FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Adaptierung von Word Embeddings für domänenspezifisches Information Retrieval

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Sebastian Hofstätter, BSc
Matrikelnummer 1225867

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Allan Hanbury
Mitwirkung: Dipl.-Ing. Navid Rekabsaz

Wien, 3. Mai 2018

Sebastian Hofstätter	Allan Hanbury

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Adapting Word Embeddings for Domain-Specific Information Retrieval

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Sebastian Hofstätter, BSc
Registration Number 1225867

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Allan Hanbury
Assistance: Dipl.-Ing. Navid Rekabsaz

Vienna, 3rd May, 2018

Sebastian Hofstätter | Allan Hanbury

Erklärung zur Verfassung der Arbeit

Sebastian Hofstätter, BSc
Mohsgasse 25, 1030 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Mai 2018

Sebastian Hofstätter

Acknowledgements

I would like to thank my advisors Navid Rekabsaz and Prof. Allan Hanbury for their continued feedback and expertise. Navid accelerated my interest in research and inspired me to do more and reach higher and I am very grateful for that. I would also like to thank Mihai Lupu for giving me advice during my work at the research group.

I like to thank my parents and my girlfriend Anna for listening to all my ideas that may or may not have made it into this thesis. Their support kept me focused.

Kurzfassung

Eine Suchmaschine sortiert Dokumente basierend auf ihrer Relevanz für die Suchanfrage. Wenn nur exakte Wortübereinstimmungen gefunden werden, können Ergebnisse übersehen werden. Das Erweitern einer Dokumentabfrage mit ähnlichen Wörtern aus einem Word-Embedding bietet ein großes Potenzial für bessere und umfangreichere Abfrageergebnisse. Die Erweiterung des Suchraums ermöglicht das Auffinden relevanter Dokumente, auch wenn diese nicht die eigentliche Anfrage enthalten. Ein zusätzliches Wort verbessert die Abfrageergebnisse nur, wenn es für das Thema der Suche relevant ist. Studien haben gezeigt, dass einige der hinzugefügten Wörter, die durch ein standard Word-Embedding gewonnen wurden, die Retrieval-Performance negativ beeinflussen.

Wir passen ein Word-Embedding so an, um die Effektivität der Suchergebnisse zu verbessern. Die Änderungen, die wir in den lokalen Nachbarschaften von Wörtern durchführen, passen das Word-Embedding an die Anforderungen einer Suchmaschine an. Wir konzentrieren uns auf sehr ähnliche lokale Nachbarschaften innerhalb des Word-Embeddings, da wir die daraus gewonnenen Informationen bei der Erweiterung einer Suchabfrage verwenden. Für die Anpassung integrieren wir externe Ressourcen in das Word-Embedding mit Retrofitting. Wir experimentieren mit verschiedenen externen Ressourcen: Latent Semantic Indexing, semantische Lexika (zB WordNet) und verschiedene Kombinationen von beiden.

Wir analysieren Veränderungen in den lokalen Wort-Nachbarschaften von Abfragebegriffen und globale Differenzen zwischen den ursprünglichen und den nachgerüsteten Vektoren. Wir evaluieren die Auswirkungen der geänderten Word-Embeddings auf domänenspezifische Retrieval-Tests, in denen wir verbesserte Ergebnisse für einige Test melden. Zusammenfassend zeigen wir, dass verschiedene Domainen am besten mit verschiedenen Modellen verstärkt werden können.

Abstract

Search engines rank documents based on their relevance to a given query – using only exact word matches might miss results. Expanding a document retrieval query with similar words gained from a word embedding offers great potential for better query results. The expansion of the search space allows to retrieve relevant documents, even if they do not contain the actual query. An additional word improves the query results only if it is relevant to the topic of the search. As observed by previous studies, an essential problem in using an out-of-box word embedding for document retrieval is that some of the added similar words have a negative impact on the retrieval performance.

We create word embedding based similarity models, which are used to expand query words in domain-specific Information Retrieval. For this we adapt an existing word embedding with additional information gained from different contexts – we incorporate them into a Skip-gram word embedding with Retrofitting. We experiment with different external resources: Latent Semantic Indexing, semantic lexicons. We also study various techniques to combine two different external resources.

We first analyze changes in the local neighborhoods of query terms and global differences between the original and retrofitted vector spaces. We then evaluate the effect of the changed word embeddings on domain-specific retrieval test collections. We report improved results on some test collections. In conclusion, we show that in two out of three test collections, incorporating external resources significantly improves the results over using an out-of-the-box word embedding.

Contents

CHAPTER 1

Introduction

Searching for information has its roots long before modern computer systems. Index catalog cards allowed librarians for centuries to find information based on basic search queries, such as the author, title, or subject of a publication. Computer systems enable a new way of search: fast full-text search. This means the user is able to search for every single word contained in a collection. Search systems can read and index every word in every Wikipedia article in a matter of minutes. Index data structures can be queried in seconds. With the abundance of available data comes the problem of aggregating the results in a useful way. A user does not benefit from millions of unordered documents, which contain the query. The ranking of those documents is a very important aspect of search systems.

The Information Retrieval (IR) discipline is concerned with finding and retrieving relevant documents based on the information need expressed as a query. This includes text search as well as music, image, and video retrieval. The document retrieval result contains multiple documents ordered based on their relevance to the query. The relevance between a document and a query is determined by a scoring function. Common scoring methods are based on statistics such as the frequency of query words in documents and the inverse document frequency of query words.

Keywords selected by humans have the benefit of distilling the document they describe in a humanly-judged relevant way. Additionally, librarians can easily search for synonyms and incorporate the context in their search, because they can use their knowledge and human intelligence to do this task. Computer systems have to extract the relevance of a document to a certain query automatically.

The Natural Language Processing (NLP) discipline studies in a broad sense the understanding of natural language by computers. One part of this field is to provide a computable representation of words, as the building blocks of language. A word is represented as a numerical vector, called word embedding. A word embedding embeds

the identity of a word in a low-dimensional vector space (commonly between 20 and 500 dimensions). All words in the used vocabulary are embedded in the same vector space. The position of the vectors in this word embedding space defines relationships between words. This is an abstract representation of the meaning of a word in the context of other words in the vocabulary.

The Skip-gram (part of word2vec) [MCCD13] word embedding creation method uses a short window-context around each word (two to five words before and after the word) to capture a dense vector. Deriving the meaning of words using the local window-context is inspired by J.R. Firth: "You shall know a word by the company it keeps" [Fir57], i.e. words in similar contexts tend to have similar vector representations.

1.1 Motivation and Problem Definition

In recent years, several works studied the use of word embeddings for IR and in particular document retrieval [GFAC16, MDC17, NMCC16]. One recent study using word embeddings in document retrieval is the Generalized Translation Model in the Probabilistic Relevance Framework [RLHZ16]. In this study, the authors use a word embedding as a source for word pair similarities and introduce a model which extends a query word by its neighbors in the vector space, inside various scoring functions. The neighbors are determined with a cosine similarity and selected with a threshold. While the model shows relative improvements in retrieval results, in another study Rekabsaz et al. [RLHZ17] point out the issues caused by using word embeddings for retrieval tasks. They show that it improves the query retrieval performance, but also leads to topic shifting and diminishes the potential gains. For example: "asthma" has other (non-related) diseases as very similar words in a Skip-gram embedding of a complete Wikipedia corpus – such as the word "diabetes" – because "asthma" is often in similar contexts as "diabetes" (both being diseases). When one searches for "asthma" one would only like to retrieve relevant documents that talk specifically about "asthma" or for example treatment techniques, diagnoses, or organizations associated with "asthma". Documents about other unrelated diseases ("diabetes") should not be retrieved.

Word embeddings are primarily developed with a focus on NLP and not IR [MCCD13, BGJM17, PSM14]. The test datasets commonly used to evaluate word embeddings are word pair analogy and similarity tasks [LGD15, MSC+13]. These benchmarks neglect many subtleties of word embeddings needed for Information Retrieval such as the set of highly related neighbors, that do not shift the topic of a word.

1.2 Contributions of the Work

We hypothesize that the representation of words, when used in Information Retrieval, should not only be based on their window-context (as used by the Skip-gram method). The vector representation should be based on a diverse set of inputs: including the window-context, as well as a document-context (created from word-occurrences in documents in a collection), and semantic lexicons (hand crafted and judged similarities created by humans). When word similarities – which are used for query expansion – are based not on one context, but a variety of contexts, the Information Retrieval results improve.

The contributions of this thesis are:

- **Exploitation and Analysis of Adapted Word Embedding Models** We create different word similarity models, which are based on a word embedding. We use a Skip-gram word embedding as a base and add additional information from other methods – we refer to them as external resources – with the Retrofitting method by Faruqui et al. [FDJ+15]. Retrofitting is a post-processing method, which moves the vectors in a word embedding, guided by word similarities from an external resource. We generate word-to-word similarities, by using these novel word embedding models.

  We set up multiple word similarity models: Retrofitting with one external resource, and Extended-Retrofitting (with two external resources), a model that averages the word similarities from two retrofitted word embeddings, and finally a Post-Filter model that filters the similarities gained from a word embedding by using a second external similarity source.

  Despite the fact that we use word embeddings created by the Skip-gram method, our models are independent of the original creation method. We use the following external resources: document-context similarities created from LSI [DDF+90] and semantic lexicons (WordNet [Mil95], FrameNet [BFL98], and PPDB [GVCB13]). We create models for different retrieval evaluation domains. The domain-specific models differ by the corpora used to create the Skip-gram embedding and the external resources used in the Retrofitting process. For each domain we choose a corpus that is best suited for the domain.

  To have a deeper analysis on the changes of the vectors in the retrofitted word embeddings, we compare local neighborhoods of words used in the retrieval evaluation with various metrics and analyze global differences between the original and retrofitted vector spaces. We found that the vectors in the retrofitted word embeddings move closer to each other, creating larger neighborhoods in the same similarity radius. We show that Retrofitting with different external resources creates similarity results that are indeed different from each other.

- **Information Retrieval Evaluation of Novel Word Similarity Models** The effectiveness of the word similarity models in Information Retrieval is evaluated

using an implementation of the Translation Models in the Probabilistic Relevance Framework [RLHZ16]. We evaluate our similarity models using our retrieval model implementation against multiple baselines with BM25 and LM scoring methods. The baselines include an unchanged query and using an out-of-the-box Skip-gram word embedding to augment a query. We use two news domain collections and a large patent collection to measure the quality of the retrieval results. Our implementation makes the retrieval models available in the popular open-source search engine Lucene as a plugin.

We present a comparison of cross-validated retrieval results between our novel word similarity models, as well as a thorough parameter exploration to find the best suited parameter configuration per test collection. We conclude that the three different test collections require different similarity models to achieve the best results. Overall the best similarity method is a Post-Filter and Retrofitting combination of two external resources which performs best on one of the news collections and the patent collection, additional it performs the same as the baseline in the second news collection. It shows that incorporating multiple resources gained from different methods and contexts improve the Information Retrieval results.

1.3 Structure of the Thesis

The structure of the thesis is as follows: in Chapter 2 we discuss the theoretical background of methods we used in this thesis. This includes an explanation of the Skip-gram model, Latent Semantic Indexing, the Retrofitting procedure, a brief introduction to inverted indexes, and the Generalized and Extended Translation Model in the Probabilistic Relevance Framework. In Chapter 3 we discuss related work in the context of word embeddings and Information Retrieval.

Our methodology is presented in Chapter 4, including the word similarity models, retrieval workflow and implementation details. Chapter 5 explains our experiment design with the data, metrics, and settings we used. The evaluation and results are detailed in Chapter 6 with a thorough analysis of the changes that happen in the word embeddings during the Retrofitting process, a discussion of the document retrieval results with parameter exploration, and cross-validated comparison of our models.

Chapter 7 concludes this thesis and in addition we present ideas for future work in Chapter 8.

CHAPTER 2

Background

This chapter describes the theoretical foundations of techniques used in this thesis. We describe the word embedding models we use in Section 2.1. Section 2.2 describes the Retrofitting process to merge multiple input resources into a single embedding. Section 2.3 describes the basic indexing data structures and scoring methods for Information Retrieval. We present the Generalized Translation Models in Section 2.4, which we use to augment a query with similar terms.

2.1 Word Embeddings

A word embedding contains a list of words as the vocabulary and it maps every word to a vector. Commonly, the vectors have between 50-500 dimensions for various tasks [MCCD13, RLH17b]. The word embedding encodes relationship between words. The semantic relation of the terms is given by their relationship in the vector space. A "relatedness" relationship is determined by a vector space distance, typically the cosine similarity. In the following we explain two methods to create word embeddings, which we use in this thesis, because both are established and well studied methods, that use different contexts to position the word vectors.

2.1.1 Latent Semantic Indexing

In this section we explain Latent Semantic Indexing (LSI) – a technique, which uses a document context to generate word-to-document relations. Deerwester et al. [DDF+90] introduced the LSI method to index and retrieve documents. Although LSI can be used for a variety of tasks in machine learning and text mining, we use it to create word-to-word similarity information.

We call it document-context, because the first step in the LSI model is to create a word-document co-occurrence matrix from a corpus. The co-occurrence counts are not

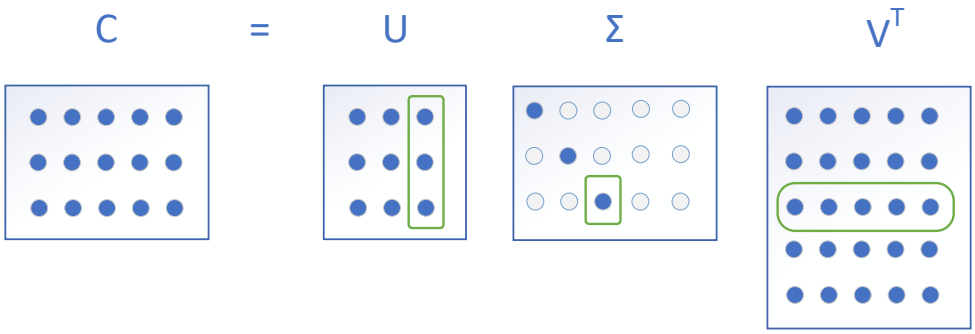

Figure 2.1: Dimensionality reduction of the 3 matrices created by SVD. In green values that are set to zero. [MRSO08]

used directly and the matrix is weighted with for example Term-Frequency Inverse-Document-Frequency (TF-IDF). Every row in the matrix corresponds to a term in the vocabulary and every column corresponds to a document. Typically, most words occur in few documents leaving the matrix very sparse. To create low-dimensional vector representations, in the next step, Singular Value Decomposition (SVD) is applied on the matrix. SVD decomposes the initial co-occurrence matrix C into three matrices. Multiplied with each other, the three decomposed matrices in fact equal the initial matrix C as shown in Equation 2.1.

$$C = U\Sigma V^\top \tag{2.1}$$

1. U is the term matrix. Every row corresponds to a term, with the same row as the term in the original matrix C. The number of columns (the vector length) equals the number of rows.

2. Σ is a diagonal matrix of singular values, e.g. only the diagonal values are non-zero. The singular values are decreasingly ordered.

3. $V^\top$ is a transposed document matrix. Every column corresponds to a single document – a document vector. The document vectors represent each document similar to the term vectors. The document vectors can be used to measure the similarity of documents.

The dimensionality of the matrices is reduced by only keeping the k largest singular values and their corresponding vectors in U and V. The other singular values are set to zero and therefore remove their corresponding vectors in U and V. This results in a rank k approximation of C with minimal error. This process and the layout of the matrices is illustrated with a single removed singular value in Figure 2.1, as illustrated by Manning et al. [MRSO08]. The removed singular value is set to zero and therefore

when multiplied the corresponding row and column in the matrices U and $V^\top$ become zero as well (highlighted in green).

Removing the zeroed-out columns and rows from the matrices leaves us with an approximation C' to the original matrix C and new matrices U' and $V'^\top$. Given the low-dimensional vector representation of words, one can compute the semantic similarity between two words using a cosine distance of their corresponding vector representations in the term matrix U'.

2.1.2 The Skip-gram Model

In Section 2.1.1 we presented the LSI method, which uses a document-context. In this section, we present a different approach, which uses a window around each word to extract information.

We focus on the popular Skip-gram architecture by Mikolov et al. (part of word2vec) [MCCD13, MSC+13], because we use it as a base word embedding for our similarity models in this thesis. We use it because, the method is well known and provides robust results across various tasks. Levy et al. [LGD15] show that different word embedding methods (Skip-gram, GloVe, PPMI) and their results strongly depend on the preprocessing of data and hyper-parameters of the word embedding models – with correct parameter tuning, the word embedding models show highly similar performances in word pair analogy and similarity benchmarks.

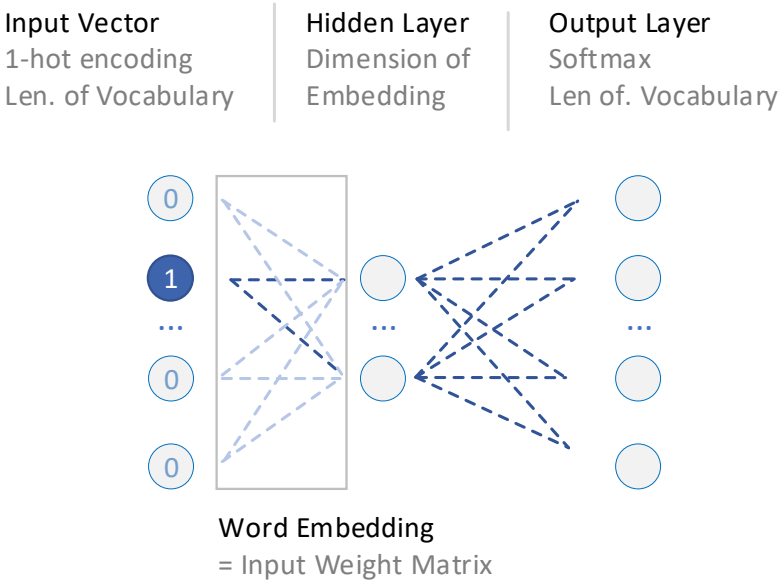

Figure 2.2: Skip-gram model neural network illustration

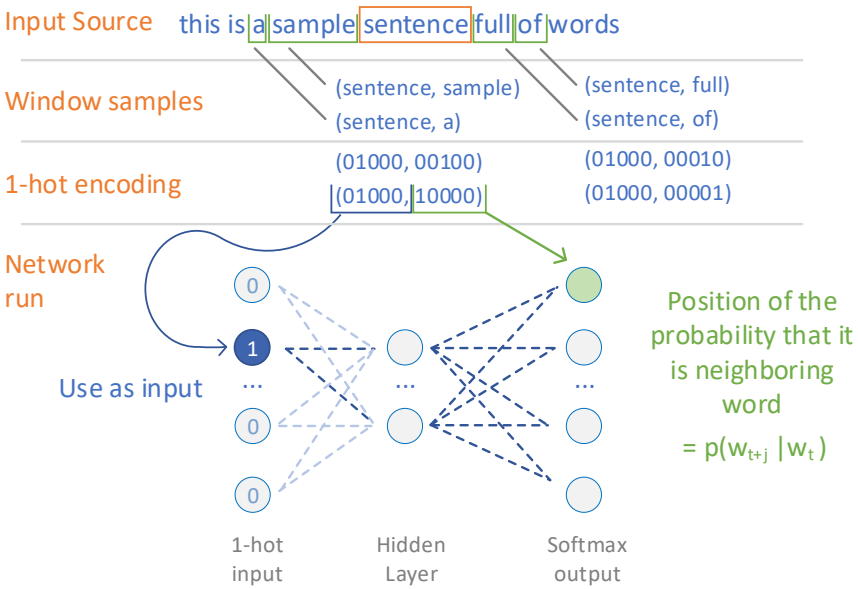

Figure 2.3: Skip-gram model neural network illustration

The Skip-gram model creates a word embedding by optimizing a different task: the embedding is a byproduct of the solution to optimize the task to predict the likelihood of the neighbors of a given word. Figure 2.2 shows the Skip-gram architecture. It uses a 1-hidden layer feed-forward neural network. The input and output layer have as many dimensions as words in the vocabulary. The hidden layer has as many dimensions as the dimensions of the word embedding vectors. The layer dimensions set the size of the weight matrices of the network. The word vectors are harvested from the network in form of the input layer weight matrix. The matrix contains a row for each word in the vocabulary.

Before training the model with a text corpus, the text is prepared: sentence and document boundary information is removed and the corpus text is split into a sequence of words. A vocabulary is generated from the resulting list of words, where very infrequent words are discarded and removed from the corpus. Commonly as a preprocessing step every character is lower-cased. Every word in the vocabulary is assigned an id (used as the index in the word embedding matrix), that does not change.

Figure 2.3 illustrates the process of a single training update in the model. The main word and its context words are transformed into a list of tuples each containing the main word and one neighbor. At this point, neighborhood distance information, i.e. how far away a word is in the window, is discarded. The characters are transformed into their 1-hot encoding representation based on their position in the vocabulary. The 1-hot encoding sets one position to one and leaves the rest of the list with zeros. The 1-hot representation of the main word is used as input vector for the neural network. This

selects the matrix row for the main word of the matrix between the input and the hidden layer. Together with the output weight matrix and the softmax function a probability for every word in the vocabulary is computed. In practice not every probability is calculated. The weights are updated by using Stochastic Gradient Descent (SGD) to minimize the loss.

The training objective of the Skip-gram model given by Mikolov et al. is to maximize the average log probability of a context of size c around every word in the training sequence $w_1, w_2, ..., w_T$, as shown in the following:

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j}|w_t) \tag{2.2}$$

The variable T equals the number of words that are used for training. The probability $p(w_{t+j}|w_t)$ is defined as a softmax function:

$$p(w_{t+j}|w_t) = \frac{\exp(v_{w_t}^\top v'_{w_{t+j}})}{\sum_{w_i \in V} \exp(v_{w_t}^\top v'_{w_i})} \tag{2.3}$$

The function transforms a vector product into a probability in the range of $[0, 1]$ and the probability distribution sums up to 1. The variable v denotes an input vector and v' denotes an output vector (the corresponding vector in the output layer). The sum in the denominator is computed over all words in the vocabulary. Using this equation is not feasible - it would take too long to compute. Therefore, Mikolov et al. compute only an approximation of Equation 2.3. Many techniques exist to approximate the softmax function, improving its efficiency. Mikolov et al. introduce Negative Sampling, instead of computing the softmax over all words in the vocabulary, most of which are not probable to appear next to the current word, the objective is changed to:

$$\begin{aligned} J_\theta &= \log \sigma({v'_{w_{t+j}}}^\top v_{w_t}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-{v'_{w_i}}^\top v_{w_t})] \\ \sigma(x) &= \frac{1}{1 + \exp(-x)} \end{aligned} \tag{2.4}$$

The objective of the Skip-gram model in Equation 2.4 has two parts: the first contains the neighboring word of the main term (which is given as training sample) and the second is the sum of a list of k negative samples. Between 5 and 20 negative words, i.e. words that do not interact with the current main term, are randomly selected from the vocabulary. This is in fact one of the core contributions of the method: being able to train the model, without having to update all weights for every iteration. Only the weights of the used samples are updated. This enables the use of huge corpora with billions of training samples [MCCD13].

In addition to the Negative Sampling method, Mikolov et al. propose to subsample very frequent words. Some occurrences are ignored, i.e. not included in the window, based on the probability shown in the following:

$$P(w_i) = 1 - \sqrt{\frac{10^{-5}}{f(w_i)}} \tag{2.5}$$

This increases the training efficiency, but also improves the vectors of rare words, because the subsampling decreases the imbalance of frequent and rare words. The frequency of a word in the training corpus is denoted by $f(w_i)$. When the training iterates over a corpus multiple times the same word might be used in one iteration and not in the other.

2.2 Word Embedding Retrofitting

We use the Retrofitting procedure to combine a word embedding created by the Skip-gram model (Section 2.1.2) with external information, such as word-to-word similarities gained from Latent Semantic Indexing (Section 2.1.1). Faruqui et al. [FDJ+15] propose the Retrofitting method to adapt the vector representations of an existing word embedding based on external resources. The Retrofitting procedure is applied to any word embedding as a post-processing step and does not require the corpus used to create the embedding. The external resource can be created from an arbitrary source as long as it provides word similarity information.

Figure 2.4 shows an illustration of a single update to a word vector in one iteration of the Retrofitting. The orange lines represent different similarity values from external resources (thicker means more similar). In this example two vectors are not in the neighborhood of the main term after the retrofitting, because they did not have a connection in the external resource. The original position of the main term, as well as every term selected by the external resource influence the new position of the main term after the iteration. Note that, only the main term moves in this example, but the other terms become main terms eventually and will move, based on their similar terms, as well during the Retrofitting.

The Retrofitting method changes the vector representations by optimizing the following objective function $\Psi(V)$:

$$\Psi(V) = \sum_{i=1}^{n} \left[\alpha_i \left\| v_i - \widehat{v}_i \right\|^2 + \sum_{(i,j) \in R} \beta_{ij} \left\| v_i - v_j \right\|^2 \right] \tag{2.6}$$

where V is the original embedding, variable $\widehat{v}$ denotes an original vector and v denotes a retrofitted vector representation, n the number of terms and $(i, j) \in R$ represents the related terms for the current vector at index i (provided by the external resource). The indexed parameter β_{ij} represents a weight between the current term and a related

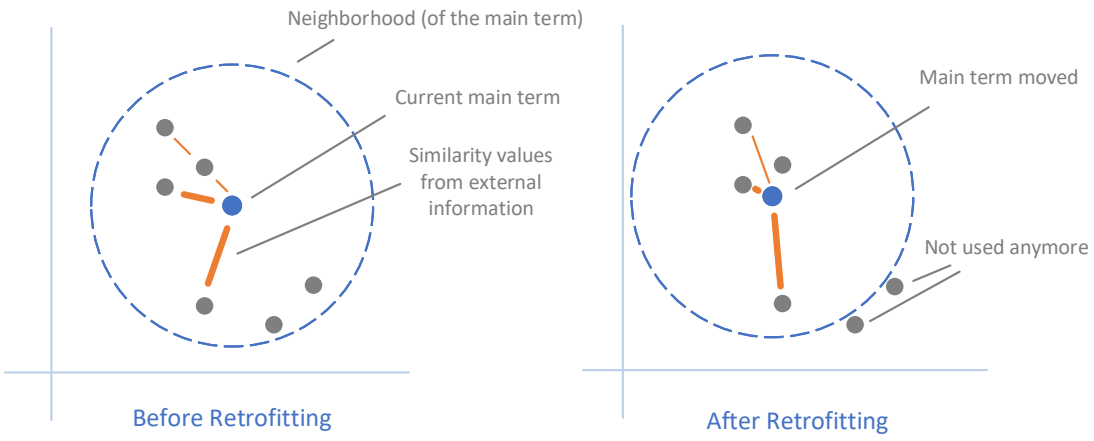

Figure 2.4: Two dimensional single iteration, single term Retrofitting illustration

term (which is provided by the external resource), while α_i sets a weight on the original position of the current term. Both regulate the influence of the original position of the current term versus the influence of a related term. Tipping the balance by increasing α_i strengthens the original position and reduces the changes and increasing β_{ij} yields a more changed embedding. Although the parameters are indexed in the model, Faruqui et al. set the parameters to constants: α_i is set to 1 and β_{ij} is set $\frac{1}{|(i,j)\in R|}$ so that the sum of all β_{ij} for a term is equal to 1 and the importance of the position in the original and the influence of the related terms is balanced.

In order to minimize the objective function, the method uses an iterative update for each individual vector, which is based on the first derivative of Equation 2.6 to find the optimal value for v_i:

$$\frac{\partial \Psi(V)}{\partial v_i} = 0 \tag{2.7}$$

Solving Equation 2.7, we result on the update that is applied to v_i:

$$v_i = \frac{\sum_{(i,j)\in R} \beta_{ij} v_j + \alpha_i \widehat{v}_i}{\sum_{(i,j)\in R} \beta_{ij} + \alpha_i} \tag{2.8}$$

This is the formula used in the implementation. The formula moves the term vector at the index i to a new position in the vector space, based on the vectors of the related terms in the external resource and the original vector of the main term. If there are no related terms in the external resource, the vector does not change. This is applied in an iterative fashion and after a few iterations the vectors converge to a minimum.

2.3 Document Retrieval

Information Retrieval and its subfield document retrieval contain many techniques and applications. We focus on the basic data structure to efficiently search text in documents and the scoring methods to rank matched documents.

2.3.1 Inverted Index

In this section, we give an overview of how the inverted index is structured and how it relates to a query in Information Retrieval. The fundamental question during a search is to assess the relevance between a given query and a document. This is formalized as a function with the query and document information as parameters. The function returns a numerical value – a score. A higher score means a higher relevance. This allows us to sort the documents and return the most relevant first.

When the user wants to search a collection of documents, one approach is to iterate over all document texts and rank each individual document based on the query. This is not a feasible solution, as the number of documents and their contents is very large and the user expects an answer in a very short time. Using an inverted index allows us to consider only documents that contain parts of the query and use precomputed statistics instead of the document text. The statistics are used by the scoring methods. The inverted index is created once and subsequently used by all queries for fast results.

As described by Manning et al. [MRSO08] the inverted index contains statistics per document and per term. Figure 2.5 gives an overview of the basic contents. Internally, the index assigns every document an id, which is commonly a consecutive number. All metadata associated with a document is stored, as well as the document length. The document length equals the number of terms in a document.

Every term points to a so called posting list. The posting list is a simple data structure that contains a list of tuples. A single tuple consists of a document id and the term frequency, i.e. the number of times the term appears in the document. Depending on the capabilities of the search system the implementation differs for the term matching, e.g. how you access a posting list from a given query term. In simple exact matching a hash based set is sufficient. The common property of different term dictionaries is that they must provide a fast access to information associated with a term.

Not every word that appears in a document is added to the index in the form in which it is written in a document. Every document that is indexed moves through a language specific pipeline of preprocessing steps:

1. The text is split by whitespaces and other control characters into a list of tokens
2. Commonly, but not necessarily, every token is stemmed, which means that the word is reduced to its stem, e.g. a word in singular and plural form map to the same stemmed word and when the user searches for a word it does not matter which form the word has in the query or document.

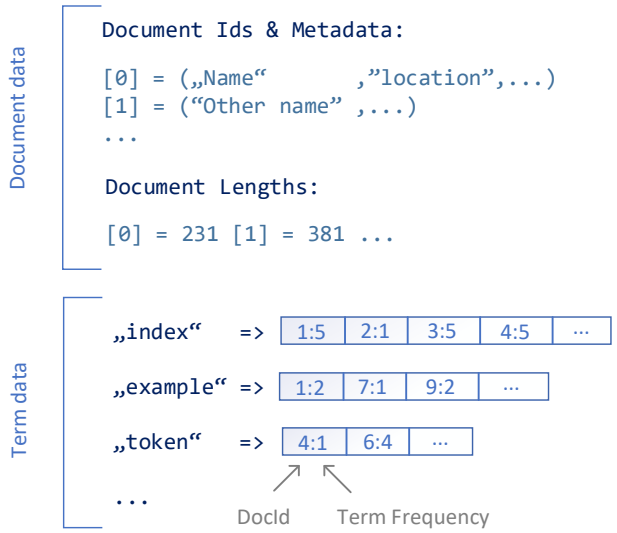

Figure 2.5: Basic inverted index contents and structure

3. Optionally, words that do not transport relevance are not added to the index. Those stop-words commonly include "the", "and", etc..

4. Finally the term is added to the index and in the terms posting list an entry with the document id is created or, if it already exists, the frequency is increased.

Full featured search engines like the open source Lucene[1] project have index implementations that offer many more features. For example: a positional index, where the term offsets are stored in addition to the raw frequency count. This allows to formulate queries that are aware of ranges between terms. In this thesis we only use a basic index described above.

2.3.2 Document Ranking

To assert the relevance of a certain document to a query, based on information saved in the inverted index (Section 2.3.1) we use a scoring formula. The popular method presented here is BM25, because we use it in our retrieval evaluation. BM25 is a widely studied scoring function, defined as follows:

[1] http://lucene.apache.org/

$$BM25(q,d) = \sum_{t \in T_d \cap T_q} \frac{(k_1+1)tf_{norm}(t)}{k_1 + tf_{norm}(t)} \frac{(k_3+1)tf_q(t)}{k_3 + tf_q(t)} \log \frac{|D| + 0.5}{df_t + 0.5} \tag{2.9}$$

$$tf_{norm}(t) = \frac{tf_d(t)}{(1-b) + b\frac{L_d}{avgdl}}$$

The formula consist of three parts, indicated by the colored blocks:

1. *Orange* The term frequency block, which uses the term frequency of the current term in the given document $tf_d(t)$ and normalizes it by the document length L_d as well as the average document length $avgdl$. It uses two hyper-parameters k_1 and b.
2. *Green* This block normalizes the term frequency in the query itself $tf_q(t)$. It uses one hyper-parameter k_3.
3. *Blue* The Inverse Document Frequency (IDF) block. It uses the number of documents $|D|$ and the document frequency of the current term, i.e. in how many documents the term appears. To prevent a division by zero, the division is adjusted with 0.5. It gives more weight to terms that appear in fewer documents and less weight to very frequent terms.

The hyper-parameters are chosen with a parameter exploration, that evaluates multiple candidates and compares the retrieval results to a gold-standard list. The hyper-parameters for BM25 have commonly used default values [RLHZ16], which we also use in this thesis.

2.4 Generalized Translation Models

In previous sections we focused on word embeddings. We now use their term relatedness information and incorporate it in Information Retrieval. The effective methods we use are called Generalized and Extended Translation Models. Rekabsaz et al. [RLHZ16] propose the Generalized and Extended Translation Models as a way to incorporate the idea of the translation model into the Probabilistic Relevance Framework.

Historically, translation models have been introduced as a way to extend language model ranking methods. The language model utilizes the probability between a query term and a document model $P(t_q|M_d)$ to rank documents. The translation model adds a translation probability between every query term and every document term. This extension is:

$$P(q|M_d) = \prod_{t_q \in q} \left(\sum_{t_d \in d} P_t(t_q|t_d) P(t_d|M_d) \right) \tag{2.10}$$

Therefore, in a ranking, more terms are considered for the result. The Generalized and Extended Translation models change the basic building blocks of Information Retrieval models. In the following we explain each of these.

For every term t, in the vocabulary of the embedding, a list $R(t)$ of similar terms is selected from a word similarity source, such as a word embedding. For every term t in a query q, the similar terms in the indexed documents are replaced by t and annotated with a weight (from a similarity source). Every instance of a similar term is updated, whether the original term is in the document or not. When we search for the term in an inverted index we return not only the original occurrences, but also the weighted replacements. This technique is formalized by:

$$\widehat{T}_d = T_d \setminus \bigcup_{t \in q} \{t' \in R(t)\} \cup \{t \in q : R(t) \cap T_d \neq \emptyset\} \tag{2.11}$$

The set of terms per document T_d is rewritten as $\widehat{T}_d$, $cos(t, t')$ denotes the similarity given by the cosine distance between t and t'. The extended term frequency per document d is the sum of the main term and all its weighted similar terms:

$$\widehat{tf}_d(t) = tf_d(t) + \sum_{t' \in R(t)} cos(t, t') tf_d(t') \tag{2.12}$$

The changed document frequency per term is given by:

$$\widehat{df}_t = |\{d \in D : t \in T_d \vee \exists t' \in R(t), t' \in T_d\}| \tag{2.13}$$

It counts all documents that either contain the main term or at least one of the similar terms. The weights of the similar terms are not used. Because the replaced terms are weighted it necessarily changes the length of a document. The redefined document length is:

$$\widehat{L}_d = \sum_{t \in \widehat{T}_d} \widehat{tf}_d(t) \tag{2.14}$$

Because the individual document lengths change, the average is changed as well. The following shows the extended average document length, where D denotes the list of documents:

$$\widehat{avgdl} = \frac{1}{|D|} \sum_{d \in D} \widehat{L}_d \tag{2.15}$$

Rekabsaz et al. [RLHZ16] propose two different variants for common scoring methods, which make use of the changed building blocks: The Generalized Translation Model (GT) and the Extended Translation Model (ET). In the following, we explain the extension of these two models to BM25 and Language Modeling.

The GT model uses only the updated index, which replaces the terms per document from Equation 2.11 and the updated term frequency from Equation 2.12. Rekabsaz et al. use a Dirichlet smoothing function to define the probabilities of the Language Model. The BM25-GT function is defined as follows:

$$BM25_{GT}(q,d) = \sum_{t \in \widehat{T}_d \cap T_q} \frac{(k_1+1)\widehat{tf}_{normGT}(t)}{k_1 + \widehat{tf}_{normGT}(t)} \frac{(k_3+1)tf_q(t)}{k_3 + tf_q(t)} \log \frac{|D|+0.5}{df_t+0.5} \tag{2.16}$$

$$\widehat{tf}_{normGT}(t) = \frac{\widehat{tf}_d(t)}{(1-b)+b\frac{L_d}{avgdl}}$$

The LM-GT function is defined as follows:

$$LM_{GT}(q,d) = \prod_{t_q \in q} \Big(\sum_{t_d \in d} \frac{L_d}{L_d+\mu} \widehat{tf}_d(t) + \frac{\mu}{L_d+\mu} \frac{tf_c(t)}{L_c} \Big) \tag{2.17}$$

The ET model makes use of all changed building blocks including the weighted document length, which changes the average document length and the updated document frequency. The BM25-ET function is defined as follows:

$$BM25_{ET}(q,d) = \sum_{t \in \widehat{T}_d \cap T_q} \frac{(k_1+1)\widehat{tf}_{normET}(t)}{k_1 + \widehat{tf}_{normET}(t)} \frac{(k_3+1)tf_q(t)}{k_3 + tf_q(t)} \log \frac{|D|+0.5}{\widehat{df}_t+0.5} \tag{2.18}$$

$$\widehat{tf}_{normET}(t) = \frac{\widehat{tf}_d(t)}{(1-b)+b\frac{\widehat{L}_d}{avgdl}}$$

The LM-ET function is defined as follows:

$$LM_{ET}(q,d) = \prod_{t_q \in q} \Big(\sum_{t_d \in d} \frac{\widehat{L_d}}{\widehat{L_d}+\mu} \widehat{tf}_d(t) + \frac{\mu}{\widehat{L_d}+\mu} \frac{\widehat{tf}_c(t)}{\widehat{L_c}} \Big) \tag{2.19}$$

We use the GT and ET models to retrieve documents in this thesis, because it does not restrict the source of the word similarities, that we want to change and it offers a precise method of expanding a query search space.

2.5 Summary

In this chapter we described the theoretical foundations of techniques used in this thesis. We present two different word embedding models, that use different contexts to generate word vectors. We describe the Retrofitting method to merge multiple input resources into a single embedding, which we use to create the core of our similarity models. The similarity models we create are used in Information Retrieval queries. As a foundation for IR we described the basic indexing data structures and scoring methods for IR. Finally, we presented the Generalized and Extended Translation Models to use our similarity models in IR to augment a query.

CHAPTER 3

Related Work

The scope of our work spans across multiple domains, therefore we split the related work chapter into two sections: Word Embedding and Information Retrieval focused related work.

3.1 Word Embedding

A variety of ways exist to produce word embeddings and researchers base new models on top of the Skip-gram and CBOW models from Mikolov et al. [MCCD13]. Most new models are compared using common benchmarks. Table 3.1 shows common word embedding benchmark results for models we present in this related work section, as reported by the respective author. As described by Faruqui et al. [FDJ+15] the WordSim-353 and RG-65 benchmarks contain 353 and 65 hand-crafted gold-standard pairs of English words and the MEN-3K benchmark contains three thousand pairs of very frequent word pairs in

Table 3.1: Comparison of related work models using common word embedding benchmarks

Authors	**Model**	**MEN-3K**	**WordSim-353**	**RG-65**
Faruqui et al.	Skip-gram (unchanged)	67.8	65.6	72.8
Pennington et al.	Glove (6B, 300 dimensions)	-	65.8	77.8
Pennington et al.	Glove (42B, 300 dimensions)	-	75.9	82.9
Faruqui et al.	Retrofitting (Skip-gram + PPDB)	73.2	70.0	76.3
Faruqui et al.	Retrofitting (Skip-gram + WordNet)	70.3	67.5	77.8
Faruqui et al.	Retrofitting (Glove-6B + PPDB)	75.1	59.3	79.6
Faruqui et al.	Retrofitting (Glove-6B + WordNet)	75.9	61.2	84.2
Speer et al.	ConceptNet Numberbatch 16.09	**86.6**	**82.8**	**89.1**
Chen et al.	CBOW + Def.Lists.	74.8	70.6	-
Jauhar et al.	Skip-gram + EM-RETRO	42.8	32.1	73.4
Wang et al.	CBOW + PPDB + Freq	68.7	61.1	71.3

a large web corpora. The results of the benchmarks are reported as Spearman's rank correlation between the gold-standard list and the model result lists.

Pennington et al. [PSM14] present an embedding creation method: GloVe (Global Vectors). They combine matrix factorization (similar to LSI) and local co-occurrence windows (similar to word2vec). GloVe creates a co-occurrence matrix for all words in the vocabulary. The low-dimensional vectors are learned to approximate the log probability of a co-occurrence of two words in the full co-occurrence matrix. Additionally, they use very large corpora to train their model. As Table 3.1 shows, a larger training corpus leads to better benchmark results for the same model.

It is possible to take a pre-built embedding and change it in a post-processing step. We already described the Retrofitting method in Section 2.2. Now we present the experiments conducted by Faruqui et al. [FDJ+15]. In the Retrofitting process Faruqui et al. augment different embeddings (based on Skip-gram, GloVe and others) independently with the semantic lexicons: PPDB [GVCB13], WordNet [Mil95], and FrameNet [BFL98]. They transform the lexicons into a unified data structure, which contains a list of related terms per term, without individual weights. All related terms have the same relative influence in a Retrofitting step. They show that the retrofitted embeddings improve the results across common semantic, sentiment analysis, and rare word benchmarks (WordSim-353, RG-65, MEN, SYN-REL). The exact results depend on the combination of input embedding and used lexicon. In general PPDB and WordNet lead to an improvement of 5 to 10 percent in the benchmark scores, whereas FrameNet does not improve the results. Because we use the Retrofitting method in our model, we also used the same semantic lexicons as Faruqui et al. in our experiments.

Speer et al. [SCH17] show that by combining multiple input sources better results can be achieved in common word embedding benchmarks (MEN, Stanford Rare Words - RW, WordSim-353, RG-65). They combine pre-computed word2vec and GloVe embeddings and retrofit them with data from ConceptNet. ConceptNet is a combination of multiple knowledge graph resources including WordNet and DBPedia. Analogous to Faruqui et al. [FDJ+15] they focus on common word embedding benchmarks and not information retrieval tasks in their evaluation. The results reported by Speer et al. are the best results for all three common benchmarks we surveyed across the related work models (see Table 3.1).

Chen et al. [CD15] present an alternative to Retrofitting, by incorporating two different contexts (the text corpus and word pairs of related words) into a single word embedding. They adapt the word2vec objective to train a joint model. They do not use post-processing, such as Retrofitting. Similar to Faruqui et al. [FDJ+15] they establish better results in the WordSim-353 and MEN benchmarks over a plain word2vec baseline.

Jauhar et al. [JDH15] adapt the Retrofitting method to allow for multi-sense word vectors that capture polysemy. They use a Skip-gram embedding and WordNet within their Retrofitting procedure. With this Jauhar et al. separate out the different meanings of vectors that are in captured in WordNet. As Table 3.1 shows, in comparison to the

other related work models the results presented by Jahuar et al. do not perform as well as the others models.

Wang et al. [WM17] extend word2vec to incorporate PPDB or WordNet semantic information into a word embedding, while treating instances of words differently, based on "information content". The information content is provided by a scoring function, which uses collection statistics to weight individual terms during the embedding learning. The rate of application of external resources is controlled based on this measurement. Their results fall short of most other models presented in this related work section (see Table 3.1).

3.2 Document Retrieval

In recent years, several studies addressed using word embeddings in the context of Information Retrieval and how to adapt embeddings or the retrieval method to improve the downstream retrieval results.

Mitra et al. [MNCC16] propose a Dual Embedding Space Model (DESM) document scoring method, that uses a word2vec word embedding trained on a query corpus. They focus on the "aboutness" of a document - does the document only contain a query term without being relevant, or is it about this query term. In the scoring function each document is defined as a normalized sum vector of all terms in that document. The model then takes each query term vector and computes the cosine distance to every document vector. Mitra et al. propose two variants of their model: the first uses only the input vectors of the word2vec model in the scoring process, whereas the second combines the input and output vectors of the word2vec neural network into a document aggregation vector. Their experiments show that the second approach captures the topical similarity better. They combine the model linearly with BM25 retrieval ranking. The combination shows significantly better results on an internal query log dataset.

Zamani et al. [ZC17] argue that for information retrieval the relevance information is more important than the semantic or syntactic similarity, captured by window-context word embedding methods. They propose a relevance-based query embedding based on a neural network model, which encodes relevance in its objective function. They learn a new embedding from scratch, while we use post-processing. Zamani et al. [ZC17] show that their word embedding model outperform word2vec and GloVe in information retrieval tasks.

Diaz et al. [DMC16] create their own local-context embedding that performs a topic-specific training and expands queries with it. They adapt the word2vec training objective to include document information based on sample query results. Their model uses a Kullback-Leibler divergence to score a document and a query, where the queries are taken from a test collection. Because they train their model with queries they use a 10-fold cross validation to mitigate overfitting their model with test data. They show that their local embedding outperforms the word2vec baseline significantly.

Grbovic et al. [GDR+15] use word embeddings created from search logs to better match advertising keywords to a search query. Based on the Skip-gram model they combine two data sources: the queries and sessions, i.e. a list of queries that a user searches for close after one another. Instead of traditional stop-words they use a list of navigational queries as stop words and subsample them according to Mikolov et al.'s [MSC+13] subsampling formula. The list contains 3000 navigational queries. Using a word2vec embedding trained with general news text yields poor results, but training word2vec with their search data improves the results significantly.

Nguyen et al. [NTSS17] work on domain-specific document embeddings in medicine. They use their embedding to perform document to document matching in medical search tasks. They extract concepts of documents using a knowledge-based medical resource. They conduct the retrieval with the contents of one document as the query. The retrieval process uses nearby documents gained from the vector space to enrich the query. Nguyen et al. show improved results for health-related search and clinical search for cohorts on the precision, recall, and MAP measures.

Guo et al. [GFAC16] introduce a novel neural ranking model (DRMM). In the search process they use a neural network based scoring method instead of a traditional scoring method, like BM25. The traditional method is used to create a candidate list of 2,000 documents, which is then re-ranked by the neural model. They use a word embedding to create a query word to document word similarity histogram. The buckets of the histograms are similarity ranges set as a hyper-parameter. The histogram is used to create a consistent input for the ranking neural network model, because it needs a fixed length - given by the input dimensions of the networks. The network only operates on word statistics from the histogram and does not use the word vectors in the neural network itself.

3.3 Summary

Multiple studies presented before improve word embeddings for NLP use cases. We can compare them with each other, because they often use common benchmarks and build on the same word embedding models (Skip-gram and Glove). It is more difficult to compare them with our work, because we focus on word similarities that are exclusively used in an Information Retrieval downstream task. We evaluate on a different set of objectives.

The Information Retrieval focused related works either change the retrieval model itself or use query information from logs or test collections to create a novel word embedding model. We do not create a new retrieval model, rather we create word embedding based similarity models which are used by the GT and ET retrieval models (presented in Section 2.4). We also do not use query information to change our embeddings. We use collection or knowledge based resources, which are not directly linked to searches.

CHAPTER 4

Methodology

This chapter provides an overview of the word similarity models used, a description of the retrieval workflow, an example how word neighborhoods change, and how we implemented our models.

4.1 Word Similarity Models

We do not study various retrieval models, our focus is on the source of the similar terms used. We use different models to provide word similarity information. This similarity information is subsequently used by the BM25-GT, BM25-ET, LM-GT, and LM-ET Information Retrieval models to conduct the retrieval. The requirements for the similarity, when used in Information Retrieval is that the additionally added words should be highly similar and topically related to the query words. Our hypothesis is that incorporating external resources in an out-of-the-box Skip-gram embedding tailors the list of similar words to the need of the retrieval tasks.

The external resources are encoded in the following data structure: The structure contains a vocabulary of available words and for every word in the vocabulary the resource contains a list of weighted similar words. If no score is available to weight a similar term, then the weight is set to a constant one. This allows to use the same data structure and models regardless of the availability of scores in the external resource. The external resource and the Skip-gram embedding might have different vocabularies, which means that only the intersection of both vocabularies contain the effective terms usable in the following word similarity models.

The output of every similarity model is a list of weighted similar words for every word in the vocabulary. We integrate the output of our models with the Translation Models by setting a threshold on the similarity score, so that the output of our models can directly

Table 4.1: Word similarity models

Model name	Description
`w2v`	Similarity source is an out-of-the-box Skip-gram embedding
`Post-Filter(*)`	Using the similarity results from a Skip-gram embedding and filter them again with the similarity information from an external resource
`Retro(*)`	Retrofit a Skip-gram embedding with word similarities gained from an external resource – the retrofitted embedding is used to provide word similarities
`Ext-Retro(* + *)`	Extended Retrofitting: Retrofit with two different external resources, both incorporating half of the influence
`Post-Filter-Retro(*,*)`	Use a Skip-gram embedding retrofitted with one external resource and post-filter the similarity results with another resource
`Retro(*) + Retro(*)`	Average the similarity results of two retrofitted embeddings, each with a separate external resource

be used by the GT & ET retrieval models. We refer to this threshold as Translation Model threshold.

Table 4.1 shows an overview of our models that we use to produce word similarity information. A star `(*)` describes a placeholder for an external resource. `Retro` stands for Retrofitting and indicates that the embedding has been retrofitted. The `Post-Filter` models are introduced by Rekabsaz et al. [RLHZ17].

`Retro(*)` The Retrofitting model is implemented using Equation 2.8. The equation provides a weight for a similarity relationship between two words (β_{ij}). We described in Section 2.2 how the sum of all β_{ij} for all similar terms must be one. An external resource might provide its own similarity score between two words. If so – to keep the "sum equals one" property – we normalize the score s_{ij} as shown in the following:

$$\beta_{ij} = \frac{s_{ij}}{\sum_{j'=1}^{m} s_{ij'}} \tag{4.1}$$

The score s_{ij} must be in the range of 0 to 1.

`Ext-Retro(* + *)` The Extended Retrofitting model combines two different external data resources (A and B) for the Retrofitting procedure. The method extends the

Retrofitting formula to allow a balanced influence of both resources. This is shown in the following equation:

$$v_i = \frac{\sum_{(i,j)\in R_A} \beta_{ij} v_j \cdot 0.5 + \sum_{(i,j)\in R_B} \gamma_{ij} v_j \cdot 0.5 + \alpha_i \widehat{v}_i}{\sum_{(i,j)\in R_A} \beta_{ij} \cdot 0.5 + \sum_{(i,j)\in R_B} \gamma_{ij} \cdot 0.5 + \alpha_i} \tag{4.2}$$

We duplicated the external term selection part and introduced a new parameter γ_{ij} that has the same functionality as β_{ij} but for the second resource. The two parameters are weighted down to only have half the influence on the new position of the retrofitted vector. This keeps the balance between α_i and the sums of γ_{ij} and β_{ij}, described in Section 2.2.

`Post-Filter(*)` The post filtering model can be used with any word embedding. The post filtering provider implements an additional step in the selection process of similar words gained from a word embedding. First, every similar word is extracted from a word embedding with the Translation Model threshold parameter. Then, every word to similar-word pair is filtered by the similarity of the word pair in the external resource `(*)`, e.g. if the similarity between the two words is smaller than a given threshold hyper-parameter than the similar word is not added to the output of the model. The post filtering does not add any additional words that have not been selected from the word embedding, it only removes words.

`Post-Filter-Retro(*,*)` This model is a nested combination model of two other models presented here. First, this model uses the `Retro(*)` model on the first given resource to create a retrofitted word embedding. It then employs the post-filtering using the `Post-Filter(*)` model with the second resource. This allows to use two external resources, while only changing the embedding with one resource.

`Retro(*) + Retro(*)` Another method to utilize two resources is to combine the similar word lists results of two retrofitted word embeddings. We retrofit the two word embeddings separately. The similarity lists are averaged, so that we have a balanced combination of the two. If the similarity of a word is low in one of the two word embeddings, then it will very likely be filtered out by the Translation Model threshold, which is applied after the combination. We added this model, because we observed that the various retrofitted word embeddings have different strengths and we want to combine them.

4.2 Retrieval Workflow

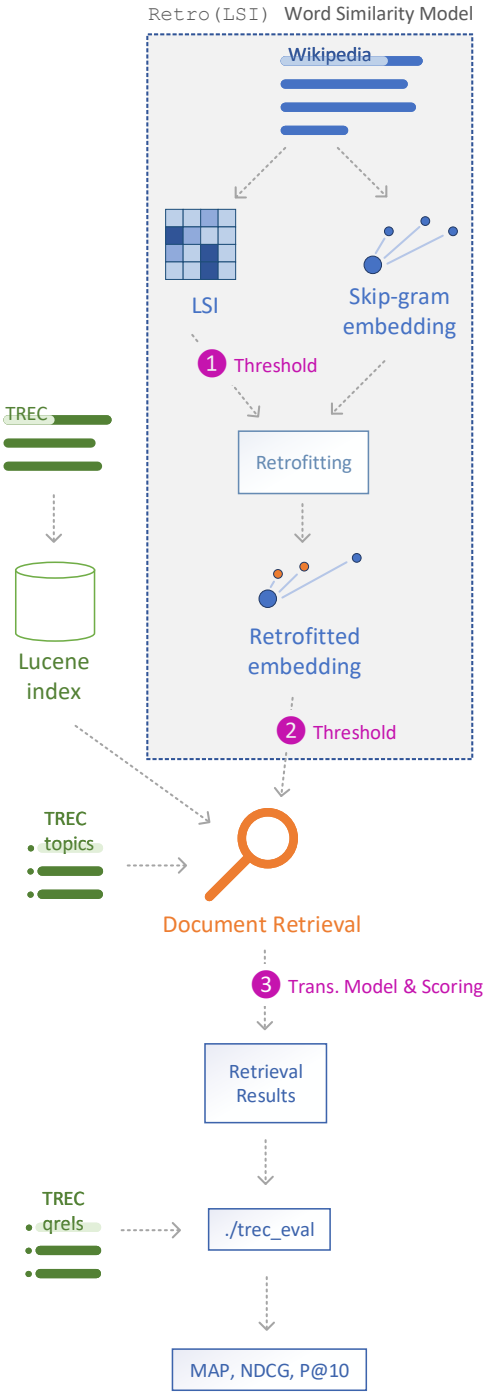

Figure 4.1: Workflow illustration

Figure 4.1 illustrates the retrieval workflow and how the word similarity models fit into it. As an example we choose the `Retro(LSI)` model.

The pipeline starts by creating the necessary prerequisites for the document retrieval task: an inverted index and a word similarity model, based on a retrofitted embedding. The inverted index (Section 2.3.1) is created from the respective test collection corpus in Lucene. The word embedding is the product of the Retrofitting process (Section 2.2), which combines a Skip-gram embedding (Section 2.1.2) with LSI term similarity information (Section 2.1.1). The word embeddings, the LSI data, and the index all use the same stemming option.

The document retrieval task, implemented in Lucene, uses the test collection topics to query the index. It uses the Generalized and Extended Translation Models (Section 2.4) to expand the index statistics used in the scoring functions, based on the word similarity gained from the retrofitted word embedding.

The retrieval results, which contain the first 1,000 ranked document identifiers per topic are evaluated with trec_eval. The program uses the qrels relevance judgments to compute evaluation metrics. The result is calculated per topic and on average over all topics. The per topic result is used to compute the significance tests and cross-validation results.

A single model run consists of multiple executions of the pipeline, using different values at the three configuration points (shown in magenta circles). (1) is the LSI threshold, selecting similar words used in the Retrofitting. (2) is the Translation Model threshold, selecting similar words used in the document retrieval. (3) is the retrieval model, one out of: BM25-GT, BM25-ET, LM-GT, or LM-ET.

As stated before, Figure 4.1 shows only one model pipeline. The other word similarity model and data combinations – as listed in Table 5.3 – differ slightly:

- `Retro(Semantic Lexicons)` does not have the external resource threshold parameter, because the lexicons do not contain a similarity score.
- The `Ext-Retro` model uses PPDB and LSI in the Retrofitting process, as mentioned in Table 5.3. PPDB is a static resource and does not need a parameter configuration. LSI is selected by the same threshold as `Retro(LSI)`.
- The `Post-Filter-Retro` model uses the same pipeline, but the selection of similar words for the retrieval task contains another layer that filters the similar words gained from the embedding. This filter introduces a third threshold parameter that is evaluated alongside a given range.
- The `Retro(LSI)+Retro(PPDB)` model duplicates the first part of the pipeline that creates the retrofitted embedding and linearly combines the similar word results of the two.

4.3 Word Similarity Model Example

The following qualitative example of the results for the words *austria* and *austrian* is created from a Wikipedia based word embedding as well as LSI similarities created from the same corpus. This example shows the subtle differences in seemingly very related words, as well as the benefits of the Retrofitting process with document-context LSI similarity. Table 4.2 and 4.3 show the neighborhoods of the respective words.

The term *austria* is highly similar to *austrian* in LSI (using the threshold of 0.82 in this example), but *austrian* is also linked to *habsburg* and *cisleithanian* - both showing a historical bias in Wikipedia documents about *austria(n)*. *Cislethania* is connected to *Austrian* history as the Wikipedia article about *Cislethania* states: "Cisleithania was a common yet unofficial denotation of the northern and western part of Austria-Hungary, the Dual Monarchy created in the Compromise of 1867"[1]. Wikipedia contains many articles about this topic mentioning both words often in similar contexts. The Skip-gram similarities also show a historic connection, but with a different term: In the original word embedding *austria* is highly similar to *hungari* (stemmed) and it would be expanded if the standard 0.7 threshold is used. In most use cases this is a clear topic shifting and it should not occur. After the retrofitting *hungari* and *germani* are both far away from the optimal 0.8 - 0.85 threshold – the topic shifting was mitigated in this case. With the word *austrian* the topic shifting manifests itself in the LSI data, not in the Skip-gram similarities. However, the Retrofitting does not move the unwanted words (*habsburg*, *cislethania*) close enough, that they are selected with a Translation Model threshold of 0.8, which shows the best results for retrofitted embeddings.

[1] See: https://en.wikipedia.org/wiki/Cisleithania *checked: 15.3.2018*

Table 4.2: LSI (at threshold 0.820) & word embedding neighborhood for the term *austria*

LSI		**Original Skip-gram**		**Retrofitted**	
austrian	0.910	austrian	0.779	austrian	0.952
		hungari	**0.734**	vienna	0.794
		vienna	0.719	graz	0.744
		germani	**0.690**	wien	0.736
		graz	0.661	linz	0.729
		oesterreich	0.658	viennes	0.722

Table 4.3: LSI (at threshold 0.820) & word embedding neighborhood for the term *austrian*

LSI		**Original Skip-gram**		**Retrofitted**	
austria	0.910	austria	0.779	austria	0.952
habsburg	**0.844**	vienna	0.733	vienna	0.800
cisleithanian	**0.838**	austro	0.711	austro	0.780
		oesterreichisch	0.673	viennes	0.761
		vorarlberg	0.662	**habsburg**	**0.755**
		graz	0.660	graz	0.737
		oesterreich	0.659	**cisleithanian**	**0.736**
		viennes	0.657	vorarlberg	0.724
		german	0.646	wien	0.722
		klagenfurt	0.640	klagenfurt	0.721

It is important to note that the vectors for two close terms, like in this example, influence each others position in the Retrofitting process, because both (*austria* and *austrian*) contain the other word in their LSI selection. This means that the vectors pull and push each other in each iteration of the Retrofitting. A position change of *austria* in the first iteration, has an influence on the position of *austrian* in the second iteration.

4.4 Implementation

The experiment pipeline is orchestrated using bash scripts, because different parts are written in different programming languages. Lucene is Java based and therefore the plugins are written in Java as well. Trec_eval is a a tool written in C. The embedding creation, retrofitting procedure, embedding analysis, cross-validation, and result transformation in plots and tables are written in Python. The programs are individually compiled and then started in the ordering set by the pipeline.

For every run we create a unique folder and copy the configuration file (containing all input paths and parameter settings) as well as all source code into it, this allows for a very good reproducibility. We do not copy the input data, but the the raw input data never changes, meaning it is not necessary to copy it for every run. We save all temporary

results, analysis output, and retrieval results in the run folder. Because a single run contains up to thousands of different experiment results and multiple different analysis tables, we create a single Excel file, which combines all result and analysis data in one place. This proves very useful in organizing the results and discussion and research process. For example, with a single Excel file it is possible to send the complete information a single run produced to other researchers via email without creating confusion about which experiment and data is referenced. To compare multiple runs, and therefore multiple models with each other, we created a meta result transformer tool. The tool reads all results from the given runs and transforms them into tables and plots.

We used the popular open-source Gensim library[2] in Python 3 to create our window-context word embeddings. The library contains a highly optimized implementation of Skip-gram as well as methods to compute the nearest neighbors of a word in any arbitrary embedding.

We implemented the Retrofitting process in Python using NumPy[3]. We use the formula shown in Equation 2.8, so that the implementation allows arbitrary external resources. The data structure of an external resource contains for every term in its vocabulary a list of similar terms and scores. The scores might be constant set to one. This allows us to experiment with semantic lexicons, document-context information, as well as various combinations of them. Terms are identified in the system with integer ids and not strings. We use integer ids to increase the performance of the system, because we can directly access an array index with them and the computation uses less memory. The computation time depends on the size of the embedding as well as the number of terms in the external data. In comparison with training a new model from scratch, the Retrofitting model is very efficient. Following Faruqui et al. [FDJ+15] we used 10 iterations, but the vectors converged usually after 6 to 7 iterations.

As described earlier in Section 2.4, we use the GT and ET models as basis for our IR tasks. The models are implemented in Java 8 as a plugin for the open-source search engine Lucene[4]. This allows the implementation to be used in a practical setting because Lucene is the core for the Solr[5] search server as well as the Elasticsearch[6] platform. A natural focus is on the performance and well tested correctness of the implementation, so that the method is feasible to use in a real-world scenario in addition to setting a reliable foundation for the evaluation of the new fusion embedding. The implementation is usable with any word embedding and any Lucene index.

Figure 4.2 shows an overview of the implemented process. Updating the index for every search is not feasible in a short time, especially if the index is large. The documents in the index remain unchanged, the similar words are gathered per query word from the word similarity model and the Lucene inverted index is queried for all original and similar words.

[2]See: https://github.com/RaRe-Technologies/gensim *checked: 21.4.2018*
[3]See: http://www.numpy.org/ *checked: 21.4.2018*
[4]See: http://lucene.apache.org/ *checked: 3.3.2018*
[5]See: http://lucene.apache.org/solr/ *checked: 3.3.2018*
[6]See: https://www.elastic.co/products/elasticsearch *checked: 3.3.2018*

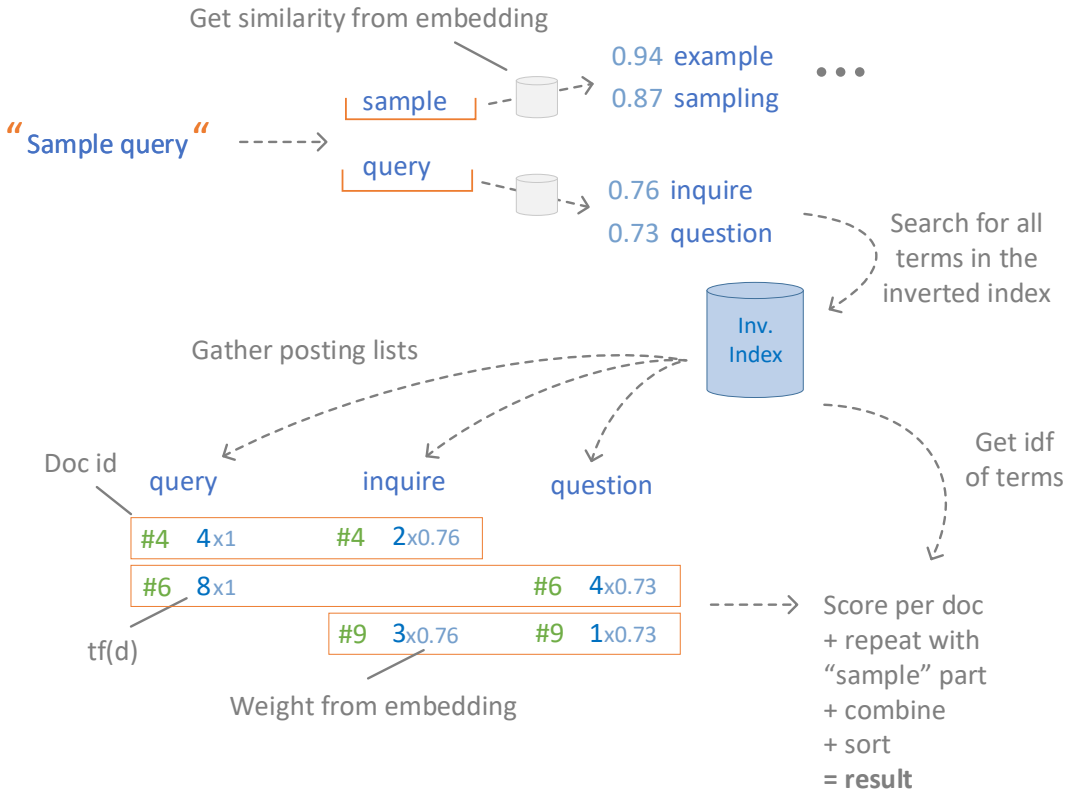

Figure 4.2: Query process illustration for an efficient Translation Model implementation

The returned posting lists are aligned, so that when iterating through them all posting lists stay at the same sorted document number and expose term frequency information per document. This alignment is necessary, because every document in Lucene is only scored once and as Equation 2.16 and 2.18 show the similar word information needs to be incorporated in a single execution of the BM25 formula. We iterate through every document and score it with the aligned term frequency information. This is repeated for every original query word and Lucene combines the scores per document, sorts them and returns the highest 1,000 documents results. The performance impact is linear in the number of added similar words and no further overhead is created. The implementation is further documented in detail in Appendix A.

The code and evaluation for the extension is available at:
https://github.com/sebastian-hofstaetter/ir-generalized-translation-models

CHAPTER 5

Experiment Design

In this chapter we present the test collections, external resources, parameter setting, evaluation metrics that we used in our experiments as well as how we calculated cross-validated results.

5.1 Test collections

Table 5.1 shows three Information Retrieval test collections, used for the evaluation. The two TREC collections are in the news domain and CLEF-IP is a patent text collection. We choose those collections to evaluate two distinctly different domains. The text from the TREC collections is stemmed using a Porter stemmer before creating the index, as it also improves retrieval performance. The text from CLEF-IP is not stemmed. All indexes use a stop-word list from NLTK. We use an English Wikipedia corpus as window-context and document-context for the TREC embeddings because the TREC collections are small and Wikipedia based embeddings yield better results, as shown by Rekabsaz et al. [RLHZ16]. The CLEF-IP collection is big enough for a good embedding and includes many patent domain specific words and meanings.

Table 5.1: A list of test collections and additional information we used

Name	**Collection**	**# Doc**	**Topics**	**Embedding**	**Stem**
TREC-123	Disc1&2	740088	150	Wikipedia	Yes
TREC-Robust-2004	Disc4&5 without CR	523951	250	Wikipedia	Yes
CLEF-IP-2013	CLEF-IP	2.6 million	50	CLEF-IP	No

Table 5.2: A summary of vocabulary size and effective words of our external resources (LSI is selected with a 0.75 threshold)

Name	# Vocab.	# Wiki	# CLEF-IP	# Avg. sim. words	Weight
LSI (Wikipedia)	482,294	220,449	-	3.22	Yes
LSI (CLEF-IP)	643,365	-	418,533	4.98	Yes
PPDB	102,903	38,533	47,422	3.64	No
WordNet(Synonyms)	148,731	24,306	23,819	2.12	No
FrameNet	10,823	6,253	6,614	39.98	No

5.2 External resources

We use two types of external resources: document-context and semantic lexicons. Both provide the same word similarity data structure described in Section 4.1. Table 5.2 gives an overview of all external resources used in our experiments. The "# Vocab." column shows the total amount of available words in each resource. However, not every word is usable, the "# Wiki" and "# CLEFIP" columns of Table 5.2 describe the number of words that intersect with the vocabulary of the Wikipedia and CLEF-IP word embedding vocabulary and therefore the effective number of words in the Retrofitting process. The "# Avg.sim. words" column shows how many similar words are effectively available per resource on average. In the LSI resources this number depends on the selected threshold. In this table we use an LSI threshold of 0.75. In the following we present more details about each external resource type:

- *Document-context* The document-context data is provided by LSI and selected with a similarity threshold. We use the LSI word matrix U', as described in Section 2.1.1 to create word-to-word based similarities for all words in the vocabulary. For every row in U', that represents a single word, we compute the cosine similarity with every other row in the matrix. We sort the list in descending order and select the words based on the given similarity threshold. We used the Gensim library for Python to compute our SVD matrices and NumPy to compute the list of similar words. The LSI vocabulary is the same as the vocabulary of the embeddings (because they are both created from the same corpora). However, for LSI the lower number of effective words is caused by the similarity threshold that filters out words that do not have at least one similar word with a higher score.

- *Semantic lexicons* We used the following semantic lexicons: WordNet [Mil95], FrameNet [BFL98], and PPDB [GVCB13]. We use them, because Faruqui et al. [FDJ+15] use them in their Retrofitting evaluation, as we have explained in Chapter 3. They also provide preprocessed data files of the lexicons, in the needed input data

structure for the Retrofitting process, as described in Section 4.1. The semantic lexicons contain hand-crafted similarity relationships between words, but word pairs are not scored. If a word is in a similarity relationship it has the implicit score of 1. As Table 5.2 shows, the semantic lexicons have smaller vocabularies than LSI and the effective word sets ("# Wiki" and "# CLEFIP") are even smaller, because the semantic lexicons are created independently from the corpora and therefore the vocabularies are less overlapping. Additionally, the semantic lexicons contain phrases, that are not supported by our system and therefore ignored and not in the effective word sets. FrameNet has a very high average number of similar words as it contains many duplicates.

Table 5.3 shows the combination of our similarity models and external resources to form our set of final similarity models. To keep the number of evaluated models manageable we only used PPDB, when we used multiple input combinations that include a semantic lexicon, because it showed the best results in the single resource Retrofitting evaluation.

Table 5.3: Combination of similarity models and external resources

Similarity model	**External resource data**
`Post-Filter(*)`	LSI word similarity
`Retro(*)`	Semantic lexicons (PPDB, WordNet, FrameNet) as used by Faruqui et al. [FDJ+15]; LSI word similarity
`Ext-Retro(* + *)`	PPDB and LSI word similarity
`Post-Filter-Retro(*,*)`	For retrofitting: PPDB; For post-filter: LSI word similarity
`Retro(*) + Retro(*)`	PPDB and LSI word similarity

5.3 Evaluation Metrics

Our evaluation focuses on MAP, P@10, and NDCG for the TREC collections and MAP, PRES, and RECALL for the CLEF-IP patent collection. The RECALL shows how many of all relevant documents have been retrieved. Finding a duplicated patent is an important goal in the patent domain. For the evaluation we used the trec_eval[1] utility. Trec_eval uses a list of human-judged document identifiers that are scored as relevant or non-relevant[2]. All other documents that are not judged are considered non-relevant.

[1]See: https://github.com/usnistgov/trec_eval *checked: 17.3.2018*
[2]See: https://trec.nist.gov/data/qrels_eng *checked: 17.3.2018*

5.4 Cross-Validation for Information Retrieval

We employ cross-validation of the experiment results to prevent overfitting the test collection data with our model. Overfitting occurs when parameters of the model are tuned specifically to improve the results of a set of queries. Because the parameters are tuned specifically for one collection they might not generalize well and the model performs worse on previously unseen queries. Cross-validation makes the evaluation and comparison of different models independent from the used parameters. Cross-validation is not used for parameter finding or tuning, rather to evaluate overall system performance in a fair setting, with no parameter bias. A significant difference between systems is a very strong signal of a better method.

We cross validate the Translation Model parameter configurations from a single experiment. This follows the experiment design concept by Rebasaz et al. [RLHZ16]. In Figure 4.1, which describes the workflow the Translation Model parameter is configuration point two. Once we cross-validated the Translation Model threshold, we pick the best LSI threshold if it is used. We do this because different LSI thresholds create different word embeddings.

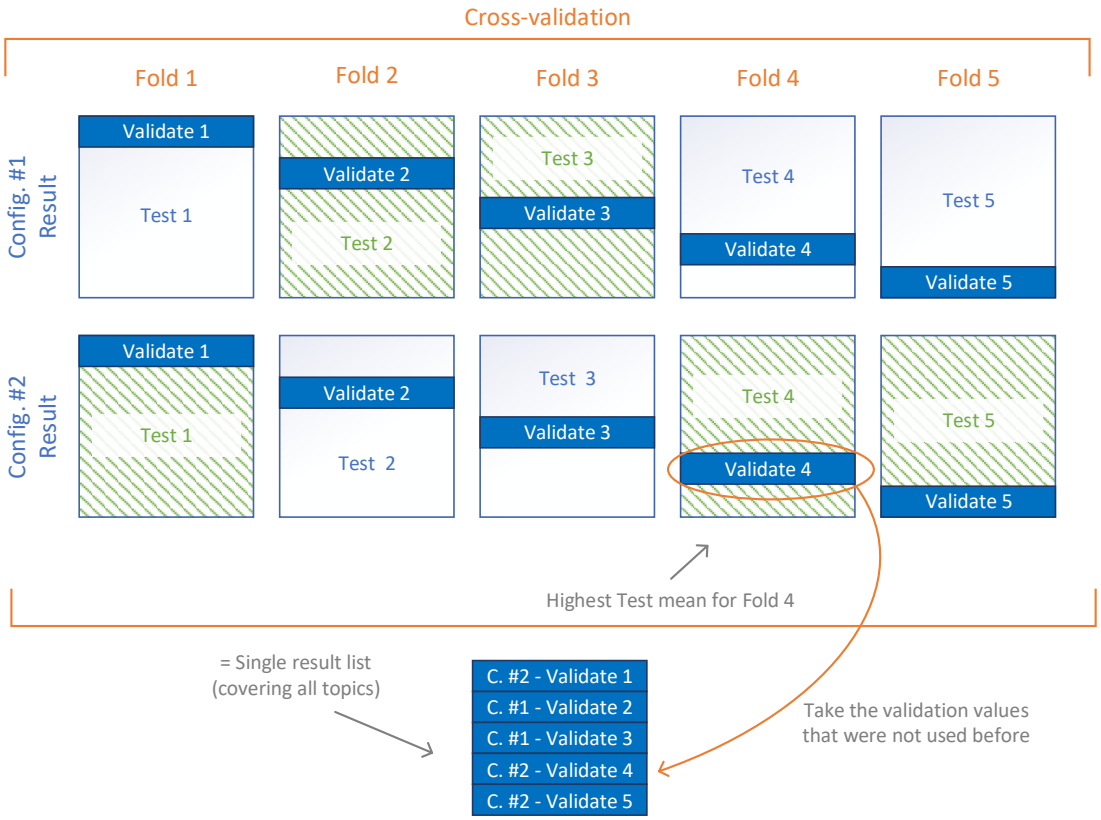

Figure 5.1: 5-fold cross-validation process for two sample experiment results

Subsequently we compare results of the cross-validation with other experiments that use the same measure, retrieval test collection, and scoring model.

Figure 5.1 shows how the cross-validation works with an example of two parameter values (Config. #1, Config. #2) used in one experiment. Because every topic is independently scored, we conduct the data splits and cross-validation after running the retrieval. The cross validation process works as follows:

1. The cross-validation begins by gathering all detailed results for every parameter configuration (based on our parameter range there are 40 different Translation model threshold configurations per experiment) - for example for 50 topics, each result is a list with 50 values based on a single evaluation metric, sorted by topic number.

2. For every result list cross-validation splits each list 5 times into 5 different folds: one validation and four test folds. Every topic (i.e. item in the list) is covered by one of the validation folds. For 50 topics, one validation fold contains 10 values and the test the other 40 values. The validation parts of one fold covers the same number of topics in all configuration results.

3. The mean for every test fold of every experiment is calculated. The highest mean value per fold is selected (one column in the figure) - which means we select one best experiment per fold. For this best experiment of a fold the validation result values are added to the result list (without checking if the validation results are better than the other experiments).

4. By selecting one validation part per fold, cross-validation creates a result list that contains exactly one result per topic – as mentioned above, every selected validation fold contains unique topic numbers.

5. With a single cross-validated result list we can calculate a single cross-validated mean result for the evaluated metric and use it to compare this experiment with others.

We report our cross-validated results in Chapter 6. Additionally, we present complete results for the best parameter configurations per experiment in Appendix B.

5.5 Parameter Settings

- Similar to Rekabsaz et al. [RLHZ16], for the Wikipedia and CLEF-IP word embedding models we created 300 dimensional vectors with the Skip-gram model running for 25 epochs, using the sub-sampling parameter set to 10^{-5}, a context window of 5 words, and word count threshold of 20.
- We computed the Retrofitting procedure with the following parameter configuration: $\alpha = 1$, and β was set to a normalized similarity score as shown in Equation 4.1. Setting alpha to 1 is recommended by Faruqui et al. [FDJ+15].
- We explored the threshold for the LSI data selection in a range from 0.5 to 0.9 in steps of 0.02. We explored the threshold for the Translation Model in a range from 0.6 to 1 in steps of 0.01. When the Translation Model threshold is set to 1 the query equals a plain query, meaning no additional terms are added to a query. We used these ranges, because they provide a reasonable robust parameter search space, where the results at the beginning and end of the ranges are not optimal and commonly show the lowest results.
- Similar to Rekabsaz et al. [RLHZ16], for BM25 we set $b = 0.6$, $k1 = 1.2$, and $k3 = 1000$, and for LM we set μ to 1000.

CHAPTER 6

Evaluation and Results

In this chapter we present our word embedding and word similarity analysis, which measures changes at different stages in the evaluation pipeline. We present the cross-validated results in domain-specific Information Retrieval test collections for our similarity models. In addition we present insights into the parameter exploration we conducted to find the best Translation Model threshold.

6.1 Word Embedding Analysis

The word embedding analysis starts with an analysis of the global differences in the vector spaces, followed by a local neighborhood analysis.

6.1.1 Retrofitting Changes

During the Retrofitting procedure every word that has similarity information in the external resource, moves in the vector space, as shown in Section 2.2. In this section, we empirically analyze relative differences and compute metrics that capture the changes in the word embedding.

Figure 6.1 shows the histogram of normalized Euclidean distances between the original and retrofitted positions of the vectors for the CLEF-IP corpus. The x-axis depicts the distance between the original and new positions. The lower the LSI similarity threshold is set the more vectors change. At a threshold of 0.8, 272,000 vectors remain unchanged, this is 50 percent higher than at 0.7. A lower threshold – more input data – does not mean, that the vectors move farther away, most of the vectors move by a distance between 0.35 and 0.5. The number of vectors that are moved in this range increase by 40 percent between the thresholds of 0.8 and 0.7. There are no vectors that move farther than the last shown range, therefore we omit the empty ranges from the histogram. Using the

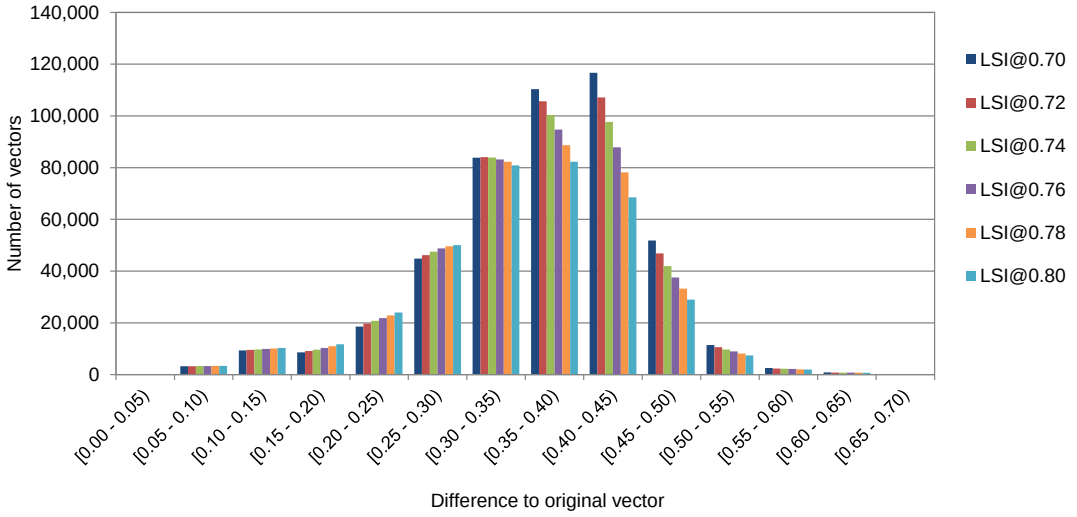

Figure 6.1: Histogram of normalized euclidean distances between the original vectors and the retrofitted vectors using different LSI thresholds for a CLEF-IP word embedding

Wikipedia corpus with LSI for the TREC test collections, we can observe a very similar pattern and therefore the collection is not displayed.

Using a semantic lexicon like PPDB or WordNet, which both have much less information on words, the histogram analysis still shows the same characteristics. It is important to note for the following analysis and results, that the semantic lexicons change much fewer vectors. Depending on the exact configuration only a fifth or less. But with every additional neighboring vector, the relative influence of each neighbor becomes lower in the Retrofitting formula, and therefore changing fewer vectors still shifts the similarity results of the word embedding. When we combine LSI with PPDB data in the `Ext-Retro(*,*)` model the histogram keeps its Gaussian-like shape, but the distance range moves closer and most changes happen between 0.15 and 0.35.

As a first observation we can conclude that the Retrofitting procedure (especially with LSI) overall has sizable impact on the global changes of the vector space.

6.1.2 Quantitative Neighborhood Analysis

In this section, we focus on the changes inside local neighborhoods, because we use the neighborhoods of individual vectors from the word embedding in the GT and ET retrieval models. A neighborhood is defined as a set of similar words, whose similarity to the main word is higher than a threshold.

Figure 6.2 shows the average number of neighbors for different thresholds using a CLEF-IP word embedding for all unique query words from the CLEF-IP-2013 test collection. We counted the number of neighbors with similarities higher than the threshold per query

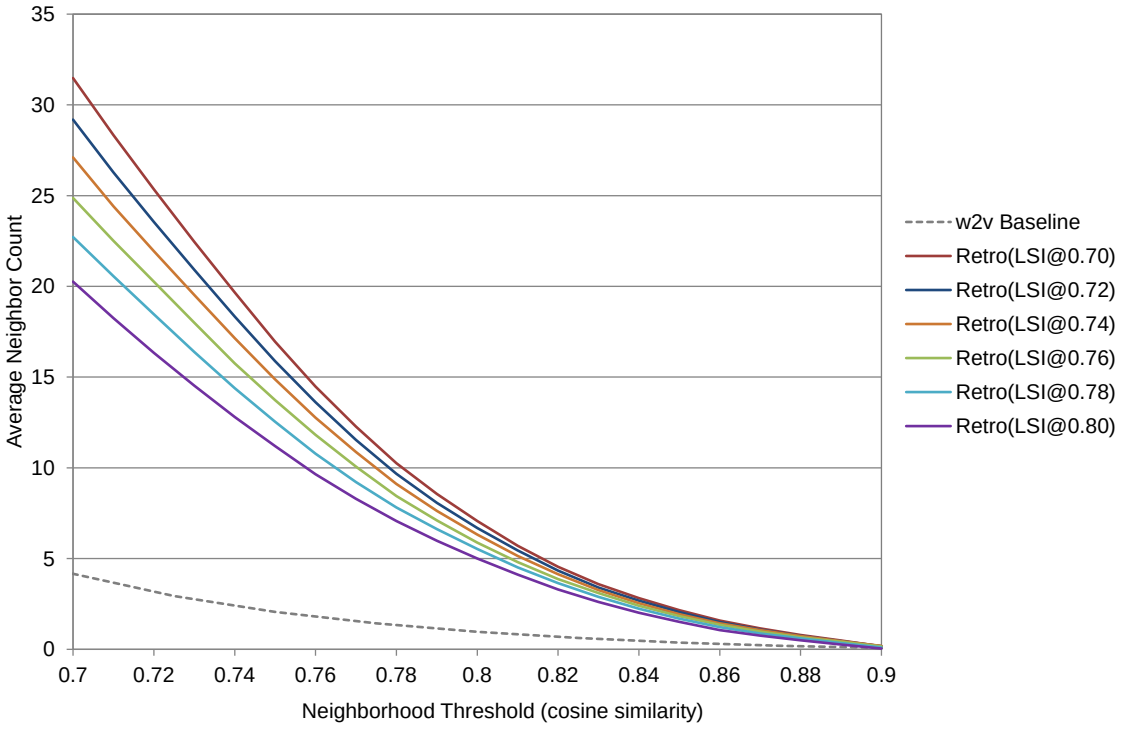

Figure 6.2: Number of average common words in the neighborhoods of query words for different cosine similarity (Translation Model threshold) for a CLEF-IP word embedding and CLEF-IP-2013 query words

word and divided the result by the number of unique query words for the average. The gray line on the bottom shows the unchanged `w2v` baseline. The lower the LSI threshold (e.g. the more information is used in the Retrofitting) the higher the neighborhood size for the same similarity. This figure shows that the vectors move closer to each other after the Retrofitting process, because more words are located in the same similarity range. Therefore, we state that the word embedding contracts during the Retrofitting.

We experiment with different similarity models as listed in Section 4.1 – Figure 6.3 visualizes differences in the neighborhoods of different similarity models. The x-axis shows different neighborhood sizes, defined by a cosine similarity. They y-axis shows the average intersection of the neighborhoods of two word embeddings, e.g. how many similar words are in the neighborhoods of the two.

For this analysis we use a Wikipedia based baseline word embedding and word embeddings retrofitted with LSI similarities created from Wikipedia and the semantic lexicon PPDB. We use the LSI similarity threshold of 0.7. We also evaluated an extended model which combines LSI@0.7 and PPDB data during the Retrofitting process. We analyze

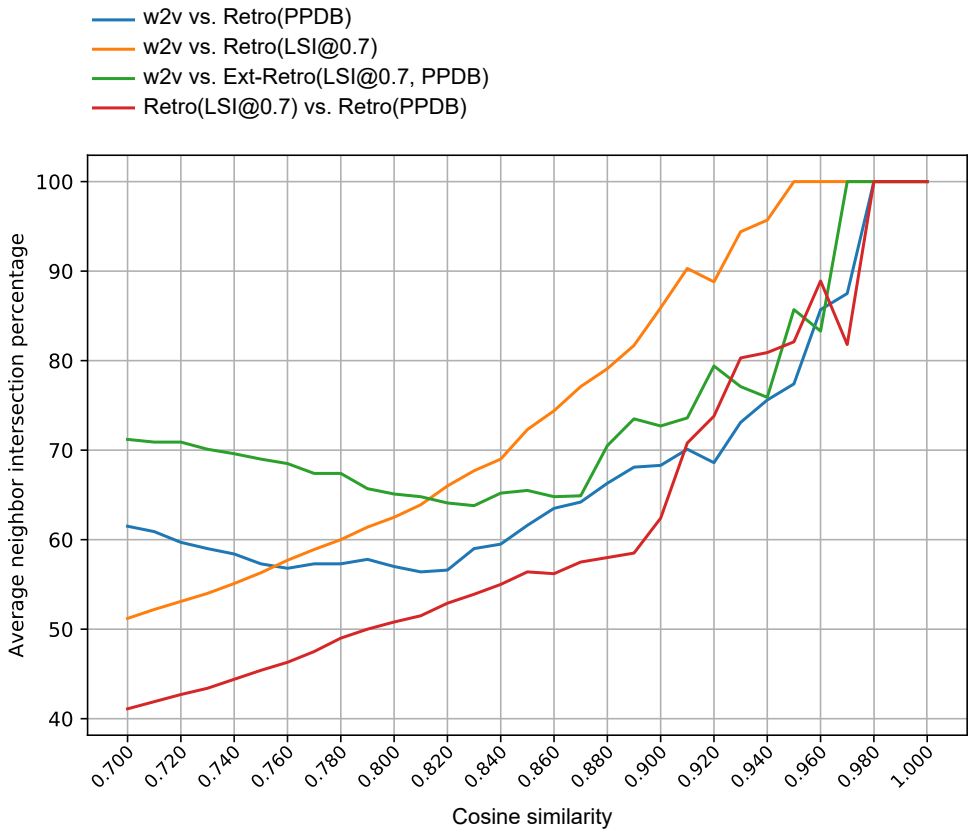

Figure 6.3: Number of common neighbors in various similarity values for Wikipedia-based word embeddings and TREC-Robust-2004 query words

neighborhoods of all unique query words from the TREC-Robust-2004 test collection.

We pairwise compare the query word neighborhoods of each created word embedding with each other by computing the number of common words between the two sets. The two sets are required to have the same number of words, so we can compare them fairly. Simply applying the same similarity threshold to both embeddings does not guarantee the same number of neighbors, in fact as Figure 6.2 shows the sizes of the neighborhoods differ substantially. We apply a similarity threshold per word to the first word embedding, which sets the number of similar words used for the second embedding. The displayed percentage is an average over all unique query words.

Compared to the original Skip-gram word embedding, `Retro(LSI)` (orange) changes fewer words in the query word neighborhoods in comparison to `Retro(PPDB)` (blue) on small neighborhoods. In larger neighborhoods, e.g. with a lower cosine similarity,

Retro(LSI) changes more words around the query words than Retro(PPDB). By comparing the intersection of neighbors between two retrofitted word embeddings (red), we can observe that the neighborhoods differ more than each individually to the original word embedding. This means that different external resources do change the word embedding in different ways.

6.2 Information Retrieval Results

In this section we present a comparison of our similarity model results, which are used by the evaluated retrieval models and a detailed parameter exploration for selected models.

6.2.1 Evaluation Results

We focus our comparison of our model results using cross-validation, described in Section 5. The baselines and their symbols for significance are summarized in Table 6.1. The STD retrieval model is a query without any changes. We report statistical significance using a two sided paired t-test and $p < 0.05$.

In the following we present the results per test collection: TREC-123, TREC-Robust-2004, and CLEF-IP-2013.

Table 6.1: Baselines and their symbols for the significance tests

Baseline	**Tested from**	**Sig. Symbol**
STD	All the models and baselines	†
w2v	Retrofitting and Post-Filter models	ρ
Post-Filter(LSI)	Retrofitting models	ν

TREC-123 The TREC-123 test collection is in the news domain. Table 6.2 shows the results of our word similarity models for BM25-GT, BM25-ET, LM-GT, and LM-ET retrieval models. The baselines are reported in the first section, the second section contains single word embedding similarity models, the third section contains similarity models that introduce an additional similarity selection step with two external resources. The best result per measure over all retrieval models is displayed in bold.

The w2v similarity model does not significantly improve over the baseline. From the similarity models with a single external resource Retro(LSI) and Retro(PPDB) improve significantly over the STD baseline. The best performing models for the TREC-123 collection are Post-Filter-Retro(LSI,PPDB) and Retro(LSI) + Retro(PPDB). Both are statistically significantly better than all baselines in the MAP score with BM25-GT and BM-ET. Both models combine two resources during the selection of similar terms.

Table 6.2: Results for TREC-123 per similarity model, Translation Model, and scoring method showing different measures

	Method	T. Model	BM25 MAP	BM25 NDCG	BM25 P@10	LM MAP	LM NDCG	LM P@10
TREC-123	STD		0.222	0.431	0.488	0.222	0.437	0.505
	Plain-w2v	GT	0.225	0.430	0.485	0.222	0.435	0.511
		ET	0.225	0.432	0.477	0.224	0.435	0.507
	Post-Filter(LSI)	GT	0.226	0.437†ρ	0.497	0.226	0.439	0.506
		ET	0.228	0.440†	0.489	0.227	0.442ρ	0.511
	Retro(FrameNet)	GT	0.228	0.427	0.501	0.224	0.438	0.517†
		ET	0.230	0.432	0.487	0.227	0.439	0.503
	Retro(PPDB)	GT	0.226	0.434	0.507ρ	0.229†ρ	0.438	0.515
		ET	0.232†	0.440	0.497ρ	0.227	0.440	0.510
	Retro(WordNet.synonyms)	GT	0.226	0.434	0.506	0.225	0.438	0.505
		ET	0.229	0.432	0.503ρ	0.222	0.434	0.513
	Retro(WordNet.synonyms+)	GT	0.223	0.438	0.481	0.223	0.441	0.520†
		ET	0.226	0.429	0.482	0.220	0.429	0.520†
	Retro(LSI)	GT	0.229†	0.440†ρ	0.494	0.229†ρ	0.443†	0.511
		ET	0.229†	0.441†ρ	0.497ρ	0.228	0.442	0.512
	Ext-Retro(LSI + PPDB)	GT	0.227	0.436	0.502	0.227ρ	0.439	0.517
		ET	0.228†	0.439†	0.501ρ	0.225	0.438	0.518†
	Post-Filter-Retro(PPDB, LSI)	GT	0.234†ν	0.441†ρ	0.502	0.233†$\rho\nu$	0.444	0.520ν
		ET	**0.236**†$\rho\nu$	0.444†ρ	0.502ρ	0.232†ρ	0.442	0.513
	Retro(PPDB) + Retro(LSI)	GT	0.235†$\rho\nu$	0.442†ρ	0.511†ρ	0.234†$\rho\nu$	0.444ρ	**0.521**
		ET	0.235†$\rho\nu$	0.442†ρ	0.505ρ	0.232†ρ	**0.444**ρ	0.518

The baseline `STD` MAP results for both BM25 and LM are very close to each other in the TREC-123 test collection. For a majority of similarity models BM25 with the Generalized Translation Model (BM25-GT) performs best in MAP. The best results are achieved with BM25-ET but closely followed by the other scoring methods. The results clearly show that this test collection benefits from a similarity model built on two different external resources during the selection of similar terms. However, the similarity model that combines two resources into a single word embedding `Ext-Retro(PPDB + LSI)` fails to improve over the single resource models.

TREC-Robust-2004 TREC-Robust-2004 covers the same general news domain as TREC-123. The results for all measures and models are shown in Table 6.3. The table has the same structure as Table 6.2 containing three sections and the best result per measure over all retrieval models is displayed in bold.

The `w2v` and `Post-Filter(LSI)` similarity models perform significantly better in the MAP measure than the `STD` baseline – for all four retrieval models. None of the other similarity models can outperform the `w2v` and `Post-Filter(LSI)` baselines. Only the `Retro(PPDB)` similarity model improves the MAP score slightly above the `w2v` results, but not significantly. `Retro(FrameNet)` even leads to results that are below the `STD` minimum.

The NDCG measure also shows that models based on PPDB and the two baselines `w2v` and `Post-Filter(LSI)` are very similar in their results and they all improve over the

Table 6.3: Results for TREC-Robust-2004 per similarity model, Translation Model, and scoring method showing different measures

	Method	T. Model	BM25 MAP	BM25 NDCG	BM25 P@10	LM MAP	LM NDCG	LM P@10
TREC-Robust-2004	STD		0.263	0.535	0.449	0.264	0.538	0.435
	Plain-w2v	GT	0.276†	0.546†	0.439	0.276†	0.551†	0.424
		ET	0.279†	0.551†	0.443	0.276†	0.549†	0.424
	Post-Filter(LSI)	GT	0.277†	0.547†	0.447	0.278†	**0.551**†	0.436ρ
		ET	0.277†	0.548†	0.447	0.276†	0.548†	0.436ρ
	Retro(FrameNet)	GT	0.259	0.527	0.447	0.269	0.535	0.432
		ET	0.258	0.530	0.449	0.266	0.536	0.432
	Retro(PPDB)	GT	0.281†	0.548†	0.453ρ	0.280†	0.550†	0.415
		ET	**0.283**†	0.550†	0.456ρ	0.276†	0.548†	0.431
	Retro(WordNet.synonyms)	GT	0.273†	0.537	0.450	0.271	0.541	0.439ρ
		ET	0.273†	0.542	0.452	0.269	0.541	0.437ρ
	Retro(WordNet.synonyms+)	GT	0.269	0.538	0.453ρ	0.268	0.541	0.429
		ET	0.270	0.536	0.454ρ	0.267	0.540	0.431
	Retro(LSI)	GT	0.274†	0.544	0.448	0.272	0.544	0.430
		ET	0.275†	0.547†	0.450	0.269	0.542	0.439ρ
	Ext-Retro(LSI + PPDB)	GT	0.279†	0.550†	0.450ρ	0.274†	0.549†	0.435ρ
		ET	0.280†	0.551†	0.456ρ	0.273†	0.547†	0.433
	Post-Filter-Retro(PPDB, LSI)	GT	0.277†	0.548†	0.451ρ	0.274†	0.549†	0.437ρ
		ET	0.276†	0.547†	0.458$\rho\nu$	0.272	0.549†	0.435
	Retro(PPDB) + Retro(LSI)	GT	0.275†	0.546†	0.457ρ	0.275†	0.551†	0.429
		ET	0.279†	0.546†	**0.459**$\rho\nu$	0.273†	0.548†	0.431

`STD` baseline. Other `Retro(*)` models based on LSI, WordNet, and FrameNet fail to improve significantly over `STD`.

CLEF-IP-2013 The CLEF-IP-2013 test collection consist of patent text. Table 6.4 shows all similarity model results for CLEF-IP-2013 in the same structure as the result tables before.

The MAP results for the `w2v` similarity model show no changes in comparison to the `STD` baseline. All of the other similarity models increase the result significantly over the two `STD` and `w2v` baselines. `Retro(PPDB)` works well with LM-ET only. The other similarity models perform best with BM25-ET. The best methods are `Post-Filter(LSI)` (with LM-GT and BM25-ET) and `Post-Filter-Retro(PPDB, LSI)` (with BM25-ET) for the MAP measure.

The LM based retrieval models outperform BM25 on the Recall measure. The `w2v` similarity model does not change the Recall results in comparison to the `STD` baseline. The `Retro(LSI)` model performs significantly better than `STD` and `w2v`. The other models do not show a clear pattern of improvement. Some perform better than the baseline, but not significantly over both baselines.

Discussion We conclude that the three different test collections require different similarity models to achieve the best results. Overall the best similarity method is

Table 6.4: Results for CLEF-IP-2013 per similarity model, Translation Model, and scoring method showing different measures

	Method	T. Model	BM25 MAP	BM25 PRES	BM25 RECALL	LM MAP	LM PRES	LM RECALL
CLEF-IP-2013	STD		0.184	0.607	0.703	0.200	0.669	0.755
	Plain-w2v	GT	0.185	0.611	0.685	0.192	0.671	0.751
		ET	0.207†	0.615†	0.679	0.200	0.665	0.758
	Post-Filter(LSI)	GT	0.215ρ	0.613	0.723	**0.248**†ρ	0.669	0.765
		ET	0.247†ρ	0.638†ρ	0.733ρ	0.228†ρ	0.689	0.785
	Retro(FrameNet)	GT	0.180	0.607	0.707ρ	0.204	0.667	0.735
		ET	0.206	0.633†	0.698	0.188	0.661	0.762
	Retro(PPDB)	GT	0.205ρ	0.594	0.682	0.198	0.655	0.765
		ET	0.194	0.625	0.715	0.240†ρ	0.667	0.758
	Retro(WordNet.synonyms)	GT	0.167	0.618	0.691	0.224†ρ	0.668	0.758
		ET	0.206	0.610	0.705	0.208	0.651	0.717
	Retro(WordNet.synonyms+)	GT	0.169	0.595	0.687	0.204	0.659	0.738
		ET	0.180	0.597	0.674	0.207	0.638	0.754
	Retro(LSI)	GT	0.227ρ	0.605	0.709	0.235†ρ	0.664	0.733
		ET	0.238†	0.639†	0.733ρ	0.221	**0.698**	**0.812**†ρ
	Ext-Retro(LSI + PPDB)	GT	0.213	0.611	0.671	0.233†ρ	0.667	0.768
		ET	0.239†ρ	0.624	0.733ρ	0.227†ρ	0.669	0.765
	Post-Filter-Retro(PPDB, LSI)	GT	0.226ρ	0.610	0.712	0.233†ρ	0.664	0.752
		ET	0.246†ρ	0.643†ρ	0.733ρ	0.218†ρ	0.686	0.788ρ
	Retro(PPDB) + Retro(LSI)	GT	0.216ρ	0.598	0.691	0.217†ρ	0.667	0.785
		ET	0.221†	0.639†ρ	0.743ρ	0.216	0.676	0.798ρ

`Post-Filter-Retro(PPDB, LSI)` which performs best on TREC-123 and CLEF-IP-2013 and shows the same results as the baseline in TREC-Robust-2004. This model requires two resources to select the threshold: the retrofitted word embedding with PPDB and the LSI similarities to filter the final result list. This model is more complex than a single word embedding. If one wants to use only a single word embedding the `Ext-Retro(LSI + PPDB)` similarity model shows robust results over all three test collections. In the news domain the semantic lexicons have more benefits than in the patent domain, where the best results are combinations of LSI.

The best overall retrieval model is BM25-ET, which in most cases provides the best results or is not significantly worse than the best results. An exception is the Recall metric on the patent collection, where a combination of LSI and LM provides a significantly better result than all other experiments.

6.2.2 Threshold parameter exploration

In this section we observe changes in results based on the changes in the Translation Model and LSI threshold parameters. We focus on Retrofitting with one resource to keep the visualization tractable.

We discussed in Section 4.2 that a single experiment depends on the Translation Model, the scoring method, the Translation Model threshold, and an external resource configuration.

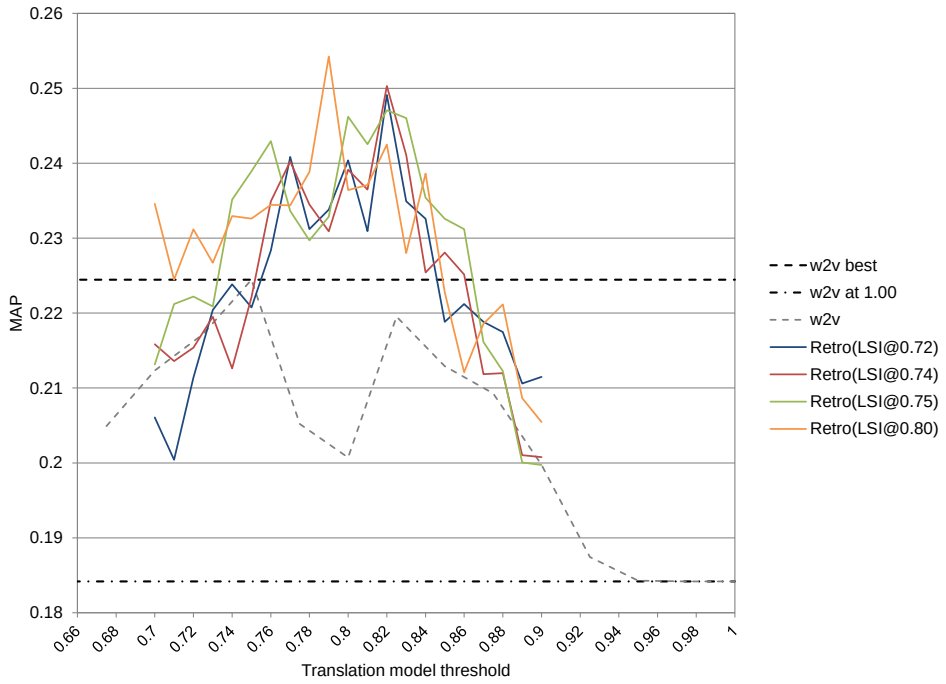

Figure 6.4: Results for different Translation Model thresholds and different LSI similarity thresholds used in Retrofitting for CLEF-IP-2013 with BM25-ET

As mentioned above, we always evaluated a range of possible thresholds. The following figures show selected results of this parameter range exploration that represent our experiments. We choose to visualize the BM25-ET MAP results, as they have proven to be overall best retrieval results. Each line in the following figures represents a retrofitted word embedding with a different external resource configuration.

When we compare models alongside the Translation Model threshold parameter we use the best baseline results as the comparison for all parameter settings – since the best Translation Model threshold is shifted, because of the contracted word embedding. The best `w2v` and `STD` baseline results are visualized by horizontal black dotted lines. The complete best result tables for all models and measures are located in Appendix B.

Figure 6.4 shows detailed results for the CLEF-IP-2013 test collection and the `Retro(LSI)` model. The `w2v` baseline (gray dotted line) does not have a clear maximum, whereas a single global maximum is more common in other collections using the Translation Models. Retrofitting the Skip-gram word embedding with LSI data transforms the results to have a single global maximum at the Translation Model threshold of around 0.8. Using the LSI threshold of 0.75 provides robust results over multiple Translation Model thresholds. The best result is a MAP score of 0.254 which is significantly higher than both baselines.

Figure 6.5 shows the results for the TREC-Robust-2004 test collection. It uses LSI data

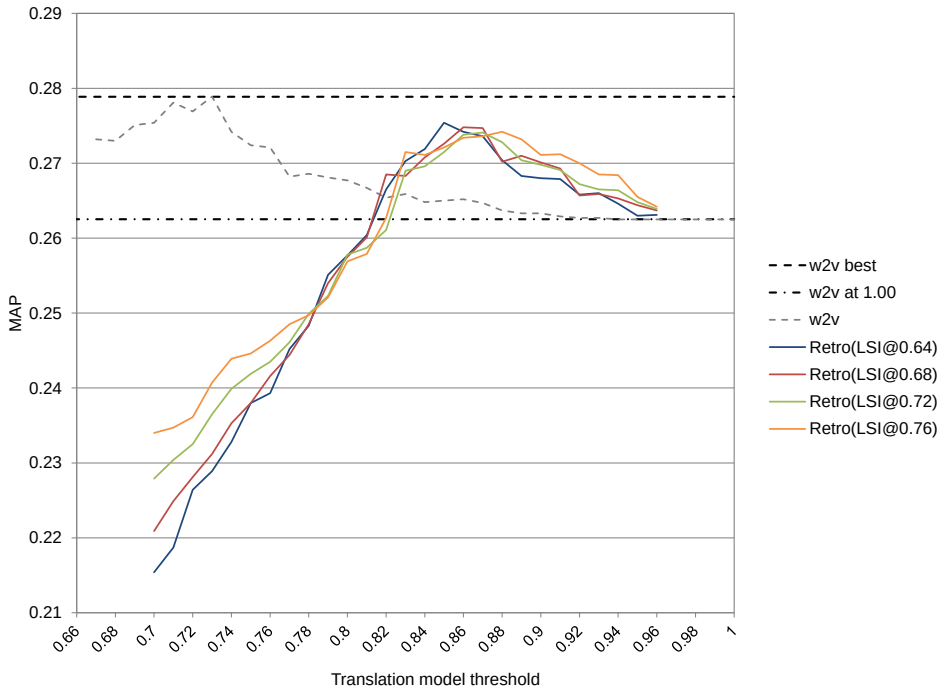

Figure 6.5: Results for different Translation Model thresholds and different LSI similarity thresholds used in Retrofitting for TREC-Robust-2004 with BM25-ET

at different thresholds to retrofit the Skip-gram word embedding (`Retro(LSI)`). The Translation Model threshold achieving the best results shifts higher to 0.86, still all LSI retrofitted word embeddings fail to improve the results of the base word embedding. Using the best threshold of the base word embedding at 0.7 with the LSI word embeddings the results decrease the results considerably under the `STD` baseline. The results are falling below the `STD` baseline much faster than the base word embedding if the Translation Model threshold is not selected carefully. A low LSI threshold only amplifies this effect.

Figure 6.6 depicts results using semantic lexicons in the Retrofitting for TREC-Robust-2004 (`Retro(Semantic Lexicons)`). Note that the scale of the y-axis (the MAP values) is very fine grained, therefore a seemingly big difference in the plot is actually very small. As shown in Table 6.3 the best baseline result for `w2v` is significantly higher than the `STD` result, but there is no statistically significant difference between the `w2v` baseline, `Retro(PPDB)`, and `Retro(WordNet-synonyms)`. As mentioned before only `Retro(PPDB)` improves the absolute result value. Again the Translation Model threshold providing the best results increases.

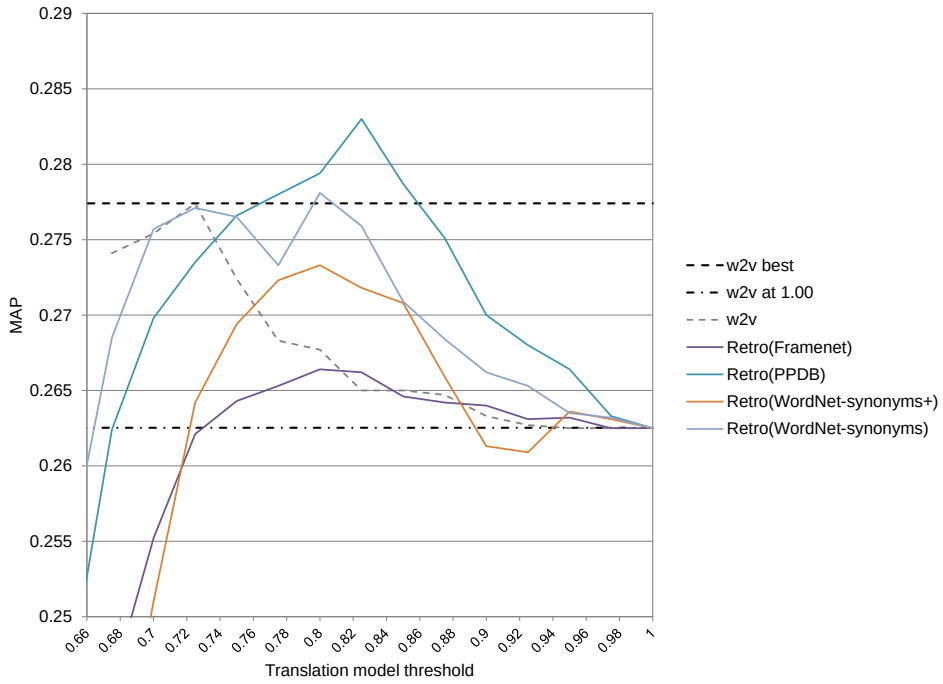

Figure 6.6: Results for different Translation Model thresholds and different semantic lexicons used in Retrofitting for TREC-Robust-2004 with BM25-ET

6.3 Summary

In the Information Retrieval results, we observe similar effects outlined in the word embedding analysis (Section 6.1): 1. Different similarity models produce different similarity results and those results do change the the overall retrieval results. We tracked these changes from the core of the word similarity model to the different retrieval result measures. 2. We conclude that overall for our three test collections and most data sources a higher Translation Model threshold is needed to achieve the best results. This shows with the neighborhood analysis, that even with a retrofitted embedding (where words are closer to each other) only a certain low amount (1-3 words on average) of words improve the retrieval results. This is a similar average to the `w2v` baseline.

CHAPTER 7

Conclusion

Incorporating similar words in a document retrieval query gained from a word embedding offers great potential for better retrieval results. The crucial point is to expand the search space with topic-related similar words, so that the expansion improves retrieval results. Expanding the query words with too many or query unrelated words reduces the retrieval performance, because results diverge into unrelated topics.

We hypothesized that the word embedding model that provides the representation of words should not only be created based on a window-context (used by the Skip-gram word embedding), but on a diverse set of input sources with different informational value. In this thesis, we incorporate various additional – external resources – into a Skip-gram word embedding in a post-processing step using the Retrofitting method. This adds more information to the embedding which is not captured in a window-context of words alone. We create word embedding based word similarity models, which are used to incorporate similar words in a domain-specific Information Retrieval query. We implemented the used retrieval models in the popular open source search engine Lucene. We compared our novel word similarity models with each other and analyzed how the word embeddings change during the Retrofitting process. We evaluated a set of parameter ranges for the selection of similar words in combination with four different retrieval models. We evaluated two news collections (TREC-Robust-2004, TREC-123) and one patent collection (CLEF-IP-2013).

Our retrofitted word embedding analysis showed that the more data is used to retrofit, the more the word embedding changes and the word vectors move closer to each other. We also showed that different external resources produce indeed different word similarity results. Overall the best word similarity model is the `Post-Filter-Retro(PPDB, LSI)` model, which is a Post-Filter and Retrofitting combination of two external resources: PPDB and LSI. In this model we combine as many inputs as possible (window-context, document-context, and semantic lexicon). The model improves significantly over an out-of-the box word embedding baseline on the TREC-123 news collections and the CLEF-IP-2013 patent collection. However, in the TREC-Robust-2004 news collection we

did not observe an improvement in comparison to an out-of-the box word embedding baseline. The TREC-Robust-2004 news collection is best served with the `Retro(PPDB)` model, which retrofits information from the semantic lexicon PPDB into a Skip-gram embedding.

CHAPTER 8

Future Work

During the work on this thesis we touched multiple areas that are interesting to pursue, but outside the scope of this thesis. In the following, we summarize our ideas for future work:

- *De-bias adapted word embeddings* Rekabsaz et al. [RLH17a] and Bolukbasi et al. [BCZ+16] showed the existing gender bias in the English Wikipedia corpus, which we use as a basis for our news domain word embedding. Bolukbasi et al. propose a method to remove gender and other measurable bias, such as racial discrimination or stereotypes, from an embedding without breaking the useful unbiased relationships of the vectors. We aim to apply this method on our new word embedding models and measure the effect it has on the evaluation tasks. Not using de-biased embeddings can be a potential problem for every downstream task, which does not explicitly study the bias because the bias only gets amplified. For example, a retrieval task that expands a neutral query with biased words will yield returned documents that contain more biased results than an unchanged query. Therefore, we find it ethical to use word embeddings in a system that are de-biased.
- *Using context everywhere* Currently the Translation Models as well as the word similarity models do not use the complete query context of a query word in selecting which words are used to expand the query. The system can not distinguish polysemous words, such as "bank". When a user searches for "bank loan", the system should be able to recognize the correct meaning of "bank", given by the second word. The solution requires changes in multiple parts of the search system:
  - *Context based embedding* Use or create an embedding to capture polysemy, where each word can have multiple vectors (and therefore different neighborhoods) based on multiple senses. This should be learned from the corpus that is used to create the embeddings.

Neelakantan et al. [NSPM14] proposed an extended version of the Skip-gram model, which automatically learns different senses or meanings per word. Every meaning of a word is captured in an individual vector. The model does not require a parameter to set the number of meanings a word has. This number is also learned automatically per word. The training time stays reasonable in the range of hours, not days. Neelakantan et al. describe a nearest neighbor search that combines the neighborhoods of the individual vectors of a word, but the model itself would also allow to retrieve the lists of neighbors per sense individually. This is not a focus of their paper and could be a new contribution in combination with context words provided by queries. Rekabsaz et al. [RLHZ17] did study the positive and negative impact of similar words to the MAP metric. They concluded that with a lower similarity threshold the potential gains are high, but diminished by negative words that cause topic shifting. Using a multi-sense embedding with a context dependent sense selection could potentially filter out more negative words. As a first step, the analysis by Rekabsaz et al. should be reevaluated with multi-sense embeddings.

  - *Multi-word or phrase selection* Currently information about phrases or context about topics gained from inspecting multiple query terms is not used: Searching for "United Kingdom" expands the individual words, but not the phrase of the combined two words. Each individual word is again expanded with single words. A good solution would be to expand this phrase with another phrase like "Great Britain". Using phrases to select similar words could provide a very useful tool against topic shifting - if the topic is encoded in the query.
  - *Similar term selection learning* Learn to select similar terms based on the whole query with a recurrent neural network and some form of reinforcement learning based on test collection gold standard ranking data. Peters et al. [PNI+18] propose deep contextualized word representations in the context of NLP applications. Their recurrent model builds on LSTM's and a nearest neighbor query is conducted with the context of a word in form of a sentence. They show that they can improve a variety of NLP tasks, when their representation is applied to the previous state-of-the-art models.

- *Supervised word representation learning* The Retrofitting approach is an unsupervised method. Using a supervised learning method could further improve a word embedding. The supervision could be gained from retrieval gold standard rankings. This would tailor the embedding specifically to the Information Retrieval domain.

- *Use a subword embedding* We used TREC/CLEF-IP test collections with complete queries, without incorrect spelling, or incomplete words. Therefore the test framework does not reflect the scenarios in which users make spelling mistakes or search for parts of words. Together with a subword matching in the search engine a subword embedding, like FastText by Bojanowski et al. [BGJM17], could prepare the system for a more real world usage scenario.

- *Evaluate languages other than English* This thesis as well as much of the research into word embeddings and text understanding centers around the English language. The models used in this thesis should also be evaluated on other languages, to test their usability outside of English speaking countries and applications.

- *Improve nearest neighbor search* One of the longest running tasks of the evaluation pipeline is to compute the nearest neighbors of all terms that have to be evaluated. For a performance oriented production system a complete computation of all neighbors in the vocabulary is necessary. As possible direction to address this problem Garcia et al. [GDNB10] propose an exact GPU-based method to retrieve k-nearest neighbors. Andoni et al. [AI08] analyze "Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions", especially the concept of locality-sensitive hashing (LSH). This concept is implemented in the open-source project Annoy[1]. Potentially the approximation method could be used during research and before publishing an exact brute-force GPU-method could be used to generate a list of all similar terms.

[1] See: https://github.com/spotify/annoy *checked: 8.3.2018*

APPENDIX A

Translation Models implemented in Lucene

This appendix describes implementation details of the Lucene plugin that implements the GT and ET retrieval models. The code and evaluation for the extension is available at: *https://github.com/sebastian-hofstaetter/ir-generalized-translation-models*

The extension project extends Lucene specific as well as Solr specific parts in the query pipeline as shown in Figure A.1. It has a Solr specific query parser (*SimilarityParser*) that can be used in the Solr configuration (*SimilarityParserPlugin*). The parser sends the query terms (pre-processed through the specified analyzer) to the *SimilarityApi*. The API is not part of this project.

The parser than creates Lucene query objects (*AugmentedTermQuery*), used by the search system. Multiple queries are concatenated via a built-in *BooleanQuery*. Each *AugmentedTermQuery* provides an *AugmentedTermWeight* which in turn provides an *AugmentedTermScorer* that coordinates the iteration over the found documents as well as the scoring of them. The *AugmentedTerm** classes are tightly coupled and cannot be used in a standalone fashion. Only the *AugmentedTermQuery* is used by other components.

Every class is tested with unit or integration tests. These tests provide a good example of how the classes are used.

In the following we describe the parts and classes of the extension project:

SimilarityParser The *SimilarityParserPlugin* is a factory method to create a new instance of the *SimilarityParser*. This is specified by Solr.

The *SimilarityParser* hardcodes as little configuration as possible: It uses the defined analyzer from the Solr schema, the default search field, and the API parameters are set in the configuration. Each term tokenized by the analyzer will be used to create an

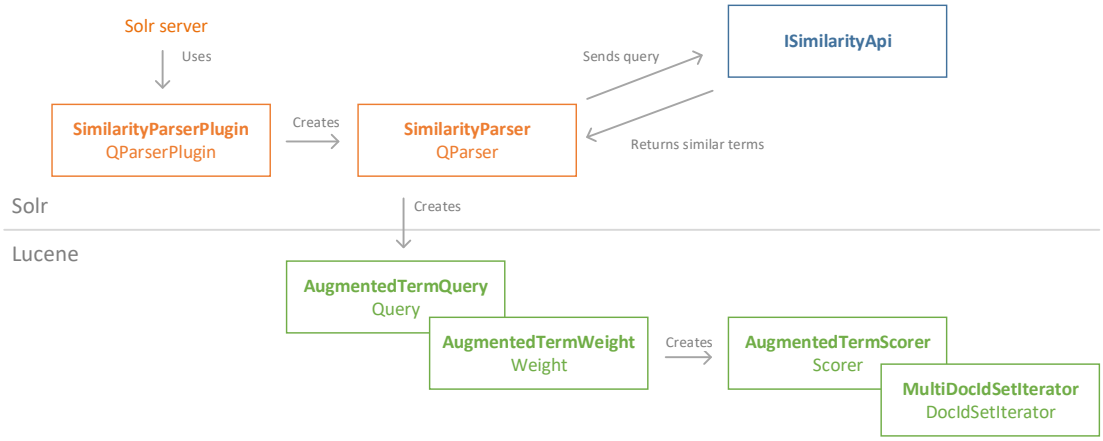

Figure A.1: Overview of the extension types needed to implement a Lucene & Solr extension

AugmentedTermQuery with information gathered from the *SimilarityApi*. The API is only accessed once per query - with a list of all terms. The API then returns similar terms and their distances to the main term. These results are parsed in Lucene usable *Term* objects that are used in the *AugmentedTermQuery*.

AugmentedTermScorer and AugmentedTermQuery The structure of the *AugmentedTerm** classes follows the requirements of how a Lucene search is conducted. Figure A.2 shows this workflow. The interaction with the Lucene query execution system cannot be changed and the extension classes have to work in the intended workflow for a plugin. Specifically, the extended term frequency that is calculated by the *AugmentedTermScorer*. The other components actively contribute information so that the term frequency of the main term and the sum of the weighted similar term frequencies can be calculated.

Note: Lucene can call Weight.scorer() multiple times, for each *LeafReader* that is used, but it does not change the main interaction and is therefore omitted from Figure A.2.

The *AugmentedTermQuery* is used as an interface for external code that uses the extension and it starts the search process. The functionality itself is limited to storing the term and similar term information as well as pulling the *TermContext* for each term. It is important that each unique term has its own context otherwise the *PostingsEnum* would be overridden in a later stage and create wrong results without throwing an error.

The term information is formed as follows (it is set in the constructor):

- Main term
- List of <*term, weight*> tuples, i.e. the similar terms

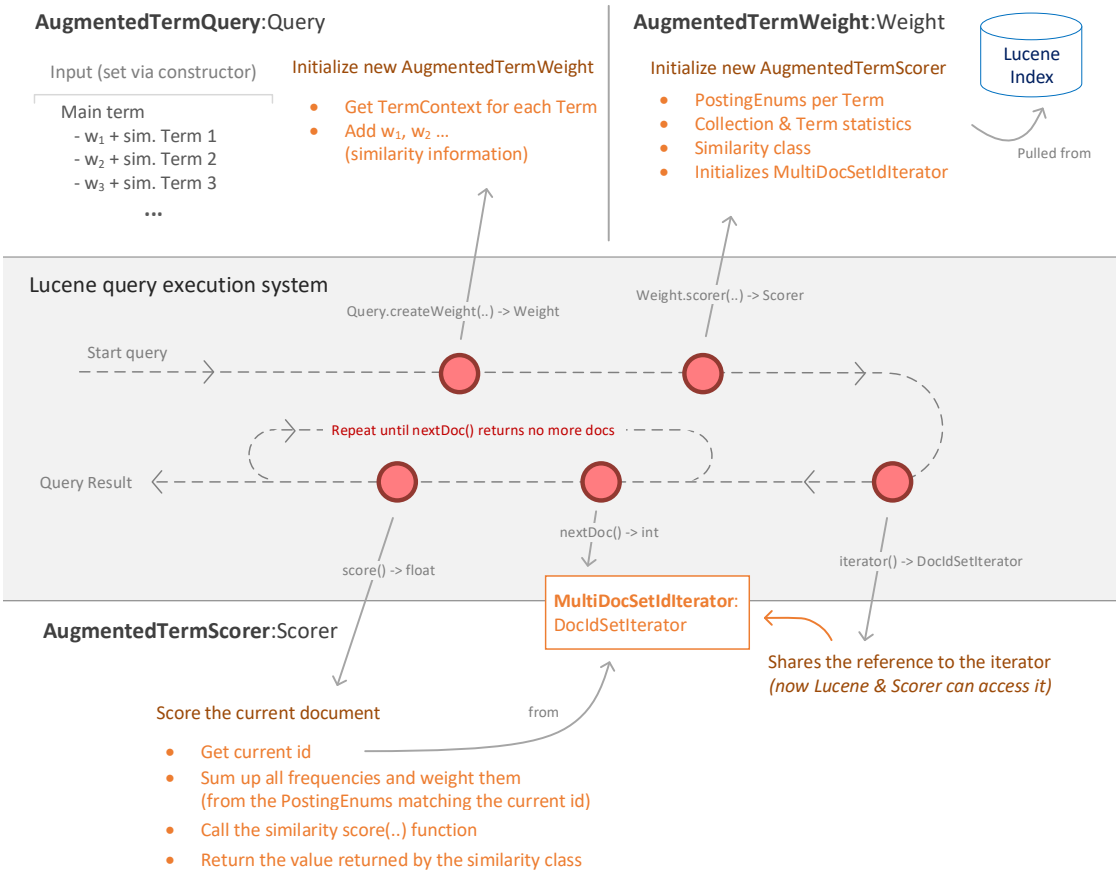

Figure A.2: Plugin types in cooperation with the Lucene search process

AugmentedTermWeight The *AugmentedTermWeight* is an inline class of *AugmentedTermQuery*, because it has to access the saved term information. The *AugmentedTermQuery* is cache-able by Lucene, because only the state of the *AugmentedTermWeight* changes during a query.

The *AugmentedTermWeight* coordinates the query: it gathers the needed reference to the *Similarity* class, the *CollectionStatistics*, the *TermStatistics*, the *PostingsEnum* for each *TermContext*, and it initializes the *AugmentedTermScorer*.

AugmentedTermScorer The *AugmentedTermScorer* works together with the *MultiDocSetIdIterator*. The Lucene query system has access to the iterator via a reference obtained by the *iterator()* method. The query system advances the iterator to the next document.

The iterator wraps all *PostingsEnum* iterators (main + similar), and iterates through the union of all documents in a linear fashion. It changes the state of all wrapped

PostingsEnum, so that the *AugmentedTermScorer* can access the postings frequency information at scoring time. It aligns multiple *PostingsEnum* instances with to the same document id (if they contain the same document). The iterator only emits ascending document id values, as defined by the specification. All *PostingsEnum* iterators are treated equally.

APPENDIX B

Result Tables

In this Appendix we present the results for the three test collections with the best parameter configurations for each experiment – as decided by our parameter exploration.

Table B.1: Results for the best parameter configurations for CLEF-IP-2013

	Method	T. Model	BM25 MAP	BM25 PRES	BM25 RECALL	LM MAP	LM PRES	LM RECALL
CLEF-IP-2013	STD		0.184	0.607	0.703	0.200	0.669	0.755
	w2v	GT	0.214†	0.611	0.709	0.218†	0.671	0.761
		ET	0.224†	0.631†	0.719	0.216	0.672	0.768
	Post-Filter(LSI)	GT	0.227†	0.617	0.723	0.248†ρ	0.675	0.765
		ET	0.248†ρ	0.644†	0.743	0.233†ρ	0.691	0.785
	Retro(FrameNet)	GT	0.211†	0.607	0.707	0.209†	0.667	0.755
		ET	0.219†	0.633†	0.718	0.212†	0.668	0.768
	Retro(PPDB)	GT	0.205	0.597	0.682	0.226†	0.657	0.765
		ET	0.221†	0.636†	0.736	0.240†ρ	0.674	0.768
	Retro(WordNet.synonyms)	GT	0.207	0.618	0.711	0.231†	0.675	0.758
		ET	0.223†	0.625†	0.725	0.226†	0.668	0.757
	Retro(WordNet.synonyms+)	GT	0.206†	0.606	0.707	0.216†	0.662	0.755
		ET	0.212†	0.624	0.712	0.220†	0.661	0.768
	Retro(LSI)	GT	0.241†ρ	0.611	0.712	0.244†ρ	0.671	0.761
		ET	**0.254**†ρ	0.641†	0.743	0.227†	**0.698**	**0.812**†ρ
	Ext-Retro(LSI + PPDB)	GT	0.222	0.613	0.716	0.233†	0.671	0.768
		ET	0.248†	0.632	0.733	0.232†	0.672	0.775
	Post-Filter-Retro(PPDB, LSI)	GT	0.230†	0.614	0.723	0.244†ρ	0.672	0.765
		ET	0.254†	0.643†	0.743	0.240†ρ	0.690	0.788
	Retro(PPDB) + Retro(LSI)	GT	0.221	0.611	0.709	0.237†ρ	0.671	0.785
		ET	0.235†	0.639†	0.743	0.253†ρ	0.677	0.798

Table B.2: Results for the best parameter configurations for TREC-123 and TREC-Robust-04

	Method	T. Model	BM25 MAP	BM25 NDCG	BM25 P@10	LM MAP	LM NDCG	LM P@10
TREC-Robust-2004	STD		0.263	0.535	0.449	0.264	0.538	0.435
	w2v	GT	0.277†	0.548†	0.449	0.277†	0.551†	0.436
		ET	0.279†	0.551†	0.451	0.276†	0.549†	0.437
	Post-Filter(LSI)	GT	0.277†	0.548†	0.451	0.278†	0.551†	0.441
		ET	0.279†	0.551†	0.453	0.276†	0.549†	0.442
	Retro(FrameNet)	GT	0.267	0.536	0.449	0.269	0.539	0.436
		ET	0.266	0.537†	0.451	0.268	0.540	0.436
	Retro(PPDB)	GT	0.281†	0.551†	0.458	0.280†	0.553†	0.437
		ET	0.283†	0.554†	0.460†	0.279†	0.551†	0.438
	Retro(WordNet.synonyms)	GT	0.278†	0.546†	0.455	0.276†	0.547†	0.439
		ET	0.278†	0.549†	0.455	0.275†	0.546†	0.440†
	Retro(WordNet.synonyms+)	GT	0.272	0.543†	0.453	0.271	0.544	0.436
		ET	0.273	0.544†	0.454	0.271	0.544	0.436
	Retro(LSI)	GT	0.277†	0.548†	0.451	0.277†	0.551†	0.439
		ET	0.279†	0.551†	0.454	0.276†	0.549†	0.440
	Ext-Retro(LSI + PPDB)	GT	0.280†	0.550†	0.454	0.277†	0.551†	0.441
		ET	0.281†	0.552†	0.458	0.276†	0.549†	0.441
	Post-Filter-Retro(PPDB, LSI)	GT	0.282†	0.552†	0.458	0.281†	0.553†	0.441
		ET	**0.283**†	**0.554**†	**0.461**†	0.279†	0.551†	0.439
	Retro(PPDB) + Retro(LSI)	GT	0.278†	0.549†	0.457	0.277†	0.551†	0.441
		ET	0.279†	0.551†	0.459	0.276†	0.550†	0.440
TREC-123	STD		0.222	0.431	0.488	0.222	0.437	0.505
	w2v	GT	0.229	0.436	0.500	0.227	0.440	0.515
		ET	0.229	0.438	0.496	0.227	0.440	0.512
	Post-Filter(LSI)	GT	0.229	0.438†	0.500	0.229	0.441	0.515
		ET	0.229†	0.440†	0.499	0.227	0.442	0.515
	Retro(FrameNet)	GT	0.228	0.434	0.504	0.226†	0.439	0.517†
		ET	0.230	0.436	0.501	0.227	0.442	0.516†
	Retro(PPDB)	GT	0.232†	0.438†	0.507	0.231†	0.442†	0.515
		ET	0.234†	0.441	0.505	0.232†	0.443	0.514
	Retro(WordNet.synonyms)	GT	0.228	0.437	0.512	0.227	0.442	0.510
		ET	0.229	0.437	0.509	0.227	0.440	0.517
	Retro(WordNet.synonyms+)	GT	0.228†	0.438	0.502	0.227	0.441	0.520†
		ET	0.228	0.436	0.501	0.225	0.439	0.520†
	Retro(LSI)	GT	0.229†	0.440†	0.500	0.229†	0.443†	0.515
		ET	0.229	0.441†	0.503	0.228	0.442	0.514
	Ext-Retro(LSI + PPDB)	GT	0.232†	0.440†	0.505†	0.231	0.442	0.518†
		ET	0.230†	0.440†	0.505	0.228†	0.441	0.518†
	Post-Filter-Retro(PPDB, LSI)	GT	0.234†	0.442†	0.515†	0.233†	0.444†	0.520
		ET	**0.236**†	0.444†	0.509†	0.232†	0.444	0.519
	Retro(PPDB) + Retro(LSI)	GT	0.235†$\rho\nu$	0.442†	0.511†	0.234†$\rho\nu$	**0.445**	**0.521**
		ET	0.235†	0.443†	0.506	0.232†	0.444	0.518

List of Figures

List of Tables

Bibliography

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Communications of the ACM*, 51:117–122, 2008.

[BCZ+16] Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4356–4364, 2016.

[BFL98] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The Berkeley FrameNet Project. In *Proceedings of the 36th annual meeting on Association for Computational Linguistics*, pages 86–90, 1998.

[BGJM17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[CD15] Jiaqiang Chen and Gerard De Melo. Semantic Information Extraction for Improved Word Embeddings. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 168–175, 2015.

[DDF+90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[DMC16] Fernando Diaz, Bhaskar Mitra, and Nick Craswell. Query Expansion with Locally-Trained Word Embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 367–377, 2016.

[FDJ+15] Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting Word Vectors to Semantic Lexicons. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615, 2015.

[Fir57] J.R. Firth. A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis*, pages 1–32, 1957.

[GDNB10] Vincent Garcia, Eric Debreuve, Frank Nielsen, and Michel Barlaud. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. In *Proceedings of the 7th IEEE International Conference on Image Processing*, pages 3757–3760, 2010.

[GDR+15] Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, and Narayan Bhamidipati. Context- and Content-aware Embeddings for Query Rewriting in Sponsored Search. In *Proceedings of the 38th International ACM Conference on Research and Development in Information Retrieval*, pages 383–392, 2015.

[GFAC16] Jiafeng Guo, Yixing Fan, Qingyao Ai, and Bruce Croft. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pages 55–64, 2016.

[GVCB13] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. PPDB: The Paraphrase Database. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 758–764, 2013.

[JDH15] Sujay Kumar Jauhar, Chris Dyer, and Eduard Hovy. Ontologically Grounded Multi-sense Representation Learning for Semantic Vector Space Models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 683–693, 2015.

[LGD15] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, 2013.

[MDC17] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to Match using Local and Distributed Representations of Text for Web Search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1291–1299, 2017.

[Mil95] George A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[MNCC16] Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. A Dual Embedding Space Model for Document Ranking. *arXiv preprint arXiv:1602.01137*, 2016.

[MRSO08] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, and Others. *Introduction to information retrieval*. Cambridge University Press, 2008.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[NMCC16] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. Improving Document Ranking with Dual Word Embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 83–84, 2016.

[NSPM14] Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. Efficient Non-parametric Estimation of Multiple Embeddings per Word in Vector Space. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1059–1069, 2014.

[NTSS17] Gia Hung Nguyen, Lynda Tamine, Laure Soulier, and Nathalie Souf. Learning concept-driven document embeddings for medical information search. In *Proceedings of the Conference on Artificial Intelligence in Medicine*, 2017.

[PNI+18] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[RLH17a] Navid Rekabsaz, Mihai Lupu, and Allan Hanbury. Explicit Neural Word Representation - Case Study on Gender Bias in Wikipedia. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2017.

[RLH17b] Navid Rekabsaz, Mihai Lupu, and Allan Hanbury. Exploration of a Threshold for Similarity Based on Uncertainty in Word Embedding. In *Advances in Information Retrieval*, pages 396–409, 2017.

[RLHZ16] Navid Rekabsaz, Mihai Lupu, Allan Hanbury, and Guido Zuccon. Generalizing Translation Models in the Probabilistic Relevance Framework. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 711–720, 2016.

[RLHZ17] Navid Rekabsaz, Mihai Lupu, Allan Hanbury, and Hamed Zamani. Word embedding causes topic shifting; Exploit global context! In *Proceedings of the 40th International ACM Conference on Research and Development in Information Retrieval*, pages 1105–1108, 2017.

[SCH17] Robert Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *Proceedings of the Conference on Artificial Intelligence*, pages 4444–4451, 2017.

[WM17] Hsin-Yang Wang and Wei-Yun Ma. Integrating Semantic Knowledge into Lexical Embeddings Based on Information Content Measurement. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 509–515, 2017.

[ZC17] Hamed Zamani and Bruce Croft. Relevance-based Word Embedding. In *Proceedings of the 40th International ACM Conference on Research and Development in Information Retrieval*, pages 505–514, 2017.