity test collection based on DBpedia. The test collection will be used in the experiments and evaluation of the models implemented in this paper. However, the test collection uses DBpedia as a knowledge base, while we will be using Freebase. This will require some modifications to be made to the test collection before taking it into use.

The test collection consists of two text files, queries.txt and qrels.txt. The first text file contains the 485 queries, which have been collected from different sources.

- 100 queries from the INEX 2012 Linked Data track

- 55 queries from the INEX 2009 Entity Ranking track
- 140 queries from the Question Answering over Linked Data Challenge
- 130 queries from the entity search task of the 2010 and 2011 Semantic Search Challenge
- 43 queries from the list search task of the 2011 Semantic Search Challenge
- 17 queries from the TREC 2009 Entity track

The qrels text file contains the relevance judgements for the queries. Table 2.4 shows an excerpt from the queries text file,and table 2.5 shows an excerpt of relevance judgements for the queries.

INEX_LD-20120111	vietnam war movie
INEX_LD-20120112	vietnam war facts

Table 2.4: Excerpt from queries.txt

INEX_LD-20120111	http://dbpedia.org/resource/Nam%27s_Angels
INEX_LD-20120111	http://dbpedia.org/resource/We_Were_Soldiers
INEX_LD-20120112	http://dbpedia.org/resource/Donald_Cook_%28Medal_of_Honor%29
INEX_LD-20120112	http://dbpedia.org/resource/Badger_Army_Ammunition_Plant

Table 2.5: Excerpt from qrels.txt

2.4.4 Ranking Entities for Web Queries (REWQ) dataset

Schuhmacher et al. [36] present a paper that proposes two datasets for evaluating entity ranking for Web queries. The paper studies queries from the document retrieval benchmarks: TREC Robust04 dataset, and TREC Web 13/14 dataset.

The relevance judgements for the datasets are created by using a document retrieval system to collect the top results. For the Robust04 dataset, an entity-aware document retrieval method, EQFE [13], was used to retrieve the top results for a query. Entity links are used to produce the document ranking, and they are created with KBBridge [12]. The queries are gathered from the TREC Robust Track '04. The top 25 performing queries of the EQFE system on the dataset are chosen and the top 19 documents for each query are collected. For each query the 50 entities with the highest mention frequency were picked. The final dataset consists of 1250 relevance judgements.

The second dataset created to benchmark entity ranking is based on the TREC Web 13/14 dataset, with the ClueWeb12 corpus. The dataset uses FACC1 for the entity annotations. FACC1 is described in Chapter 2.4.2. 22 queries were randomly picked from the TREC Web 2013/2014 dataset. The Sequential Dependency Model (SDM, [27]) is used to retrieve the top 20 documents. The final dataset consists of all entities per query. However, entities that occurs less than three times are filtered out.

A learning-to-rank approach is used where different features are studied that use documents, entity mentions, and knowledge base entities. Two learning-to rank methods were used in the paper, namely the Ranking Support Vector Machine (SVM, [23]), and an algorithm to optimize an underlying retrieval metric directly, which has been implemented in the RankLib package [4].

[4] http://people.cs.umass.edu/ vdang/ranklib.html

Chapter 3

Entity Retrieval Approach

This chapter contains the models that will be implemented, compared and evaluated. Language model will be used as the baseline method, which the other implementations will use for comparison of the evaluation measures.

3.1 Indexing

The first step in our approach is to create the index. We want to create an index for the ClueWeb B13 collection. Before the index can be created, we preprocess the data by removing HTML and markup tags from each document. The index will be created with three fields of different types. Table 3.1 shows the three field names and corresponding types.

Field name	**Field type**
ID	ID
contents	TEXT_TVP
contents_annotated	TEXT_NTVP

Table 3.1: Field names and field types for the index

The different field types contain different properties. ID is indexed, stored and tokenized. TEXT_TVP is stored, indexed and tokenized with term vectors and positions. TEXT_NTVP is not stored, but it is indexed and tokenized with term vectors and positions. The "ID" field will contain the unique document identifiers, which is the WARC-TREC-ID for each record in the WARC files in the ClueWeb B13 collection. The WARC ID is also used in the FACC collection to denote the relationship between a warc record and an annotation. The "contents" field is used for the content of the documents. The content will first be preprocessed by stripping away HTML and markup tags, and stop words will be removed. The last field, "contents annotated", and the only field that is not stored, will also contain the contents of the documents. However, in the "contents annotated" field, the annotations in the documents are replaced with the FACC identifier.

Instead of running a single indexing procedure for the whole collection, we will index each subfolder of the ClueWeb collection separately. The ClueWeb collection consists of 20 sub-folders. The reason for indexing each subfolder is that it makes it easier to parallelize the indexing. We can initiate several runs simultaneously. After the subfolders are indexed, we will merge the indexes to create one final index with the whole Clueweb B13 collection indexed.

In Figure 3.1 we can see an illustration of the indexing procedure for the whole collection. Multiple instances of the system is run for different subfolders of ClueWeb. After the indexes are created, they are given as input to the merge_indexer who produces the final merged index.

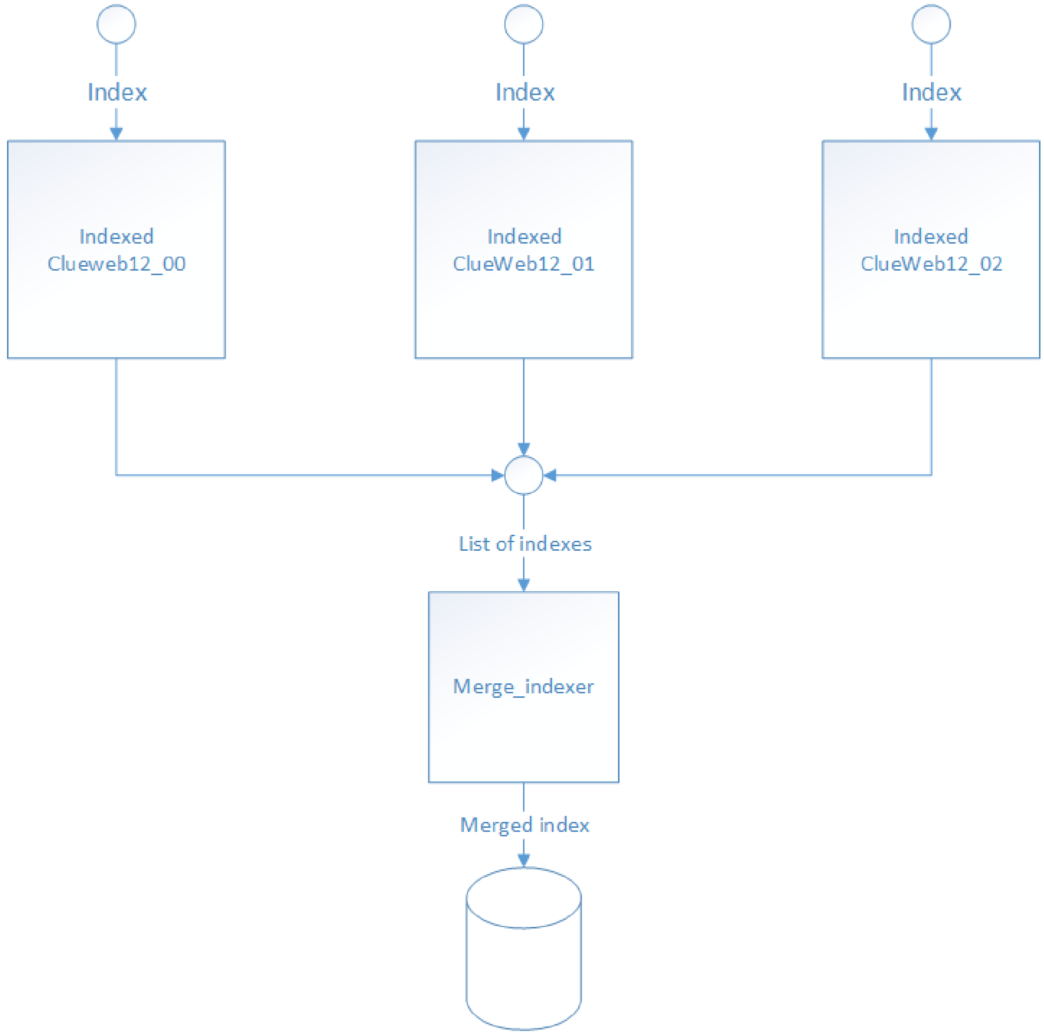


Figure 3.1: Illustration of the indexing procedure

3.2 Retrieval Models

In this section, we will introduce Model 1 and Model 2 from Balog et al. [5], and explain why Model 2 was chosen to be implemented. Model 1 and Model 2 are both used for expert finding in enterprise corpora in [5]. Expert finding is similar to entity retrieval, since experts are just another type of entities. We will implement Model 2, and compare it to the baselines using evaluation measures.

For Model 1 and Model 2 we need to estimate the probability that a document d is associated with entity e. The associations a(d, e) have already been made available for each document and entity. They are found in the FACC dataset, which we will use when implementing the system. We distinguish between two different ways of estimating the probabilities of the associations. The document-centric perspective is to estimate the probability $p(d|e)$. This is expressed as:

$$p(d|e) = \frac{a(d,e)}{\sum_{d' \in D} a(d',e)} \tag{3.1}$$

D is the set of documents, in our case this are the documents from the ClueWeb dataset that the FACC has annotated. If we use $p(d|e)$ to rank documents, the top documents will be the ones that the entity e are strongest associated with.

The second way to estimate the association between documents d and entities e, is the candidate-centric perspective. The probability we want to estimate here is $p(e|d)$ and is expressed as:

$$p(e|d) = \frac{a(d,e)}{\sum_{e' \in E} a(d,e')} \tag{3.2}$$

E is the set of possible entities. The candidate-centric perspective is different from the document-centric perspective, since here we will find the entity that made the biggest contribution to the document d.

3.2.1 Model 1

An entity in this model is represented by a multinomial probability distribution over the terms, $P(t|e)$. Model 1 uses smoothing, since $P(t|e)$ may contain zero probabilities. An entity model θ_e is inferred for each entity. The probability of a term given the entity model is $P(t|\theta_e)$. $n(t,q)$ is the number of times t occurs in q. If the number is greater than zero, that means that the query is represented by a set of terms. θ_e is the entity model. The query likelihood is calculated by taking the product across all the terms in the query:

$$P(q|\theta_e) = \prod_{t \in q} P(t|\theta_e)^{n(t,q)} \tag{3.3}$$

$P(t|\theta_e)$ is constructed by taking a linear interpolation of the background model $P(t)$, and the smoothed estimate:

$$P(t|\theta_e) = (1 - \lambda) \cdot P(t|e) + \lambda P(t) \tag{3.4}$$

$P(t|e)$ is calculated by taking the product of the maximum likelihood estimate of the term in a document, and the document-entity association. This is calculated for all the documents, which gives an estimate of $P(t|e)$. Using a document-centric perspective, this can be expressed as:

$$P(t|e) = \sum_d P(t|d)P(d|e) \tag{3.5}$$

or by using a candidate-centric perspective:

$$P(t|e) = \sum_d P(t|d)P(e|d) \tag{3.6}$$

The final Model 1 is then:

$$P(q|\theta_e) = \prod_{t \in q} [(1 - \lambda)(\sum_d P(t|d)P(d|e)) + \lambda P(t)]^{n(t,q)} \tag{3.7}$$

Model 1 uses all the term information from all the documents associated with the entity, and uses this to construct the entity's language model. The entity model is then used to score each query, which is how likely the entity would produce a certain query.

3.2.2 Model 2

The probability of $P(q|e)$ is computed by assuming conditional independence between the query and the entity.

By taking the sum of all documents, we obtain $P(q|e)$. There are two ways to express this, the document-centric way and the candidate-centric way.

Document-centric:

$$P(q|e) = \sum_d P(q|d)P(d|e) \tag{3.8}$$

Candidate-centric:

$$P(q|e) = \sum_d P(q|d)P(e|d) \tag{3.9}$$

$P(d|e)$ is the probability of a document given an entity, and $P(e|d)$ is the probability of an entity given a document.

$P(q|d)$ is the query likelihood, which is the probability of a query given the document. This is estimated by:

$$\prod_{t \in q} P(t|\theta_d)^{n(t,q)} \tag{3.10}$$

The final Model 2 is:

$$P(q|e) = \sum_d [[\prod_{t \in q} ((1-\lambda)P(t|d) + \lambda P(t))]^{n(t,q)}] f(d,e), \tag{3.11}$$

$f(d,e)$ is $P(d|e)$ or $P(e|d)$ depending on if a document-centric or candidate-centric perspective is used.

The process of the model can be simplified to:

1. Given a collection of documents, determine a set of relevant documents D_q by ranking them according to the query, and calculating $P(q|d)$
2. Examine each document in D_q, and look at the entities associated with the document
3. Calculate $P(d|e)$ or $P(e|d)$ for each document
4. Finally calculate $P(q|e)$ by taking the sum for all documents of $P(q|d) \cdot P(d|e)$

Model 2 differs from Model 1 because an entity model is not directly created. Model 1 creates an entity model as a textual representation of the entity, and uses the representation to assess how probable the entity is to produce the given query. This is quite different from Model 2, where the documents act as a separator between the query and the entity. Another difference between the models is that Model 2 can be implemented on top of a standard document index, while Model 1 requires a separate entity-term index to be created and maintained. Model 2 was chosen for implementation, since it outperforms Model 1 on nearly all the measures [5].

Chapter 4

Implementation of System

This chapter will give a detailed explanation of the implementation of the system. The goal of the system is to perform entity retrieval on an annotated corpus, using language model and Model 2. The input of the system will be a set of queries, and the output should be a ranked list of entities. To achieve the project goals, we need to implement a system that preprocesses the ClueWeb data and creates a Lucene index of the preprocessed data. The index will later be used to evaluate the models and other baselines. There were several challenges with preprocessing the dataset due to heterogeneous data. The data consists of Web documents, which can be in different encodings and contains page text, css, xml, html, javascript and images. The size of the ClueWeb data made some optimizations necessary to get a reasonable execution time.

The pipeline of the system is demonstrated in Figure 4.1. Input is given to the preprocessor, which then cleans the data, and replaces the entity mentions with annotations. The cleaned data is given to the indexer, which creates the Lucene index.

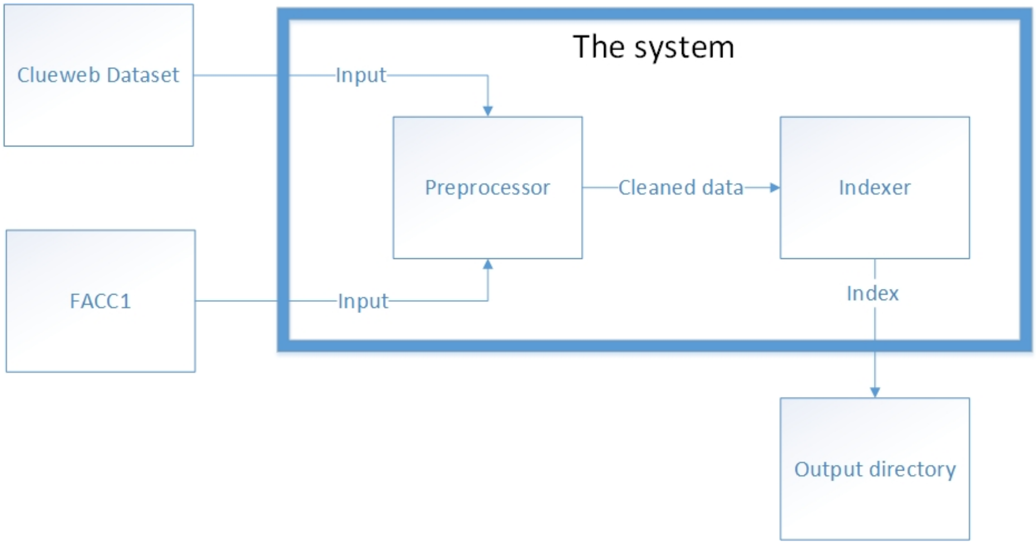


Figure 4.1: Pipeline of the System

4.1 Preprocessing of ClueWeb12

The first step in the system is to preprocess the data. This is divided into two tasks

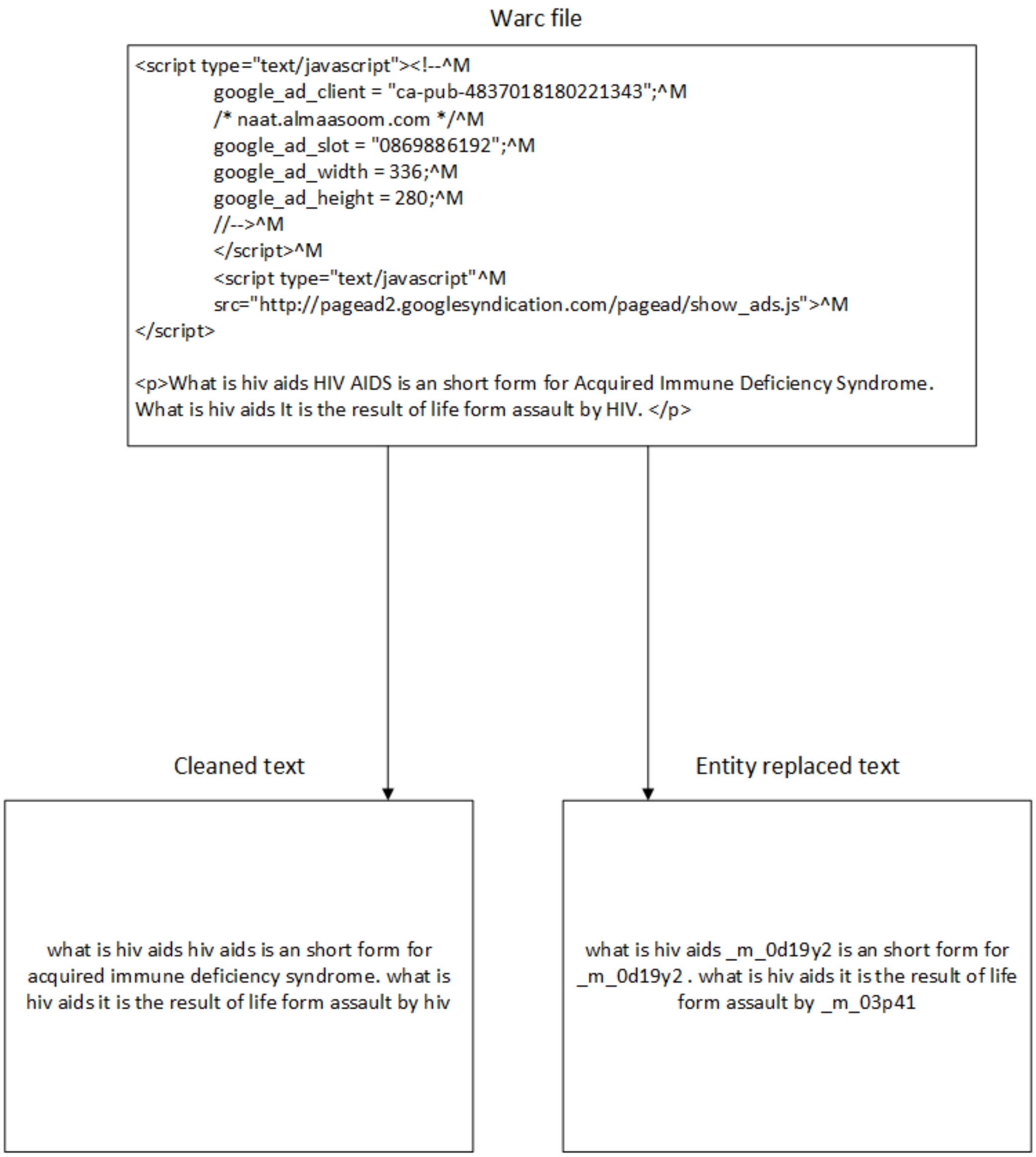
1. Finding the entity mentions in the ClueWeb data, and replacing them with the entities.

2. Cleaning up the ClueWeb data by converting the files to plain text.

The non-trivial task is the first task. Finding the entity mentions with the given offsets requires a system, that has to clean the entity mentions found and the annotations. There are also numerous edge-cases, that makes the task more difficult. The WARC files in the ClueWeb dataset contains records with several different encodings. The data is collected from Web pages, which contains HTML, CSS and JavaScript. They are all considered noise in the context of the preprocessing. We want to remove all the tags, and only keep the plain text.

We need to group each WARC file with its corresponding FACC1 file containing the annotations. Each record in a WARC file has a document identifier (WARC-TREC-ID). Table 2.3 from the FACC file shows that each annotation is also stored with a WARC-TREC-ID. The data is traversed, using the document identifier to find the annotations in the records. The byte offsets from the FACC1 are used to find the entity mentions. To make sure we have found the correct entity mention with the byte offsets, we compare the found entity mention with the entity mention in the annotation. The found entity mention is then replaced with the Freebase identifier. After the entity mentions are replaced, we clean the whole text and continue to the next record.

Figure 4.2 shows an excerpt of the preprocessing of record clueweb12-0000tw-00-00013 in the warc file 0000tw-00. Two copies of the content of the warc file is created, one with entities replaced in the content, and another with just the cleaned content. The cleaned text and the entity replaced text are both needed for the index. We can see in Figure 4.2 that the cleaning process removes everything contained in the script tag, and also removes the HTML tag, but keeps the content inside the HTML tag.



```
<script type="text/javascript"><!--^M
            google_ad_client = "ca-pub-4837018180221343";^M
            /* naat.almaasoom.com */^M
            google_ad_slot = "0869886192";^M
            google_ad_width = 336;^M
            google_ad_height = 280;^M
            //-->^M
            </script>^M
            <script type="text/javascript"^M
            src="http://pagead2.googlesyndication.com/pagead/show_ads.js">^M
</script>

<p>What is hiv aids HIV AIDS is an short form for Acquired Immune Deficiency Syndrome.
What is hiv aids It is the result of life form assault by HIV. </p>
```

what is hiv aids hiv aids is an short form for acquired immune deficiency syndrome. what is hiv aids it is the result of life form assault by hiv

what is hiv aids _m_0d19y2 is an short form for _m_0d19y2 . what is hiv aids it is the result of life form assault by _m_03p41

Figure 4.2: Excerpt of the preprocessing of the warc file 0000tw-00

4.1.1 Replacement of Entities

The first step was to find the entity mentions in the ClueWeb collection, and replace them with the Freebase IDs. This was achieved by going through all the FACC files for each WARC file, and locating the annotations. The FACC file containing the annotations, contained the byte offsets to each annotation. The byte offsets were used to find the entity mention. We also compared the found entity mention with the entity mention from FACC, to make sure we found the correct entity mention.

After a match was found for an annotation, the bytes containing the annotation was added to a list, annotation_bytes. The annotations (Freebase IDs) were added to a dictionary with the start positions of the byte offsets as the keys. After all matches for a record were found, a replacement content was created, which contains the record with all the entity mentions replaced with annotations. The method for doing the replacements:

```
replacement_content = bytearray()
for count, byte in enumerate(record_content):
    if count not in annotation_bytes:
        replacement_content.append(byte)
    else:
        if count in annotation_list:
            replacement_content+= annotation_list[count]
return replacement_content
```

This method is run for each record that contains annotations. record_content contains the original record from the WARC file. The next step in the system after the replacements is the cleaning of both the original record, and the replacement record.

4.1.2 Cleaning the Data

The cleaning process is divided into several steps:

1. Decode the text
   The documents in the warc files are encoded in many different encodings. The text needs to be decoded, and placed in a bytearray. The annotation's byte offsets are used to locate the entity mentions.

2. Strip HTML tags, JavaScript and CSS
   Lucene only accepts plain text files. It can not parse HTML documents. We also only need the title and contents of the documents, the rest is considered noise. A HTML parser is used to remove HTML-tags,

JavaScript and CSS from the text, but keep the content intact. E.g, the following text: “<title>An Example Page <title>”, would produce this output, “An Example Page”, after this step is completed.

3. Resolve HTML-entities
   HTML has some reserved characters that are replaced with character entities. The character entities are used to display reserved characters in HTML. This needs to be resolved, because we want the characters that the HTML-entities represent. Python’s built-in HTML Parser has an undocumented method for resolving entities that was used. An example of an entity is “&amp”, which will be resolved to “&”. It is important that the entities are resolved, when we compare the entity mentions to the FACC, which is written in plain text.

4. Replace multiple white spaces and remove punctuation
   This is just a simple cleanup step to keep the text short, and remove unnecessary characters.

5. Transliterate an Unicode object into an ASCII string
   Some of the annotations contain non-Roman text. The annotations needs to be transliterated when comparing the entity mentions in the FACC to ClueWeb.

6. Remove HTTP header
   The HTTP header is not needed for the Lucene index, so it is removed in the preprocessing phase. The byte offsets in the annotations are calculated from the zero location of the HTTP header. Therefore we wait until the entity mentions are found and replaced, before removing the header.

All this steps are performed when cleaning the data, and they are also used when comparing the entity found in the ClueWeb data, with the entity mention in the FACC data.

4.1.3 Challenges

The preprocessing proved to be more challenging than what the researcher assumed. Several different problems were found while implementing and testing the preprocessor. Most of the challenges are connected to the first task, finding and replacing entity mentions, which is described in Chapter 4.1.1. The challenges are described and solutions are proposed in the following paragraphs.

Encoding

There were several problems with the encoding. The original encoding in the FACC are not always correct. If the document is decoded with the wrong encoding, some of the bytes will not be recognized. This will either give an error, or replacement characters will be used, depending on the type of flag used when decoding. To further illustrate this problem we will use a real case, from the record clueweb12-0505wb-25-04144 in the 0505wb-25 warc file. In the meta tag, the charset is set gb2312, which is a character set for Chinese characters. Table 4.1.3 shows an excerpt from the FACC file, that illustrates the issue.

clueweb12-0505wb-25-04144	UTF-8	Xizhimen	16914	16922	/m/03sv1y

Table 4.1: Excerpt from 0505wb-25.anns.tsv

The second column is the encoding that was used to process the entry when creating the annotations. A different encoding was used, and this is true for several of the annotations. A simple solution would be to use the same encoding as the one provided in the FACC data. However, we want the data cleaned, and need to use the correct encoding to get readable text. The problem this introduces is that the byte offsets, for the entity mention, have been set using the encoding from the excerpt. This means that when we decode the clueweb12-0505wb-25-04144 with the correct encoding, we will get a different amount of bytes. That makes it difficult to find the correct entity mention, and replace it.

The solution to this was to first try the FACC encoding, and then try with the different common encodings, "UTF-8", "ISO-8859-1", and "windows-1252". Additionally cchardet [9], a python library for detecting encoding of text, is used in the worst-case scenario when no match is found with the common encodings. The reason for not using detection as the first option, is that detecting the encoding is time consuming. There is also no guarantee that the detected encoding is the actual encoding of the document, it is merely an educated guess made by the library.

Tags

HTML-tags had to be removed from the documents. This was done using Beautiful Soup 4 [1], which is a python library for pulling text from HTML files. LXML [2] was used as the HTML-processor. The reason for using Beautiful Soup was that it is easy to use, and also provided methods for stripping away script and style-tags. HTMLParser was first used, since it is a part of native Python, but the performance is worse than LXML, and additionally HTMLParser was not able to remove all the different tags.

Byte Order Mark

The entity mentions found were in some cases wrong. This was caused by the fact that some of the records in the WARC files contain a byte order mark (BOM). BOM is a set of bytes at the start of the text stream, that is used to signal which encoding the document is in [40]. The set of bytes represent the unicode code point U+FEFF. When decoding the document in python with UTF-8 encoding, you get a unicode string with BOM. The byte offsets for the annotations were found with the BOM ignored. This caused a problem were in some cases, the byte offsets found the wrong entity mention, because of the added bytes from the BOM. It was a simple problem to fix when discovered. Python has a codec called “utf-8-sig” in its list of codecs [17]. The “utf-8-sig” codec removes the BOM when decoding a text. We added the codec to the list of possible encodings.

ASCII representation

Step 5 of the preprocessing steps, the transliteration of Unicode objects into ASCII strings, introduced some challenges. In some cases, when trying to match the entity mention in FACC to ClueWeb, the characters in the ClueWeb document could not be represented with ASCII. The found entity in ClueWeb could contain a unicode character, that is not in the first 127 ASCII characters. This made it difficult to match the entity mention with FACC. The solution to Step 5 was to use a Python library called unidecode

[1] https://www.crummy.com/software/BeautifulSoup/

[2] http://lxml.de/

[37] that tries to transliterate Unicode to ASCII. Unfortunately it is hardly perfect, and after testing the system on the ClueWeb12 data there were several cases, where the library did not manage to transliterate the characters. However, This happened in a fairly low amount of cases, and always with the same characters. A quick solution was to add an ad-hoc method, that checked for those specific characters, and transliterated them. An excerpt of the method:

```
for character in text:
    if character == u'\u03bc':
        character = 'u'
    elif character == u'\u03f5':
        character = 'e'
```

The unicode characters “\u03bc” and “\u03f5” is instead represented by the characters “u” and “e”, which ASCII knows how to represent.

Unicode Replacement Characters

The last problem was the most challenging one. As mentioned earlier, FACC1 did not always use the correct encoding when finding the annotations. This proposed a challenge when working with Asian encodings in ClueWeb documents. When decoding a text in Python, a replacement flag can be set, that tells the decoder to use replacement characters if it not able to decode some of the text. However, when using the correct encoding stated in the ClueWeb document, the byte offsets would find the wrong entity mention. This is related to FACC1 using the wrong encoding to find some of the entity mentions, so the number of bytes of the decoded text is different.

The problem with using the same encoding as stated in FACC1, is that Python 2.7 does not conform with the Unicode 5.2.0 standard, regarding the replacement characters. An issue was raised in the Python bug tracker and resolved [38]. Unfortunately this issue was fixed only for Python 3.4 and Python 3.3, but not for Python 2.7, which is the Python version the researcher is using. The researcher is assuming that the researchers at Google used Java, or another language that conforms with the Unicode standard, to decode the ClueWeb document. The challenge occurred when the ClueWeb document had one of the following encodings: EUC_KR, SHIFT_JIS, BIG5, GB18030, ISO-8859-9. The documents with those encodings were few, and the entity mention were always located with the same number of bytes from the byte offsets.

Since all options were exhausted, and time-constraints had to be considered, a solution was proposed. We would add a set number to the byte offsets, whenever the problematic encodings were encountered. This proved to give the correct location for the entity mention. This was thoroughly tested on the troublesome documents, to see that the entity mentions were correctly replaced with the Freebase identifiers.

4.2 Indexer

4.2.1 Lucene

Lucene[3] is a high-performance full-text search engine library written in Java. It is available as an Open Source Software under the Apache License. There are several implementations of Lucene written in other languages than Java.

PyLucene[4], a GCJ-compiled version of Java Lucene integrated with Python, is used for indexing the ClueWeb12 collection. PyLucene is a Python wrapper, that embeds a Java VM with Lucene into a Python process. This makes it possible to use the Lucene library in Python.

4.2.2 Indexing Process

The goal of the indexer is to create a Lucene index, that will be used to evaluate the models. We want to create the index with three fields:

- WARC-TREC ID
- Cleaned text with entity mentions replaced with Freebase Identifiers
- Cleaned text

The preprocessed data is collected in a list, and each document in the list is added to the index with the three fields specified. The text is added to contents with field name, field value and field type. The contents are then added to the index as a Lucene document. A document in Lucene, is a set of fields with a name and a value. The field, WARC-TREC ID, is the unique identifier we store for each document in Lucene.

[3] https://lucene.apache.org/core/

[4] http://lucene.apache.org/pylucene/

The indexer creates a Lucene index for each subfolder in the ClueWeb collection. This was done instead of creating one single big index, to make it possible to parallelize the indexing process. It is also helpful in case of errors in the index. The troublesome subfolders can be re-indexed, instead of indexing the whole collection. After the indexes have been created, they are merged by the merge indexer. Lucene has a method called “addIndexes” that makes it possible to merge several indexes into one single index.

4.2.3 Testing the Index

We had three requirements to the index that had to be tested thoroughly, before we could be sure that the index was created successfully.

1. Check that the correct entity mentions are replaced with the correct Freebase IDs
2. Check the number of documents in the index, and compare it to the WARC files
3. Number of terms should match in the fields “contents” and “contents_annotated”

Requirement 1 was checked manually by using Luke [5], a GUI tool for Lucene. Luke makes it possible to open an index, browse the documents, field contents, and much more. The way we tested the requirement was by opening a sample of documents, then retrieving the term vectors for the field “contents”, and the field “contents annotated”. The term vectors contains the terms, location and number of occurrences. This was used to compare the number of occurrences of the Freebase IDs, with the original FACC files.

Requirement 2 showed that the Warc library had some limitations on its ability to read all records in a WARC file. In several cases, it was unable to read the last few records in some of the WARC files. This was solved by making a new method to read warc records, that overwrites the original method from the Warc library. The problem with gzip WARC files not being properly read has been raised on the github page for the library [6].

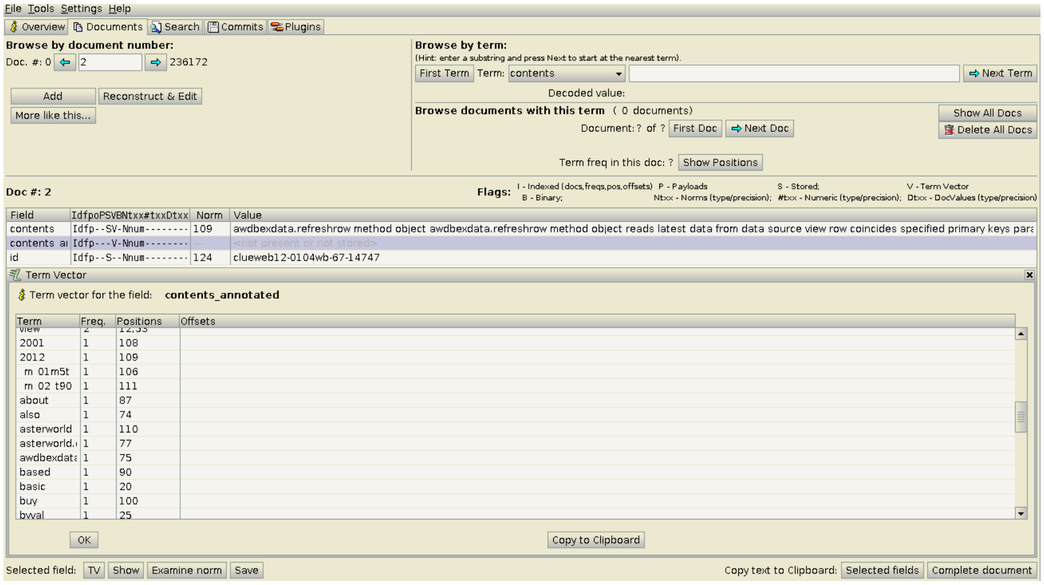
Requirement 3 was checked in a similar fashion as the first requirement. We used the term vectors again to compare the number of terms in the

[5] https://github.com/DmitryKey/luke

[6] https://github.com/internetarchive/warc/issues/21

fields “contents” and “contents annotated”. It is important that the term vectors match. To illustrate it better, we will use an example. Let us assume that FACC annotated “HIV” once in a given document, then the number of occurrences in “contents annotated” for that document would be:
“HIV” occurrences in “contents” – “HIV” entity mentions.

Figure 4.3 shows a screenshot of the Luke GUI, with the term vectors for the field, “contents_annotated”, for the document clueweb12-0104wb-67-14747. This is how requirement 1 and requirement 2 was checked through Luke, by copying the term vectors to the clipboard, and then comparing the term vectors for “contents” and “contents_annotated”.



Browse by document number:
Doc. #: 0 2 236172

Browse by term:
(Hint: enter a substring and press Next to start at the nearest term).
Term: contents
Decoded value:

Browse documents with this term (0 documents)
Document: ? of ?
Term freq in this doc: ?

Doc #: 2

Flags: I - Indexed (docs,freqs,pos,offsets) P - Payloads S - Stored; V - Term Vector
B - Binary; Ntxx - Norms (type/precision); #txx - Numeric (type/precision); Dtxx - DocValues (type/precision)

Field	IdfpoPSVBNtxx#txxDtxx	Norm	Value
contents	Idfp--SV-Nnum--------	109	awdbexdata.refreshrow method object awdbexdata.refreshrow method object reads latest data from data source view row coincides specified primary keys para
contents aı	Idfp---V-Nnum--------	---	<not present or not stored>
id	Idfp--S--Nnum--------	124	clueweb12-0104wb-67-14747

Term vector for the field: contents_annotated

Term	Freq.	Positions	Offsets
view	2	12,33	
2001	1	108	
2012	1	109	
m 01m5t	1	106	
m 02 t90	1	111	
about	1	87	
also	1	74	
asterworld	1	110	
asterworld.ı	1	77	
awdbexdatı	1	75	
based	1	90	
basic	1	20	
buy	1	100	
bwal	1	25	

Selected field:
Copy text to Clipboard:

Figure 4.3: Luke GUI for Lucene

4.3 Scorer

The last part of the system was the implementation of a module that scores and retrieves documents. Model 2 was implemented as the scorer. The model is explained in detail in Chapter 3.2.2. The scorer takes a list of queries as the input, and the path to a Lucene index directory. The output is written in TREC format, to make it possible to evaluate the results against relevance judgements. The scorer calculates $p(q|e)$ for the given queries and entities.

To avoid having to score each document in the index for each query, which would be computationally expensive, we will use Lucene to retrieve the top n documents from the collection for each query. n is a parameter that has to be specified before running the scorer. The query likelihood $p(q|d)$ is calculated for the top n documents using a standard language modeling approach. D_q is the list of top n documents with $p(q|d)$ scores calculated for each document.

$p(e|d)$, the likelihood of the entity given the document, is calculated using equation 18 from Balog et al. [6]. The implementation of Model 2 is depicted in Algorithm 4.1:

```
for each query $q$:
    top_n_docs = Lucene.score(q, n)
    $D_q$ = language_model.score(q, top_n_docs)
    for each document $d$, $p(q|d)$ in $D_q$:
        for each entity associated with document d:
            $p(e|d) = \frac{N(e,d)}{\sum_{e'} n(e',d} \cdot log \frac{|D|}{|d':n(e,d')>0|}$
        if $p(e|d) > 0$:
            $p(q|e) + = p(q|d) \cdot p(e|d)$
```

Algorithm 4.1: Model 2 implementation

4.4 Optimization

The ClueWeb B13 collection consists of 33 447 warc files, with a compressed size of 389 GB, and an uncompressed size of 1,95 TB. Since the collection is very large, it was necessary to look at ways to optimize the system. The preprocessing of the whole collection is a CPU-expensive task, considering all the steps that have to be executed.

The preprocessing had to be parallelized as it was a bottleneck. To achieve parallelization in Python, it is necessary to use the built-in multiprocessing module to fork multiple processes that execute in parallel. Python threads provide I/O interleaving, but are executed in serial due to the global interpreter lock. GIL is a lock that prevents two or more native threads in Python from executing Python bytecodes simultaneously. This lock is necessary due to CPython's memory management not being thread-safe [18]. CPython is the implementation of Python used in this project.

The multiprocessing module creates a pool of workers, with a python function called map, that allows us to map arguments to a function. The problem with map is that it does not support multiple arguments. In Python 3.3 multiple arguments for map is implemented in a function called starmap, but unfortunately this is not available for Python 2.7. However, there are libraries available through pypi, the Python package index, that implements the same functionality for Python 2.7. This was implemented, and made it possible to use multiple processors to read and preprocess the ClueWeb collection.

Parallelizing the indexing process introduced some challenges. It is only possible to have one IndexWriter on a directory, since the Lucene IndexWriter creates a write lock on the directory. The goal was to use several processors to add documents to the index, since the add document method in Lucene is thread-safe. The first challenge was that the multiprocessing module in Python needs to be able to pickle the objects used. Pickling in Python is another word for serializing. Pickling is the process where a Python object is translated into a byte stream. The pickling module was unable to pickle the Lucene IndexWriter.

Instead of spending to much time trying to parallelize the indexer, we decided to run the system in several instances. In that way, the preprocessing is parallelized by the number of processors specified, and the indexing is running in parallel depending on how many instances of the system that is currently running.

4.5 Technical Details

This section will contain the technical details of the system, and the time needed for the different processes.

The system was run on Svein Erik Bratsberg's computer, mersey3, at NTNU. Table 4.2 shows the hardware specifications of the computer.

Memory	4 * 8GB DIMM DDR3 Synchronous 1600 MHz
Hard Disk	Seagate Desktop HDD ST2000DM001-1CH1, 64MB Cache SATA 6.0Gb/s
Processor	Intel Core i7-4779 CPU @ 3.40 G, 16 cores

Table 4.2: Hardware specifications of mersey3

The required run-time of preprocessing ClueWeb B13 and indexing the preprocessed data is approximately 8 days. The size of ClueWeb B13 compressed is 389 GB, and uncompressed is 1.95 TB. However, the system is able to read the WARC files compressed, therefore it is not necessary to decompress the files before running the preprocessing and indexing. The FACC dataset is 449 GB, therefore it is necessary to have at least 838 GB storage available for the datasets. The Lucene index created by the system after preprocessing and replacing entity mentions is approximately 422 GB. Since we first index each subfolder of ClueWeb B13, before merging the indexes, it is necessary to have enough storage for the individual subfolders, and also the final merged index. The indexing procedure requires around 844 GB available storage.

The required time to run the Model 2 scorer for all the queries (485) in the DBpedia entity test collection, when scoring the top 1000 documents for each query, is approximately 490 minutes.

4.6 System Description

Figure 4.4 illustrates a simple view of the finished system. The implementation of the system is publicly available at `https://github.com/Tino92/tino-thesis`.

The following is a description of the modules and classes in the system.

clueweb_facc_preprocessor
The clueweb_facc_preprocessor module contains two classes, Annotation and WarcEntry. The Annotation class is used to store and parse annotations. The WarcEntry class contains all the methods related to a warc file, to be able to replace annotations and clean the records.

indexer
The indexer module contains only one class. The class is named Indexer. The indexer has a main method that takes in arguments for the input path of ClueWeb and FACC, the output path of the Lucene index, and the number of processes that should run the system. It is responsible for reading and indexing files. The replacement and cleaning of entities are delegated to the WarcEntry.

merge_indexer
The merge_indexer is made as a separate module with its own main method. It takes in arguments for the index directory, and the output directory of the merged index. The module was made separate, instead of just as a method in the indexer, since it is not specific to the ClueWeb or FACC datasets. It can be used separately if Lucene indexes needs to be merged.

clueweb_facc_scorer
This module contains the implementation of Model 2. The module consists

of one class, Model2, which handles loading queries, the Lucene index, and the associations. It also contains a method for scoring all the queries.

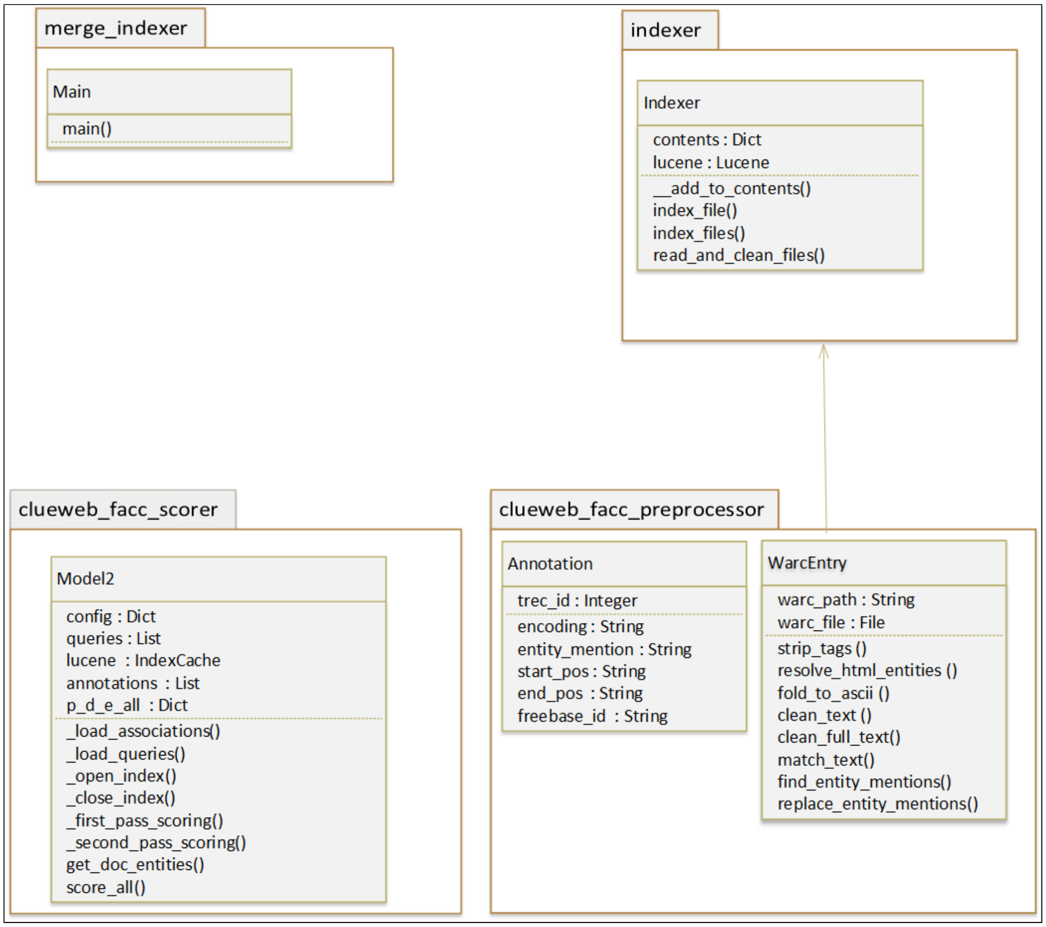


Figure 4.4: Class diagram of the system

Chapter 5

Results and Discussion

This chapter will describe the evaluation methods that were used, and present the results that were produced from the experiments. The results will be analysed.

5.1 Test Collections

The evaluation will be performed on three different test collections from different sources. This section will contain the details to each test collection.

DBpedia Entity Test Collection [4]
In our experimental setup we preprocessed the ClueWeb B13 collection, and created a Lucene index. The test collection used for the experiments is publicly available [4]. The queries and the corresponding relevance judgements from the test collection, have been collected from six different entity retrieval evaluation campaigns. The total number of queries is 485. The test collection is described in more detail in Chapter 2.4.3.

Retrieval results are obtained for the following query sets:

- **INEX-LD**: Covers named entity queries, type queries, relation queries, and attribute queries, (e.g. “vietnam war movie”, “List of films from the surrealist category”).
- **SemSearch ES**: Consists of mainly named entity queries, (e.g. “ben franklin”, “carl lewis”).

- **ListSearch**: Combines three query sets (INEX-XER, SemSearch LS, TREC Entity), which primarily consists of type queries (e.g. "Hugo awarded best novels", "films shot in Venice").
- **QALD-2**: Contains natural language questions of 4 different types: e.g., entity: "Who developed Skype?", type: "Give me all companies in Munich.", attribute: "Which awards did WikiLeaks win?", relation: "Who is the husband of Amanda Palmer?".

Table 5.1 shows the statistics of the query sets used:

Query set	**Number of queries**	**Query types**
INEX-LD	100	entity, type, attribute, relation
SemSearch ES	130	entity
ListSearch	115	type
QALD-2	140	entity, type, attribute, relation

Table 5.1: Statistics of the queries [42]

The knowledge base used for the test collection is DBpedia[1]. We want to use Freebase [19] instead, since the ClueWeb annotations are using Freebase. Therefore a script was created that take a qrels file and Freebase links as input, and creates a new qrels file with Freebase entities replacing the DBpedia entities. The Freebase links consists of links between DBpedia and Freebase entities.

However, Freebase does not contain all the entities in DBpedia. Hence, The qrels file with Freebase entities do not contain all the relevance judgements from the original qrels file. The DBpedia entities that do not have corresponding Freebase entities were excluded from the original qrels file, and also for the run files for the models from Balog and Neumayer [4]. The new run files and qrels file, which excludes the DBpedia entities not found in Freebase, were run with TrecEval to create new evaluation results.

REWQ Robust [36]

The document collection used for this test collection is the TREC Disk4+5. The queries are collected from the TREC Robust Track '04. There are a total of 25 queries, with 50 relevance judgements per query, which makes a total of 1250 relevance judgements.

[1] http://wiki.dbpedia.org/

REWQ ClueWeb [36]
The document collection for this test collection is the ClueWeb12 dataset, which we used to create the Lucene index. The queries are collected from the TREC Web Track '13/14.

Table 5.2 shows some sample queries, with some of the corresponding relevance judgements for both of the collections.

Dataset	**Query**	**Relevance Judgements**
Robust04	poliomyelitis and post polio	Centers_for_Disease_Control_and_Prevention, China,
Robust04	killer bee attacks	Africa, Africanized_bee, Agriculture, Nevada
ClueWeb12	rules of golf	8_Simple_Rules, Golf, History_of_golf, Rules_of_golf
ClueWeb12	what is madagascar known for	Africa, Madagascar_dry_deciduous_forests

Table 5.2: Excerpt from REWQ Robust04 and REWQ ClueWeb12

5.2 Evaluation

TREC[2] is an evaluation campaign that focuses on different information retrieval research areas. It is co-sponsored by the National Institute of Standards and Technology (NIST) and the Intelligence Advanced Research Projects Activity (IARPA). The goal of TREC is to support and encourage research in IR by providing the infrastructure for evaluation of IR systems. This includes providing sample corpora, queries and tools to benchmark retrieval systems.

We will be using one of the resources provided by TREC, specifically a benchmark tool named TrecEval. It is a text utility that evaluates the effiency of information retrieval systems, and allows evaluation of TREC results. The input arguments that TrecEval accepts are the ground truth, in our case this is the qrels file which has been mentiond in Chapter 2.4.3, and the results from running our system.

The ground truth has to follow the format in Table 5.3.

query-number	Q0	document-id	relevance

Table 5.3: Ground truth format for TrecEval

The results obtained from the experiments has to be in the format shown in Table 5.4

[2] http://trec.nist.gov/

query-number	Q0	document-id	rank	score	run-id

Table 5.4: Results format for TrecEval

The evaluation measures we will use are different for each dataset. REWQ Robust04 and REWQ ClueWeb12 will use the same measures, namely MAP, P@10, NDCG and NDCG@10. The measures are calculated by TrecEval. The qrels for REWQ Robust04 and REWQ ClueWeb12 uses multiple levels of relevance, therefore they use NDCG, normalized discounted cumulative gain. NDCG uses graded relevance as a measure of gain, which is accumulated by starting at the top of the ranking, and reducing the gain for the lower ranks. This takes into consideration that ideal ranking should first return the documents with the highest relevance level. MAP, mean average precision, is the mean of the average precision for each query. P@10, precision at 10 documents, is the precision score after 10 documents have been retrieved. Precision is a score that is calculated by dividing the relevant documents retrieved with all the retrieved documents. The reason for using P@10 is that it is a useful measure for Web search engines, since users expect to find the relevant results on the first search results page.

For the DBpedia entity test collection we will use MAP, P@10, P@20 and b-pref. NDCG will not be used, since the qrels only contain binary relevance. P@20 is the precision score after 20 documents have been retrieved. b-pref is a measure designed to consider whether relevant documents are ranked above irrelevant ones.

5.3 Results

The results for the test collections are presented in this section. MAP, P@10, P@20 and b-pref measures for the DBpedia test collection is presented in Table 5.5. The results are categorised for the query sets described in Chapter 5.1, and finally results for all the queries combined are presented. The baselines are presented in the first five rows, which are provided from the DBpedia entity test collection paper [4]. The last row for each query set are the results achieved from running the implementation of Model 2.

Method	**SemSearch ES**			
	MAP	**P@10**	**P@20**	**b-pref**
LM [4]	0.3329	0.2372	0.1682	0.6852
BM25 [4]	0.3452	0.2419	0.1655	0.6935
MLM-tc [4]	0.3747	0.2643	0.1829	0.7178
BM25F-tc [4]	0.3534	0.2504	0.1674	0.7001
PRMS [4]	0.3542	0.2434	0.1686	0.6936
Model 2	**0.1475**	**0.1217**	**0.0771**	**0.3421**
	ListSearch			
	MAP	**P@10**	**P@20**	**b-pref**
LM [4]	0.1700	0.2235	0.1739	0.4031
BM25 [4]	0.1786	0.2383	0.1865	0.4202
MLM-tc [4]	0.1641	0.2104	0.1661	0.3979
BM25F-tc [4]	0.1711	0.2278	0.1791	0.4081
PRMS [4]	0.1923	0.2504	0.2057	0.4179
Model 2	**0.0677**	**0.1122**	**0.0983**	**0.3123**
	INEX-LD			
	MAP	**P@10**	**P@20**	**b-pref**
LM [4]	0.1082	0.2180	0.1800	0.3058
BM25 [4]	0.1274	0.2350	0.1935	0.3304
MLM-tc [4]	0.1076	0.2170	0.1835	0.2998
BM25F-tc [4]	0.1267	0.2410	0.1885	0.3255
PRMS [4]	0.0965	0.1950	0.1570	0.2808
Model 2	**0.0374**	**0.1030**	**0.0725**	**0.1583**
	QALD-2			
	MAP	**P@10**	**P@20**	**b-pref**
LM [4]	0.1141	0.0568	0.0442	0.2360
BM25 [4]	0.1276	0.0719	0.0547	0.3149
MLM-tc [4]	0.1051	0.0532	0.0414	0.2421
BM25F-tc [4]	0.1121	0.0683	0.0525	0.2956
PRMS [4]	0.1120	0.0727	0.0536	0.2633
Model 2	**0.0678**	**0.0371**	**0.0321**	**0.4074**
	All queries			
	MAP	**P@10**	**P@20**	**b-pref**
LM [4]	0.1846	0.1781	0.1363	0.4102
BM25 [4]	0.1978	0.1907	0.1444	0.4443
MLM-tc [4]	0.1917	0.1810	0.1383	0.4182
BM25F-tc [4]	0.1936	0.1907	0.1415	0.4366
PRMS [4]	0.1926	0.1859	0.1419	0.4187
Model 2 DBpedia v3.7 Freebase links	**0.0863**	**0.0952**	**0.0714**	**0.3302**
Model 2 DBpedia v3.9 Freebase links	**0.0903**	**0.0975**	**0.0731**	**0.3397**

Table 5.5: Evaluation measures for DBpedia entity test collection

The results for Model 2 are low compared to all the baselines for all the query sets. Model 2 has lower scores for the different evaluation measures. We decided to evaluate Model 2 on different datasets, to further analyse the reason for the low results.

In Table 5.6, the results for REWQ Robust04 and ClueWeb12 datasets are presented. Two methods are presented, Model 2 that we implemented, and RankLib from Schuhmacher et al. [36]. RankLib[3] is a library of learning-to rank algorithms. The coordinate ascent algorithm as implemented in RankLib is used.

Method	**REWQ Robust04**			
	MAP	**P@10**	**ndcg**	**ndcg@10**
Model 2	0.0645	0.2640	0.2468	0.2838
RankLib [36]			0.936	0.817
	REWQ ClueWeb12			
Model 2	0.2637	0.3676	0.4674	0.4738
RankLib [36]	0.328		0.572	0.710

Table 5.6: Evaluation measures for REWQ Robust04 and Cluweb12 datasets

The results demonstrate that the datasets that the test collections are based on, are highly influential on the evaluation measures. The REWQ ClueWeb12 dataset, which uses FACC1 for entity annotations, gave the best results for Model 2, with MAP score of 0.2637 and P@10 score of 0.3676. The lowest results were achieved for the DBpedia test collection, with a total MAP score of 0.0903 and P@10 score of 0.0975 for all the queries. The NDCG and NDCG@10 scores were only calculated for the REWQ datasets, and also here the scores are higher for ClueWeb12 compared to Robust04. ClueWeb12 achieves higher NDCG scores with a NDCG score of 0.4674 and NDCG@10 score of 0.4738, compared to Robust04 with NDCG score of 0.2468 and NDCG@10 score of 0.2838.

It is apparent how big the difference is between the results for REWQ ClueWeb12 and DBpedia test collection. We argue that the low results for the DBpedia test collection can be explained with the relevance judgements being created from DBpedia, and some of them not having any entry in the FACC collection. The high results for REWQ ClueWeb12 can be attributed

[3] http://people.cs.umass.edu/ vdang/ranklib.html

to the fact that the document collection used is ClueWeb12 with FACC, which is the same collection we used to create our index.

5.4 Analysis

In this section, we will analyse different aspects of the results. The results for the DBpedia test collection were lower than all the baselines, and will be thoroughly examined in the analysis. We will also discuss the differences in results for the datasets.

Analysis of recall for DBpedia entity test collection
We decided to evaluate Model 2 for top 100, 1000 and 10000 documents. We will use the measures: Recall, MAP and P@10. Recall is the fraction of relevant documents that are retrieved. The entities that are not retrieved from the top documents, will not be scored by the model.

In Table 5.7, we have run Model 2 for the top 100, 1000 and 10000 documents for each query. Recall, MAP and P@10 shows an increase from top 100 to top 1000 documents. However, this is not the case for the top 1000 and top 10000 documents. Similarly MAP and P@10 have not improved by increasing top-n documents to 10000.

The results suggest that increasing the number of relevant documents, can improve entity retrieval performance until a certain point (i.e n=1000). After that, the number of relevant entities and consequently the MAP and Precision are not improved. We attribute this to the fact that the ClueWeb collection provide different context for the entities than DBpedia, and this can be another evidence that the entity retrieval performance can be highly dependent on the corpus entities are extracted from.

top n documents	**Recall**	**MAP**	**P@10**
100	0.1101	0.0824	0.0893
1000	0.1245	0.0903	0.0975
10000	0.1253	0.0902	0.0942

Table 5.7: Recall for top n documents when retrieving all entities associated with the documents.

Analysis of query subsets for DBpedia test collection

There are some differences in the evaluation measures for each query set. The query sets are of varying difficulty, which the baselines clearly show. SemSearch ES gives the highest MAP and P@10, followed by ListSearch, INEX-LD and finally QALD-2. SemSearch ES consists of mainly named entity queries, (e.g. "ben franklin"), which are easier to rank since the entities are named in the query. This demonstrates that named entity queries are easier to retrieve for Model 2, and also for the baselines. P@10 and P@20 is fairly similar for SemSearch ES, ListSearch and INEX-LD, while it has a much lower score for QALD-2. This could be explained by the fact that QALD-2 contains natural language questions. It is apparent from the results that QALD-2 is the most difficult query set for the different models.

The problem is that Model 2 has a lower score for all the query sets compared to the baselines. Model 2 follows the same trend as the baselines, with a lower score for relation or attribute queries, and a higher score for named entity queries. P@10 is approximately 50 % lower for Model 2 compared to MLM-tc for SemSearch ES, ListSearch and INEX-LD. For the query set QALD-2, P@10 is 30 % lower for Model 2 compared to MLM-tc. b-pref is actually higher for Model 2 for the query set QALD-2, compared to the baselines. The final score for b-pref for all the queries for Model 2 is 19 % lower, than the b-pref score of MLM-tc. This is a much lower difference, than the difference between Model 2 and MLM-tc for the other measures.

Since Model 2 follows the same trend as the baselines for the retrieval scores, it points to the dataset being the deciding factor that could explain the low results achieved for Model 2.

Comparison of Model 2 and MLM-tc for sample queries

We will do an error analysis by comparing Model 2 with MLM-tc, since MLM-tc is also based on language model, it will be a meaningful comparison. For each of the query sets, two queries are picked, and the number of relevant entities retrieved is compared for both models. Equation 2.4 shows the formula for MLM.

The approach we used to Model 2 is the candidate-centric approach, where we calculate $P(q|e)$. This is accomplished by first retrieving the top N documents based on the query likelihood, $P(q|d)$, which is calculated using the standard language model approach. This has already been implemented and is a well-known model. The focus of this discussion will be on how $P(e|d)$

is calculated in Model 2. The query and entity are separated in the model, using the document as the common connection between the query and entity probabilities. From Model 2 we can expect that the score will increase based on how many times an entity is mentioned in the relevant documents.

Table 5.8 presents the results from running Model 2 and MLM-tc on two sample queries for each query set. This was performed to determine differences in the models and the querysets. num_rel is the number of relevance judgements for the query, and num_rel_ret is the number of relevance judgements that were retrieved. There are some differences apparent in the table. MLM-tc has a higher retrieval than Model 2 on most of the queries. We can see that Model 2 is able to retrieve at least one relevant entity for all the queries, the primary problem is being able to retrieve all the relevant entities.

			Model 2	**MLM-tc**
Query sets	**Query**	num_rel	num_rel_ret	num_rel_ret
INEX-LD	vietnam war movie	52	4	19
	John Lennon Yoko Ono album Starting Over	8	0	7
SemSearch ES	el salvador	24	5	12
	university of phoenix	5	4	3
ListSearch	toy train manufacturers that are still in business	39	2	19
	Organizations that award Nobel prizes.	8	6	2
QALD-2	Who is the governor of Texas?	1	1	1
	Give me all films produced by Hal Roach.	504	1	88

Table 5.8: Evaluation of Model 2 and MLM-tc for sample queries from each query set

Chapter 6

Conclusion & Future Work

6.1 Conclusion

The goal of the thesis is to perform entity retrieval on entity annotated corpus. To achieve that goal we need to develop a system that is able to preprocess the ClueWeb corpus, and replace entity mentions in ClueWeb with the corresponding annotations from FACC. The system should also index the preprocessed data, and finally Model 2 should be implemented and evaluated. Entity retrieval was successfully performed on the annotated corpus, and Model 2 was implemented and tested on three different test collections, DBpedia entity test collection, REWQ ClueWeb12 and REWQ Robust04.

The contribution of this thesis is the system that were made to preprocess and index the ClueWeb corpus, and the results from the experiments. The results were presented and discussed in detail. This thesis has shown that the results of Model 2 is very dependent on the test collection used. Model 2 is able to achieve decent results when using the REWQ ClueWeb12 dataset, where the mean average precision is comparable to the learning-to-rank method RankLib, which has a slightly higher score. We showed that Model 2 achieved lower scores than the baselines for the DBpedia entity test collection. We argued that this is partly caused by the test collection being made for DBpedia, while we are using FACC for our index.

We can conclude that the project goals of this thesis were fulfilled. We have successfully performed entity retrieval, using Model 2, on the entity annotated corpus. Model 2 has been evaluated on three different test collections, and compared to several baselines. With the highest results for Model 2 achieved for REWQ ClueWeb12, giving MAP score of 0.2637, P@10 score of 0.3676, NDCG score of 0.4674 and NDCG@10 score of 0.4738.

6.2 Future Work

We have suggested some future work that became apparent during the work on the thesis.

- Testing the other expert retrieval model (i.e. Model 1) for the same test collections.
- Improving the cleaning and indexing execution time by looking at optimization measures.
- Indexing the whole ClueWeb12 dataset, instead of the ClueWeb12 B13 dataset which is a 7 % uniform sample. A larger dataset could affect the results, and would be meaningful to compare with the results from this thesis.

Bibliography

[1] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 020139829X.

[2] K. Balog, P. Serdyukov, and A. P. de Vries. Overview of the TREC 2010 entity track. In *Proceedings of the Nineteenth Text REtrieval Conference (TREC 2010)*. NIST, February 2011.

[3] K. Balog, P. Serdyukov, and A. P. de Vries. Overview of the TREC 2011 entity track. In *Proceedings of the Twentieth Text REtrieval Conference (TREC 2011)*. NIST, February 2012.

[4] Krisztian Balog and Robert Neumayer. A test collection for entity search in dbpedia. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 737–740, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2034-4.

[5] Krisztian Balog, Leif Azzopardi, and Maarten de Rijke. Formal models for expert finding in enterprise corpora. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 43–50, New York, NY, USA, 2006. ACM. ISBN 1-59593-369-7.

[6] Krisztian Balog, Leif Azzopardi, and Maarten de Rijke. A language modeling framework for expert finding. *Inf. Process. Manage.*, 45(1): 1–19, January 2009. ISSN 0306-4573.

[7] David Bounie and Laurent Gille. Info capacity— international production and dissemination of information: Results, methodological issues

and statistical perspectives. *International Journal of Communication*, 6 (0), 2012. ISSN 1932-8036.

[8] David Carmel, Ming-Wei Chang, Evgeniy Gabrilovich, Bo-June (Paul) Hsu, and Kuansan Wang. Erd'14: Entity recognition and disambiguation challenge. *SIGIR Forum*, 48(2):63–77, December 2014. ISSN 0163-5840.

[9] cchardet. Universal encoding detector. https://pypi.python.org/pypi/cchardet/1.0.0, 2015. [Online; accessed 27-November-2015].

[10] Yen-Pin Chiu, Yong-Siang Shih, Yang-Yin Lee, Chih-Chieh Shao, Ming-Lun Cai, Sheng-Lun Wei, and Hsin-Hsi Chen. Ntunlp approaches to recognizing and disambiguating entities in long and short text at the erd challenge 2014. In *Proceedings of the First International Workshop on Entity Recognition &#38; Disambiguation*, ERD '14, pages 3–12, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3023-7.

[11] Marco Cornolti, Paolo Ferragina, Massimiliano Ciaramita, Hinrich Schütze, and Stefan Rüd. The smaph system for query entity recognition and disambiguation. In *Proceedings of the First International Workshop on Entity Recognition &#38; Disambiguation*, ERD '14, pages 25–30, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3023-7.

[12] Jeffrey Dalton and Laura Dietz. A neighborhood relevance model for entity linking. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval*, OAIR '13, pages 149–156, Paris, France, France, 2013. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE. ISBN 978-2-905450-09-8.

[13] Jeffrey Dalton, Laura Dietz, and James Allan. Entity query feature expansion using knowledge base links. In *Proceedings of the 37th International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, SIGIR '14, pages 365–374, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2257-7.

[14] Gianluca Demartini, Tereza Iofciu, and Arjen P. Vries. *Focused Retrieval and Evaluation: 8th International Workshop of the Initiative for*

the Evaluation of XML Retrieval, INEX 2009, Brisbane, Australia, December 7-9, 2009, Revised and Selected Papers, chapter Overview of the INEX 2009 Entity Ranking Track, pages 254–264. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-14556-8.

[15] Michael Ringgaard Evgeniy Gabrilovich and Amarnag Subramanya. Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0). http://lemurproject.org/clueweb12/FACC1/, 2013. [Online; accessed 17-February-2016].

[16] Paolo Ferragina and Ugo Scaiella. Tagme: On-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 1625–1628, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0099-5.

[17] Python Software Foundation. 7.8. codecs — codec registry and base classes. https://docs.python.org/2/library/codecs.html, 2016. [Online; accessed 17-February-2016].

[18] Python Software Foundation. Globalinterpreterlock - python wiki. https://wiki.python.org/moin/GlobalInterpreterLock, 2016. [Online; accessed 17-February-2016].

[19] Google. Freebase data dumps. https://developers.google.com/freebase/data, 2016.

[20] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. A greedy algorithm for finding sets of entity linking interpretations in queries. In *Proceedings of the First International Workshop on Entity Recognition &#38; Disambiguation*, ERD '14, pages 75–78, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3023-7.

[21] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. Entity linking in queries: Tasks and evaluation. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ICTIR '15, pages 171–180, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3833-2.

[22] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, chapter On the Reproducibility of the TAGME Entity Linking System, pages 436–449. Springer International Publishing, Cham, 2016. ISBN 978-3-319-30671-1.

[23] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X.

[24] Thomas Lin, Patrick Pantel, Michael Gamon, Anitha Kannan, and Ariel Fuxman. Active objects: Actions for entity-centric search. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 589–598, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1229-5.

[25] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.

[26] Edgar Meij, Wouter Weerkamp, and Maarten de Rijke. Adding semantics to microblog posts. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, WSDM '12, pages 563–572, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0747-5.

[27] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 472–479, New York, NY, USA, 2005. ACM. ISBN 1-59593-034-5.

[28] Rada Mihalcea and Andras Csomai. Wikify!: Linking documents to encyclopedic knowledge. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 233–242, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-803-9.

[29] David Milne and Ian H. Witten. Learning to link with wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 509–518, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3.

[30] Gregory B. Newby, Christopher T. Fallen, and Kylie McCormick. Lucene for n-grams using the clueweb collection. In Ellen M. Voorhees and Lori P. Buckland, editors, *TREC*, volume Special Publication 500-278. National Institute of Standards and Technology (NIST), 2009.

[31] Paul Ogilvie and Jamie Callan. Combining document representations for known-item search. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 143–150, New York, NY, USA, 2003. ACM. ISBN 1-58113-646-3.

[32] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 275–281, New York, NY, USA, 1998. ACM. ISBN 1-58113-015-5.

[33] Jeffrey Pound, Peter Mika, and Hugo Zaragoza. Ad-hoc object retrieval in the web of data. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 771–780, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8.

[34] The Lemur Project. The clueweb12 dataset: Dataset details. http://lemurproject.org/clueweb12/specs.php, 2015. [Online; accessed 27-November-2015].

[35] Lawrence R. Rabiner. *Readings in Speech Recognition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-124-4.

[36] Michael Schuhmacher, Laura Dietz, and Simone Paolo Ponzetto. Ranking entities for web queries through text and knowledge. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1461–1470, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3794-6.

[37] Tomaz Solc. Unidecode 0.04.19 : Python package index. https://pypi.python.org/pypi/Unidecode/0.04.19, 2016. [Online; accessed 17-February-2016].

[38] Python Bug Tracker. Issue 8271: str.decode('utf8', 'replace') – conformance with unicode 5.2.0 - python tracker. http://bugs.python.org/issue8271, 2016. [Online; accessed 17-February-2016].

[39] Arjen P. Vries, Anne-Marie Vercoustre, James A. Thom, Nick Craswell, and Mounia Lalmas. Overview of the inex 2007 entity ranking track. In Norbert Fuhr, Jaap Kamps, Mounia Lalmas, and Andrew Trotman, editors, *Focused Access to XML Documents*, pages 245–251, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85901-7.

[40] W3C. The byte-order mark (bom) in html. https://www.w3.org/International/questions/qa-byte-order-mark.en.php, 2016. [Online; accessed 17-February-2016].

[41] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, April 2004. ISSN 1046-8188.

[42] Nikita Zhiltsov, Alexander Kotov, and Fedor Nikolaev. Fielded sequential dependence model for ad-hoc entity retrieval in the web of data. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 253–262, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3621-5.