Embedded Retrieval

Word Embeddings for Information Retrieval

Lukas Paul Achatius Galke

Master's Thesis
March 2017

Knowledge Discovery
Department of Computer Science
Kiel University

Advised by
Prof. Dr. Ansgar Scherp
Ahmed Saleh

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Abstract

We assess the suitability of word embeddings for practical information retrieval. While limiting ourselves to unsupervised models, we compare the performance of several techniques that leverage word embeddings to retrieval models (i. e., provide a query-document similarity): the intuitive word centroid similarity, dedicated paragraph vectors, the physically inspired Word Mover's distance, as well as a novel IDF re-weighted word centroid similarity.

In our comparison, we thrive to simulate a strictly practical setting: short queries, a boolean matching operation, only the first twenty retrieved documents are considered. We evaluate the performance using the ranking metrics mean average precision, mean reciprocal rank and normalised discounted cumulative gain. Additionally, we inspect the retrieval models' sensitivity to document length by using either only the title or the full-text as documents.

We conclude that word centroid similarity is the best competitor to state-of-the-art retrieval models and can be further improved by re-weighting the word frequencies according to inverse document frequency before aggregating the respective word vectors of the embedding. The proposed cosine similarity of IDF re-weighted word vectors is competitive to the TF-IDF baseline and even outperforms it in case of the news domain with a relative percentage of 15%.

In the context of this research contribution, a dedicated information retrieval framework has been developed. The key features include the incorporation of embedding-based retrieval models, the simulation of a practical setting, automatic evaluation as well as convenient extendability by new retrieval models. The corresponding user's guide and developer's guide are part of this work.

Acknowledgements

I thank Ansgar Scherp and Ahmed Saleh for their great advises and guidance before and during the writing of this thesis. Thanks also to Till Blume for hints on the structure and style as well as Florian Mai for valuable discussions about the content. I would also like to thank Bram Moolenaar for creating the wonderful text editor with which every single line of this work is written. Finally, I thank my parents, Sven and Lioba Galke, for raising (and maintaining) my interest in computer science and their steady support throughout my studies.

Preface

This thesis is divided into two components. A research contribution of 8 pages ACM double-column style (`sigconf` of acmart [1]) and a technical report describing the developed information retrieval framework. The technical report is in turn divided into a user's guide and a developer's guide. For reasons of style consistency, the research contribution is adapted to the Kiel Computer Science Series style (`ifiseries` of KCCS [2]) in the present thesis. The developed information retrieval framework is available on GitHub [3].

[1] https://www.acm.org/publications/proceedings-template
[2] https://www.informatik.uni-kiel.de/~discopt/kcss/index.html
[3] https://github.com/lgalke/vec4ir

Contents

List of Figures

List of Tables

Research

Chapter 1

Embedded retrieval

1.1 Introduction

Word embeddings have become the default representation for text in many neural network architectures and text processing pipelines [BDV+03; BCV13; Got16]. In contrast to the typical bag-of-words representations, word embeddings are capable of capturing semantic and syntactic relations between the words [MSC+13; PSM14]. So far, they have been successfully employed in various natural language processing tasks such as word analogies, clustering, and classification [MSC+13; PSM14; KSK+15; BA16]. Word embeddings are recognised as the main reason for natural language processing (NLP) breakout in the last few years [Got16].

A word embedding is a distributed vector representation for words [MSC+13]. Each word is represented by a low-dimensional (compared to the vocabulary size) dense vector, which is learned from raw text data. In several natural language processing architectures such as neural networks these representations serve as first layer for the conversion from raw tokens (words) to a more useful representation. The property that semantically related terms are clustered close to each other in the representation space proves the usefulness of this approach for classification and other NLP tasks. However, transferring the success of word embeddings to the ad-hoc Information Retrieval (IR) task is currently an active research topic. While embedding-based retrieval models could tackle the vocabulary mismatch problem by making use of the embedding's inherent similarity between distinct words, most of them struggle to compete with the prevalent strong baselines, namely

TF-IDF [SB88], Okapi BM25 [RWH+95] and their relatives.

The majority of practical information retrieval systems rely on an extended boolean model [SFW83; MRS08]. Extended boolean models generalise both standard boolean models and vector space models. These extended boolean models are highly efficient, since the documents can be stored in an inverted index [MRS08]. Thus, the IR system stays responsive even if a huge amount of documents is indexed. Those practical IR systems employ a binary matching operation on the inverted index to reduce the set of documents, to which the similarity of the query is computed (see Figure 1.1). However, some advanced techniques based on query expansion and relevance feedback can be safely incorporated in an extended boolean model.

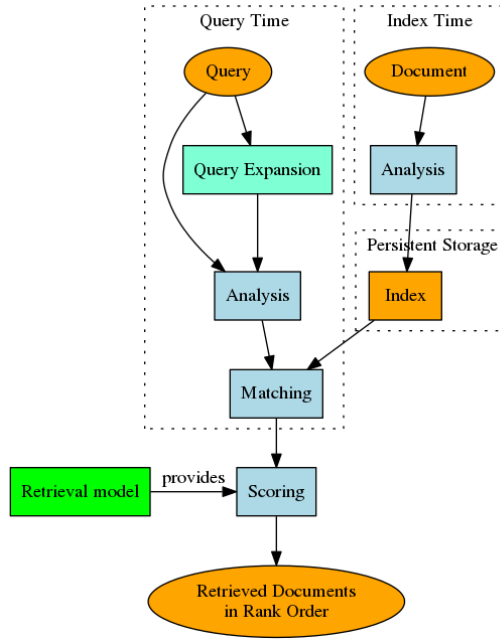

Figure 1.1. A simplified information retrieval system.

We consider a practical ad-hoc IR task which is composed of two core

steps: matching and scoring [MRS08]. In the matching step, documents of the corpus are matched against a query, typically by (binary) term co-occurrence: either the documents contain at least one term of the query or not (boolean `OR` query). In the scoring step, these matched documents are ranked according to their relevance to the query. As these core IR tasks are different from other NLP tasks, the incorporation of word embeddings is challenging. Since we evaluate the suitability of embedding-based retrieval models in a practical context, we fix the matching operation and concentrate on the similarity scoring operation. Additionally, we restrict ourselves to purely unsupervised models. Please note that every retrieval model could be potentially improved by query-relevance information. We also exclude pseudo-relevance feedback, since it is typically not applied in a practical IR setting [1].

In this paper, we compare and evaluate several similarity metrics for query-document pairs using word embeddings and assess their suitability in a practical IR setting. The considered approaches are word centroid similarity and a novel IDF re-weighted variant, Word Mover's distance [KSK+15] and paragraph vectors [LM14]. Practical IR systems allow treating the fields (`title`, `full-text`, `date`, ...) of a document differently. Thus, we analyse whether the performance of the embedding-based techniques depends on document length. In summary, we will answer the following research questions:

1. Which embedding-based techniques are suitable for practical information retrieval?

2. How does their performance depend on document length?

The remainder of this chapter is structured as follows: after providing some related work in Section 1.2, we describe the embedding algorithms and retrieval models in Section 1.3. In Section 1.4, we describe the conducted experiments in detail. The results and their discussion are provided in Section 1.5 and Section 1.6, respectively. Finally, we draw a conclusion in Section 1.7.

[1] Pseudo-relevance feedback is not natively included in Apache Lucene, thus SOLR and Elasticsearch

1.2 Related work

In this section, we give brief overview on related work from the information retrieval and word embedding fields.

Information Retrieval Extended boolean models such as TF-IDF [SB88] and Okapi BM25 [RWH+95] rely on bag-of-words representations, re-weighted by inverse document frequency. While still considered a strong baseline, these models (along with others) struggle to deal with two typical difficulties of the IR task: *term dependencies* and *vocabulary mismatch* [MRS08]. The former means the independence assumption of terms does not hold in natural language, the latter describes the problem of disregarding semantically related terms, when exact matching fails. Early approaches to tackle the term dependency problem involved word n-gram models. However, Fagan [Fag87] showed that these approaches are not so successful, most probably caused by higher sparsity of the more complex n-grams. [Zha08]. There are several probabilistic models that rely on language modelling. The documents are ranked either by each document language model's probability of generating the query or by the probability of generating the document, given the query language model [BBL99; PC98; MLS99; Hie98]. The divergence from randomness retrieval model was shown to outperform BM25 consistently on several TREC collections [AR02].

Topic Models The idea of distributed representations for documents goes back to latent semantic indexing by Furnas et al. [FDD+88]. It relies on a singular value decomposition of the term-document matrix. It was extended with a probabilistic variant by Hofmann [Hof99]. Finally, Blei, Ng, and Jordan [BNJ03] proposed the probabilistic topic model Latent Dirichlet Allocation (LDA) in 2003.

Word Embeddings Bengio, Ducharme, Vincent, and Janvin [BDV+03] first introduced a statistical language model based on neural networks,

so-called neural net language models. These language models form the basis for word embeddings learned by a neural network. Mikolov et al. [MSC+13] proposed a neural network based word embedding (Word2Vec), in which the representations are learned by training to reconstruct each word's context (skip-gram model). The success of the Word2Vec model relies on skip-gram training with negative sampling, an efficient training algorithm (not involving dense matrix multiplication). Beside other word embeddings [CW08; MYH09; TRB10], it is notable that a word embedding can also be computed by directly factorising the global co-occurrence matrix as done with GloVe [PSM14]. Le and Mikolov [LM14] further extend the Word2Vec approach by additionally modelling representations of whole documents by paragraph vectors (Doc2Vec). Their experiments indicate that these distributed representations are useful for information retrieval tasks. However, the evaluation task is to find one relevant document out of three (given 80% training data), which is not a classical ad-hoc query task.

Word Embeddings for IR Clinchant and Perronnin [CP13] proposed a method for aggregating word vectors with the Fisher kernel to a document level. The authors applied their approach in ad-hoc retrieval outperforming Latent Semantic Indexing, but not TF-IDF or divergence from randomness. Zheng and Callan [ZC15] learn to re-weight word embeddings using BM25 in a supervised context. Kusner, Sun, Kolkin, and Weinberger [KSK+15] proposed the Word Mover's distance, a similarity metric between documents based on word embeddings. Inspired by the Earth Mover's distance, the Word Mover's distances solves an optimisation problem for the minimum cost of transportation between the words of two documents. The cost of moving from a single word to another is the cosine distance of their respective word vectors. Recently, Zamani and Croft [ZC16] proposed embedding based query language models, a dedicated retrieval technique based on word embeddings which thrives to tackle the vocabulary mismatch problem by incorporating word embeddings into query language models. They propose two methods for embedding-based query expansion as well as a method for embedding-based pseudo-relevance feedback.

1.3 Embedding-based retrieval

In the following, we will sketch the two major algorithms for learning a word embedding: Word2Vec (Section 1.3.1) and GloVe (Section 1.3.2). Then, we will depict the application of embedding-based techniques for computing a query-document similarity by word centroid similarity (Section 1.3.4), IDF re-weighted word centroid similarity (Section 1.3.5), Word Mover's distance (Section 1.3.6) and paragraph vectors (Section 1.3.3). A graphical overview of the interaction of the techniques is given in Figure 1.2.

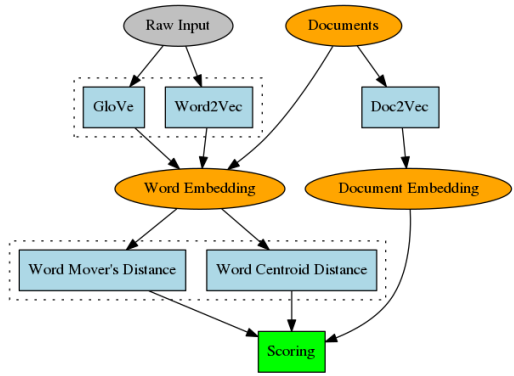

Figure 1.2. Overview of techniques for embedding-based retrieval

Notation We will use the following notation throughout the section:

$\boldsymbol{A}_i$ Row i of matrix $\boldsymbol{A}$ as a column vector (i. e. transposed row vector)

$\boldsymbol{X} \in \mathbb{R}^{n \times m}$ nBOW (L2-normalised bag of words) representation of n documents with m vocabulary words

$\boldsymbol{q} \in \mathbb{R}^m$ nBOW representation of the query

$\boldsymbol{W} \in \mathbb{R}^{m \times h}$ Word embedding consisting of m word vectors with h dimensions

$k \in \mathbb{N}$ number of documents to retrieve

1.3.1 Skip-gram negative sampling (Word2Vec)

Word2Vec [MSC+13] is an unsupervised algorithm learning a dense vector representation for each word from a corpus. One distinguishes between continuous bag of word (CBOW) and the skip-gram model. While the former aims to predict the centre word from its context, the latter aims to predict the context from the centre word. More formally, given a sequence of $w_1, \ldots, w_T$ words and a context window size, the training objective is to maximise:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leqslant j\leqslant c, j\neq 0}\log p(\boldsymbol{W}_{t+j} \mid \boldsymbol{W}_t)$$

The conditional probability $p(W_{t+j}|W_t)$ is typically given by the (hierarchical) softmax distribution. Training with negative sampling is an approximation for the full softmax. During training, negative samples are drawn from the vocabulary but do not appear in the actual context. A simplified algorithmic structure for skip-gram negative sampling algorithm can be defined as follows: Given a vocabulary $\mathbb{V} \subset \mathbb{N}$ and a stream of T words $\boldsymbol{w}$ with $\forall t < T : w_t \in \mathbb{V}$,

Algorithm 1: Skip-gram negative sampling algorithm

Data: Stream of words $\boldsymbol{w}$
Result: Word embedding $\boldsymbol{W}$

```
while t < T do
    Let w_t be target word with context
      C = {w_{t-c}, ..., w_{t-1}, w_{t+1}, ..., w_{t+c}};
    Look up word vector h := W_{w_T} for target word w_T;
    Predict with context words C (positive examples) along with
      negative examples sampled from V\C via logistic regression
      from word vector h;
    Update word vector h by back-propagation;
    T ← T + 1;
```

In the remainder of this work, we consider the skip-gram variant trained with negative sampling for Word2Vec.

1.3.2 Global word sectors (GloVe)

Global word vectors [PSM14] or GloVe is an algorithm to learn a word embedding by directly factorising the term co-occurrence matrix. More specific, the training objective (See Equation 1.3.1) is to learn word vectors $\boldsymbol{W}$ whose dot product equals the joint probability given by co-occurrence matrix $\boldsymbol{C}$.

$$\boldsymbol{W}_i^T \bar{\boldsymbol{W}}_k + b_i + \bar{b_k} \stackrel{!}{=} \log\left(1 + \boldsymbol{C}_{ik}\right) \tag{1.3.1}$$

The obtained word vectors can be used as basis for the document-level similarities: word centroid similarity and Word Mover's distance. When the global co-occurrence matrix is discarded (typically after initial learning of the word vectors), up-training of an existing model with GloVe is not possible.

1.3.3 Paragraph vectors (Doc2Vec)

The paragraph vector model (or Doc2Vec) [LM14] extends the Word2Vec model by a paragraph id as an additional input. A dense paragraph vector is learned for each paragraph (or document) in addition to the word vectors. We consider the distributed memory approach which models a paragraph identifier as if it was an artificial word token from the context. Given the query, A paragraph vector model is capable of inferring a paragraph vector for a new model by training for a few epochs without updating the weights. We employ the paragraph vector model to infer document vectors for each document. At query time we infer the document vector for the query. Finally, the matched documents are ranked by descending cosine similarity. For the inference step, we chose to train over five epochs with a learning rate decaying from 0.1 to 0.0001.

1.3.4 Word centroid similarity (WCS)

Given the term occurrence matrix $\boldsymbol{X}$, where X_{ij} is the number of occurrences of word j in document i, we compute the centroid of the document as follows. First, we normalise each row $\boldsymbol{X}_i$ to unit L2-norm (nBOW representation). The word j corresponds to the respective word vector in the embedding $\boldsymbol{W}_j$. Second, we obtain the word centroid representation of documents by matrix multiplication $\boldsymbol{C} = \boldsymbol{X} \cdot \boldsymbol{W}, \boldsymbol{C} \in \mathbb{R}^{n \times h}$. Now, the cosine similarity of the query to the centroids provides a notion of similarity:

$$\mathrm{WCS}(q, i) = \frac{\left(\boldsymbol{q}^T \cdot \boldsymbol{W}\right) \cdot \boldsymbol{C}_i}{||\boldsymbol{q}^T \cdot \boldsymbol{W}|| \cdot ||\boldsymbol{C}_i||}$$

The employed norm $||\cdot||$ is the L2-norm. Given a query, the documents are ranked by descending cosine similarity to the query. In case of length-normalised word frequency vectors, the resulting ranking of word centroid similarity is equivalent to the one of word centroid distance mentioned by Kusner, Sun, Kolkin, and Weinberger [KSK+15].

1.3.5 IDF re-weighted word centroid similarity (IWCS)

In addition, we propose a variant of the WCS, where the documents' bags of words are re-weighted by inverse document frequency as in TF-IDF, before the centroids are computed. Consider a bag-of-words representation $\boldsymbol{X}$ of the documents, where X_{ij} corresponds to the number of occurrences of word i in document j. We first re-weight $\boldsymbol{X}$ with respect to inverse document frequency:

$$X'_{ij} = X_{ij} \cdot \mathrm{idf}(j)$$
$$\mathrm{idf}(j) = \log \frac{1 + n}{1 + \mathrm{df}(D, j)}$$

The document frequency $\mathrm{df}(D, j)$ is the number of documents that contain word j. Then, we again normalise the rows of $\boldsymbol{X}$ to unit L2-norm and compute the centroids: $\boldsymbol{C} = \boldsymbol{X}' \cdot \boldsymbol{W}, \boldsymbol{C} \in \mathbb{R}^{n \times h}$. Finally, we compute the cosine similarity to the query and rank the results in descending order (as

in the WCS case).

1.3.6 Word Mover's distance (WMD)

The Word Mover's distance is a distance metric between two documents. The cumulative cost of moving the words of one document to another document is minimised. The cost function for moving from one word to another is defined as the euclidean distance between the word vectors $c(i,j) = ||\boldsymbol{W_i} - \boldsymbol{W_j}||_{\mathrm{L2}}$. The minimisation problem is constrained, such that all words of the source document and the destination document must be taken into account. The resulting transportation problem can be formalised as the following linear program, where T_{ij} denotes how much of the word i in the source document is moved to word j of the destination document [KSK+15]:

$$\min_{T \geqslant 0} \sum_{i,j=1}^{m} T_{ij} \cdot c(i,j)$$

$$\sum_{j=1}^{n} T_{ij} = d_i \forall i \in \{1, \ldots, m\}$$

$$\sum_{i=1}^{n} T_{ij} = d_j \forall j \in \{1, \ldots, m\}$$

It can be shown that word centroid distance as well as the relaxed Word Mover's distance (a variant of WMD that leaves out one of the constraints) is a lower bound for the Word Mover's distance. These lower bounds can be used to reduce the computational cost [KSK+15]. In addition to the full Word Mover's distance (WMD), we also evaluate a variant which takes the top k documents returned by IWCS and re-ranks them according to Word Mover's distance (IWCS-WMD).

1.4 Experimental setup

1.4.1 Task

Given a collection of documents D, a set of queries Q and relevance scores for each query-document pair $\mathcal{R} : Q \times D \to \mathbb{N}$ (the gold standard), the task is to return a ranked list of k (preferably) relevant documents. We evaluate these results according to $\mathcal{R}$. The values of $\mathcal{R}$ are restricted to binary $\{0, 1\} \subset \mathbb{N}$. Since we are interested in the performance of the retrieval models in a practical setting, we put the following constraints on the retrieval models:

- We perform a (disjunctive) boolean matching operation.
- We make no assumptions about the queries when indexing documents.

In this setting, we compare the performance of the embedding-based retrieval models with respect to the document's field `title`, `abstract`, `full-text` using either short or long queries. We evaluate the embedding-based retrieval models WCS, IWCS, WMD, IWCS-WMD and Doc2Vec inference for the embedding algorithms Word2Vec, GloVe and Doc2Vec.

1.4.2 Datasets

NTCIR2 The NTCIR2 dataset consists of 134,978 documents and 49 topics. The documents are composed of a `title` and an `abstract` field. The topics consist of the fields `title`, `description` and `narrative`. From these we use the `title` as *short* query and the `description` as *long* query. Additionally, two sets of relevance scores are provided that associate topics and documents (boolean). From these we chose the second set of relevance scores `rel2` with on average 43.6 (SD: 48.8) relevant documents per query. The relevance scores of the first set are always included in the second set. This results in a higher diversity for the ranking task. The relevancy judgements are not complete, i. e. there are query-document pairs for which no judgement is given. We assign these documents a relevancy of zero, when evaluating the models.

Economics The Economics dataset consists of 61,792 documents and 4,518 topics with an average of 72.98 (SD: 329) relevant documents per query. The documents are scientific publications from the economics domain. As topics, we use the concepts of a domain-specific thesaurus. A concept in the thesaurus consists of one preferred label and several alternative label. We employ the preferred labels of the concepts as queries. Each document of the collection is manually annotated (by domain experts) with a set of concepts. Hence, we consider a document being relevant to a topic, if and only if the document is annotated with the corresponding concept.

Reuters The Reuters dataset consists of 100,000 documents (random sample from Reuters RCV-1 [LYR+04]) and 102 topics from the news domain. The documents were manually annotated with one or more of the topics. On average, there are 3,143 (SD: 6,316) relevant documents per topic. Each document consists of a `title` and a `full-text` field. The descriptor label of a topic consist of two to three words. We employ these descriptor labels as query. The assignment of the label to the document resembles relevancy.

1.4.3 Embedding Models

Following the results of Mikolov et al. [MSC+13] and Kusner, Sun, Kolkin, and Weinberger [KSK+15], employing a well-trained general purpose embedding model is preferable over a corpus-specific model (caused by the surplus in diversity of contexts for each word during training). For this reason, and for the sake of a consistent comparison over the datasets, we employ pre-trained general-purpose word embeddings. Thus, the evaluation is not sensitive to the dataset and its specific training procedure (hyper-parameters are often sensitive to the training corpus). Although the absolute performance of the embedding-based techniques may be further increased by training a corpus-specific model, we assume that the relative performance of the different similarities would not differ as long as the model itself is trained properly. As representative for Word2Vec,

we employ the popular `GoogleNews` model. For GloVe we employ a similar model (vocabulary size of 2.2 billion, vectors of 300 dimensions, cased analysis), which is trained on the Common Crawl [2]. As treatment for out-of-vocabulary words, we found that ignoring them results in better over-all performance than initialising them with random vectors or up-training the missing words (See Appendix A.3).

Table 1.1. Abbreviation, corpus, word count, vocabulary size, dimensionality, analysis, training algorithm of the pre-trained models

Model	Corpus	Tokens	Vocab	Dim	Analysis	Training
W2V	GoogleNews	$100 \cdot 10^9$	$3 \cdot 10^6$	300	cased	Word2Vec
GLV	CommonCrawl	$840 \cdot 10^9$	$2.2 \cdot 10^6$	300	cased	GloVe
D2V	Wikipedia	$224 \cdot 10^9$	$3 \cdot 10^6$	1000	uncased	PV-DM

1.4.4 Preprocessing

Considering the analysis procedure of documents and queries, we use the same preprocessing steps for all retrieval models: First, we transform the raw string into lower case. Second, we tokenise the string by splitting it into words of at least two word-characters length, while treating any non-word character as delimiter. Finally, we remove common English stop words. To keep complexity under control, we do not apply stemming and only consider uni-gram models. Furthermore, we do **not** remove queries that contain out-of-vocabulary words. For a detailed overview on the resulting out-of-vocabulary statistics with respect to the consulted embedding model, we refer to Appendix A.2.

1.4.5 Evaluation

We consider three evaluation metrics: mean average precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative gain

[2] A dataset of crawled web data from https://commoncrawl.org/

(NDCG). For all metrics, we limit the considered documents to the top $k = 20$ retrieved documents. This reflects the typical user behaviour in a practical web search task.

Let D be the set of documents, Q the set of queries, and $\mathcal{R} : Q \times D \to N$ the relevance score of a document for a query. Then a retrieval model can be described as $M : Q \to D^k, q \mapsto y$ with $y \in D^k$ being the top-k retrieved documents in rank order. Thus the multi-set of results for queries Q and a retrieval model M can be written as:

$$R_{M,Q} = \left\{ (\mathcal{R}(q,d))_{d \in M(q)} \mid q \in Q \right\}$$

For a proper definition of the metrics, we operate on these sets $R_{M,Q}$.

Mean average precision (MAP) Precision is defined as the fraction of retrieved documents that are relevant and number of retrieved documents. The average precision (AP) is computed over the precision values, limited to the top $i = 1, \ldots, k$ retrieved documents. The mean of these average precision values is aggregated over the query set Q:

$$\begin{aligned} \text{Precision}(r,k) &= \frac{|\{r_i \in r \mid r_i > 0\}|}{|k|} \\ \text{AP}(r,k) &= \frac{1}{|r|} \sum_{i=1}^{k} \mathit{Precision}((r_1, \ldots, r_i), i) \\ \text{MAP}(R_{M,Q},k) &= \frac{1}{|Q|} \sum_{r \in R_{M,Q}} \text{AP}(r,k) \end{aligned}$$

Mean reciprocal rank (MRR) The reciprocal rank of a query's result is the fraction of the index of the first relevant document.

$$\text{MRR}(R_{M,Q},k) = \frac{1}{|Q|} \sum_{r \in R_{M,Q}} \frac{1}{\min\{i \mid r_i > 0\}}$$

In case none of the retrieved documents is relevant, the reciprocal rank is set to zero.

Normalised discounted cumulative gain (NDCG) We compute the NDCG for a single result list as follows:

$$\mathrm{DCG}(r,k) = r_1 + \sum_{i=2}^{k} \frac{r_i}{\log_2 i}$$

$$\mathrm{NDCG}_q(r,k) = \frac{\mathrm{DCG}(r,k)}{\mathrm{IDCG}_{q,\mathcal{R},k}}$$

where $\mathrm{IDCG}_{\mathrm{q},\mathcal{R},\mathrm{k}}$ is the best possible (ideal) DCG value for the specific query q with respect to the gold standard $\mathcal{R}$. In case there are more relevant documents than k, the IDCG is also computed on the truncated optimal results. Once again, we average NDCG over the queries, providing mean and standard deviation.

1.5 Results

NTCIR2 Considering the results for the NTCIR2 dataset, we inspect four configurations of either the `title` or the `abstract` field and either short (See Table 1.2) or long (See Table 1.3) queries. We observe that using the `title` field leads to better results in all metrics and for all techniques. The TF-IDF baseline yields better results in terms of MAP on the `title` field than on the `abstract` field (compare .35 to .46 MAP with short queries, and .35 to .40 MAP with long queries). In case of short queries, both variants of the Word Mover's distance (WMD, IWCS-WMD) perform consistently worse than IWCS as a query-document similarity. In case of long queries and the `title` field, the IWCS-WMD with the GloVe model attained the highest MAP value, .02 higher than the one of the baseline and 0.01 higher than the MAP of IWCS with the Word2Vec model. Still, in case of the `full-text` field, the MAP value of IWCS with the Word2Vec model (.36) is higher than the WMD re-ranked variants (.30 and .35 respectively). The TF-IDF baseline is outperformed by IWCS in terms of MAP in 3 out of 4 configurations. Still, the margin is rather small (ranging from .01 to .02). In terms of MRR, the baseline could only be outperformed in one configuration by IWCS with a difference of .01. The NDCG values of the baseline are not reached by any embedding-based retrieval model.

Table 1.2. Results for the NTCIR2 dataset using short queries and either the title or the full-text field with respect to the evaluation metrics mean average precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative gain (NDCG), limited to k=20 retrieved documents

Field	`title`			`full-text`		
Metric	MAP	MRR	NDCG	MAP	MRR	NDCG
TF-IDF	**.46** (.38)	**.55** (.45)	**.19** (.18)	.35 (.37)	.41 (.43)	**.18** (.20)
WCS_{GLV}	.37 (.36)	.42 (.42)	.16 (.18)	.29 (.31)	.40 (.43)	.15 (.17)
WCS_{W2V}	.33 (.34)	.35 (.38)	.14 (.16)	.33 (.35)	.39 (.43)	.13 (.15)
$IWCS_{GLV}$	.41 (.36)	.49 (.44)	.18 (.18)	.32 (.32)	.39 (.41)	.17 (.18)
$IWCS_{W2V}$	.38 (.35)	.45 (.43)	.17 (.18)	**.36** (.34)	**.42** (.41)	.17 (.18)
IWCS-WMD_{GLV}	.35 (.32)	.40 (.38)	.17 (.17)	.35 (.36)	.41 (.42)	.17 (.18)
IWCS-WMD_{W2V}	.30 (.31)	.34 (.37)	.15 (.17)	.29 (.32)	.33 (.39)	.15 (.17)
WMD_{GLV}	.25 (.33)	.27 (.37)	.11 (.17)	.18 (.27)	.21 (.33)	.08 (.14)
WMD_{W2V}	.27 (.35)	.29 (.40)	.11 (.16)	.22 (.29)	.24 (.34)	.10 (.14)
D2V	.27 (.32)	.33 (.39)	.13 (.16)	.29 (.34)	.35 (.42)	.13 (.16)

Table 1.3. Results for the NTCIR2 dataset using long queries and either the title or the full-text field with respect to the evaluation metrics mean average Precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative Gain (NDCG), limited to k=20 retrieved documents

Field	`title`			`abstract`		
Metric	MAP	MRR	NDCG	MAP	MRR	NDCG
TF-IDF	.40 (.29)	**.51**(.39)	**.20** (.15)	.35 (.32)	**.47** (.43)	**.20** (.21)
WCS_{GLV}	.29 (.29)	.38 (.41)	.15 (.16)	.27 (.26)	.35 (.37)	.14 (.14)
WCS_{W2V}	.30 (.26)	.38 (.38)	.15 (.15)	.30 (.32)	.37 (.41)	.13 (.14)
$IWCS_{GLV}$	.37 (.34)	.45 (.43)	.17 (.16)	.33 (.30)	.44 (.41)	.16 (.16)
$IWCS_{W2V}$	.41 (.35)	.50 (.41)	.19 (.15)	**.36** (.33)	**.47** (.43)	.17 (.16)
IWCS-WMD_{GLV}	**.42** (.36)	.50 (.44)	.17 (.14)	.30 (.30)	.37 (.38)	.17 (.18)
IWCS-WMD_{W2V}	.40 (.31)	**.51** (.41)	.18 (.14)	.35 (.34)	.40 (.41)	.16 (.16)
WMD_{GLV}	.10 (.22)	.12 (.26)	.04 (.08)	.12 (.21)	.14 (.25)	.06 (.10)
WMD_{W2V}	.22 (.33)	.25 (.39)	.08 (.11)	.30 (.32)	.37 (.41)	.13 (.14)
D2V	.24 (.31)	.27 (.37)	.11 (.16)	.16 (.25)	.19 (.31)	.08 (.11)

Economics For the Economics dataset (see Table 1.4), we observe that once again the retrieval over titles yields consistently higher metric values in terms of MAP, MRR and NDCG. Considering the `title` field, the IWCS

is similar to the baseline in terms of MAP (.37). The MRR and NDCG values attained by IWCS are slightly higher than the ones of WCS (.01). In case of `full-text`, no embedding-based technique could outperform the baseline. Doc2Vec inference is the closest competitor with .28 compared to .34 MAP of the baseline.

Table 1.4. Results for the Economics dataset using either the title or the full-text field with respect to the evaluation metrics mean average Precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative Gain (NDCG), limited to k=20 retrieved documents

Field		`title`			`full-text`	
Metric	MAP	MRR	NDCG	MAP	MRR	NDCG
TF-IDF	**.37** (.38)	.42 (.44)	.26 (.30)	**.34** (.35)	**.40** (.43)	**.26** (.30)
WCS_{GLV}	.36 (.37)	.42 (.44)	.25 (.29)	.21 (.29)	.25 (.36)	.13 (.19)
WCS_{W2V}	.36 (.37)	.41 (.43)	.25 (.29)	.26 (.31)	.32 (.40)	.19 (.24)
$IWCS_{GLV}$	**.37** (.37)	**.43** (.43)	.26 (.29)	.23 (.30)	.28 (.37)	.16 (.22)
$IWCS_{W2V}$	**.37** (.37)	**.43** (.43)	**.27** (.30)	.26 (.31)	.32 (.40)	.19 (.24)
$IWCS\text{-}WMD_{GLV}$	.33 (.35)	.38 (.41)	.25 (.28)	did not finish		
$IWCS\text{-}WMD_{W2V}$	.32 (.34)	.36 (.41)	.25 (.28)	did not finish		
WMD_{GLV}	.28 (.34)	.32 (.41)	.19 (.27)	did not finish		
WMD_{W2V}	.27 (.34)	.31 (.41)	.19 (.27)	did not finish		
D2V	.30 (.36)	.35 (.42)	.21 (.28)	.28 (.31)	.33 (.39)	.22 (.26)

Reuters Considering the results for the Reuters dataset (see Table 1.5), we observe that IWCS outperforms the baseline in case of the `title` as well as the `full-text` field. The IWCS attains a MAP of .60 compared to .52 of TF-IDF ($\approx$ 15% relative improvement). The results for the two embeddings Word2Vec and GloVe are more or less tied in all cases. In case of full-text with the Word2Vec model, re-weighting the top k documents with WMD could slightly improve the MAP (.56 compared to .55), while the NDCG is equal to one of IWCS and the MRR is slightly lower (.58 of TF-IDF compared to .60).

Table 1.5. Results for the Reuters dataset using either the title or the full-text field with respect to the evaluation metrics mean average Precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative Gain (NDCG), limited to k=20 retrieved documents

Field	`title`			`full-text`		
Metric	MAP	MRR	NDCG	MAP	MRR	NDCG
TF-IDF	.52 (.35)	.61 (.43)	.41 (.32)	.51 (.37)	.58 (.43)	.44 (.36)
WCS_{GLV}	.55 (.31)	.63 (.40)	.42 (.29)	.51 (.33)	.60 (.41)	.44 (.33)
WCS_{W2V}	.54 (.33)	.63 (.41)	.43 (.31)	.52 (.35)	.57 (.41)	.46 (.35)
$IWCS_{GLV}$	.58 (.31)	**.69** (.39)	.45 (.29)	.54 (.34)	**.63** (.41)	.47 (.33)
$IWCS_{W2V}$	**.60** (.33)	**.69** (.40)	**.47** (.32)	.55 (.35)	.60 (.41)	**.49** (.36)
$IWCS\text{-}WMD_{GLV}$	.54 (.30)	.62 (.39)	.43 (.49)	.55 (.34)	.61 (.41)	.46 (.33)
$IWCS\text{-}WMD_{W2V}$	.54 (.33)	.58 (.40)	.44 (.32)	**.56** (.37)	.58 (.42)	**.49** (.37)
WMD_{GLV}	.49 (.32)	.54 (.39)	.38 (.29)	.43 (.32)	.50 (.41)	.37 (.31)
WMD_{W2V}	.48 (.34)	.53 (.41)	.39 (.31)	.41 (.34)	.45 (.41)	.33 (.32)
D2V	.48 (.32)	.55 (.41)	.36 (.30)	.43 (.33)	.52 (.43)	.36 (.32)

1.6 Discussion

Comparing the `full-text` to the `title` field, we notice that the ranking quality (in terms of MAP, MRR and NDCG) is higher for the `title` field, consistently over all datasets and over all metrics (the only exception being the NDCG on the Reuters dataset). We can only infer that the surplus of information from the full-text does *not* help the scoring algorithms to produce a better ranking, but rather clutters the relevancy to the information need of the query.

Considering long and short queries, the results for the NTCIR2 dataset indicate that longer queries lead to higher metric values for all embedding-based retrieval models. We assume, that this is caused by the surplus of words carrying semantic information. In contrast, the MAP values of the baseline drop from .46 when using long queries to .40 when using short queries. Thus, we can exclude that the rise in performance is caused by the higher amount of matched documents.

For the embedding-based retrieval models, we observe that the newly proposed IWCS outperforms all other techniques. The only exception are

the `title` field of the Reuters dataset and the `title` field of the NTCIR2 dataset when using long queries. In those cases, the additional re-ranking of IWCS-WMD slightly improves the attained MAP by .01 in two cases and .05 in one case. Comparing WCS to IWCS, we observe that the aggregation of IDF re-weighted term-document vectors lead to better performance in all metrics over all datasets. This implies that it is valuable to reduce the impact of common words in the corpus, also in case of word embeddings. Comparing IWCS to the TF-IDF baseline, we observe that the two techniques yield similar performances in all metrics. The two techniques have in common that both re-weight the terms by inverse document frequency.

We furthermore investigate in which cases the IWCS has an advantage over TF-IDF. Consider a document containing a high amount of occurrences of the word `automobile` and query consisting of the term `car`. The document would be scored by TF-IDF relatively low since the term `car` does not occur frequently in the document. IWCS would score the document higher because the vector representations for `car` and `automobile` are close to each other in the embedding space (cosine similarity of .58 in the considered Word2Vec model). Especially on the Reuters dataset from the news domain, the IWCS outperforms the baseline with a relative percentage of 15% (MAP .60 compared to .52). We assume, that on the one hand the scientific datasets contain more domain-specific words that are not included in the embedding models. On the other hand, most of the words of the Reuters dataset from the news domain are contained in the word embedding model, since they are also trained on news data.

The embedding algorithms Word2Vec and GloVe produce similar results. One theoretical advantage of Word2Vec is that it can be up-trained. However, up-training did not increase the performance in our setting (See Appendix A.3). Still, up-training might be valuable in an incrementally growing corpus of documents as in a typical practical IR setting. Without up-training, new and possibly domain specific words that carry valuable semantic information, could never contribute to a document's representation. Please note that in contrast to GloVe, Word2Vec is capable of learning a word embedding iteratively (also from a base-model that is learned by GloVe). During the experiments, we gained the following valuable insights:

- ▻ Ignoring out-of-vocabulary words results in a higher performance than initialising them by random vectors. (See Appendix A.3).
- ▻ Up-training of missing word vectors with (frozen) original vectors does not improve retrieval performance of embedding-based techniques (up to 100 epochs with skip-gram negative sampling) (See Appendix A.3).
- ▻ All-But-The-Top embedding post-processing [MBV17] does not improve the performance of embedding-based retrieval models (See Appendix A.4).

Considering the WMD, we inspect the relative performance of IWCS and IWCS-WMD in detail, in order to gain insight on their relation. Please recall, that IWCS-WMD takes the top k documents returned by IWCS and re-ranks them with respect to Word Mover's distance. While IWCS-WMD is able to improve the result of IWCS in three (out of 16) cases, the performance of IWCS-WMD is in general lower than the one of IWCS. Therefore, we conclude that the WMD is not worth the additional computational effort in a time-sensitive practical IR setting. When comparing IWCD-WMD to the full Word Mover's distance (WMD), we notice a considerable drop in performance. The additional documents that are taken into account by the full WMD are not helpful for the ranking quality of the results. As the computational effort is even higher in this case, we cannot recommend the usage of unmodified WMD for practical IR.

Regarding Doc2Vec inference, we observe that in general more words on both the query and the document side, lead to a lower ranking quality. We assume that the semantics of the queries or documents are cluttered by numerous of too generic words. The only exception is the configuration of short queries and abstracts of the NTCIR2 dataset. In this case Doc2Vec was the only technique whose performance could be improved with using abstracts instead of titles. This behaviour can be explained by Doc2Vec's more sophisticated approach of creating a document representation by inference. However, Doc2Vec inference was not able to attain a ranking quality competitive to IWCS in our experiments.

As a limitation, we note that all embedding-based retrieval models do not tackle the complete vocabulary mismatch problem (the document

does not contain a single word of the query). Leaving out the matching operation decreases the performance of all embedding-based techniques considerably.

1.7 Conclusion

We confirm that word embeddings can be successfully employed in a practical information retrieval setting. The proposed cosine similarity of aggregated, IDF re-weighted word vectors is competitive to the TF-IDF baseline and even outperforms it in case of the news domain with a relative percentage of 15%.

Implementation

Chapter 2

User's guide

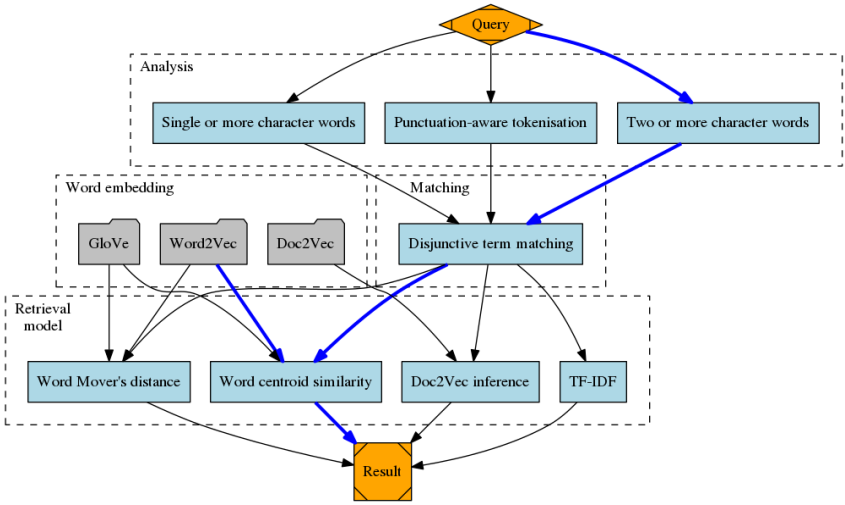

Figure 2.1. Detailed information retrieval pipeline as considered in the `vec4ir` framework. With the query node as starting point, each out-going edge is a configurable option of the native evaluation script. Rectangles resemble algorithms, and folder-like shapes resemble data stored on disk. One example configuration is highlighted by bold edges.

2.1 Philosophy

The information retrieval framework `vec4ir` is not designed for production usage. Rather, the target is to *simulate* a practical information retrieval setting. In order to encourage research in this field, the focus lies on extensibility. Adding a new retrieval model for evaluation should be as easy as possible. The target audience are researchers evaluating their own retrieval models and curious data scientists, who want to evaluate which retrieval model fits their data best.

2.2 Terminology

In the following, We recapture the most important definitions that are relevant to the framework.

2.2.1 Information Retrieval (IR)

The information retrieval task can be defined as follows:

Given a corpus of documents D and a query q, return $\{d \in D \mid d \text{ relevant to } q\}$ in rank order.

A practical information retrieval system typically consists of at least the two components: matching, similarity scoring. Several other components can also be considered, such as query expansion, pseudo-relevance feedback, query parsing and the analysis process itself.

2.2.2 Matching

The matching operation refers to the initial filtering process of documents. In its easiest way the output of the matching operation contains all the document which contain *at least one* of the query terms. Other variants of

the matching operation may also allow fuzzy matching (up to a certain threshold in Levenshtein distance) or query expansion.

2.2.3 Similarity scoring

The scoring step refers to the process of assigning scores. The scores resemble the relevance of a specific document to the query. They are used to create a ranked ordering of the matched documents. This sorting happens either ascending, when considering distance scores, or descending in case of similarity scores.

2.2.4 Word embeddings and Word2Vec

A word embedding is a distributed (dense) vector representation for each word of a vocabulary. It is capable of capturing semantic and syntactic properties of the input texts [MSC+13; PSM14]. Interestingly, even arithmetic operations on the word vectors are meaningful: e.g. '*King*' − '*Queen*' = '*Man*' − '*Woman*'. The two most popular approaches to learn a word embedding from raw text are:

- ▻ Skip-Gram Negative Sampling by Mikolov et al. [MSC+13] (Word2Vec)
- ▻ Global Word Vectors by Pennington, Socher, and Manning [PSM14] (GloVe)

Skip-gram negative sampling (or Word2Vec) is an algorithm based on a shallow neural network which aims to learn a word embedding. It is highly efficient, as it avoids dense matrix multiplication and does not require the full term co-occurrence matrix. Given some target word w_t, the intermediate goal is to train the neural network to predict the words in the c-neighbourhood of w_t: $w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}$. First, the word is directly associated to its respective vector, which as used as input for a (multinomial) logistic regression to predict the words in the c-neighbourhood. Then, the weights for the logistic regression are adjusted, as well as the vector itself (by back-propagation). The Word2Vec algorithm employs negative sampling: additional k noise words which do not appear

in the c-neighbourhood are introduced as possible outputs, for which the desired output is known to be `false`. Thus, the model does not reduce the weights to all other vocabulary words but only to those sampled k noise words. When these noise words appear in a similar context as w_t, the model gets more and more fine-grained over the training epochs. In contrast to word to Word2Vec, the GloVe [PSM14] algorithm computes the whole term co-occurrence matrix for a given corpus. To obtain word vectors, the term co-occurrence matrix is factorised. The training objective is that the euclidean dot product of each two word vectors match the log-probability of their words' co-occurrence.

2.2.5 Word centroid similarity (WCS)

An intuitive approach to employ word embeddings in information retrieval is the word centroid similarity (WCS). The representation for each document is the centroid of its respective word vectors. Since word vectors carry semantic information of the words, it is assumed that the centroid carries a notion of similarity. At query time, the centroid of the query's word vectors is computed and the cosine similarity to the centroids of the (matching) documents is used as a measure of relevance. When the initial word frequencies of the queries and documents are first re-weighted according to inverse-document frequency (i.e. frequent words in the corpus are discounted), the technique is labelled IDF re-weighted word centroid similarity (IWCS).

2.3 Features

The key features of this information retrieval framework are:

- ▻ Simulation of a practical IR setting
- ▻ Native word embedding support in conjunction with `gensim` [ŘS10])
- ▻ Built-in evaluation
- ▻ API design inspired by `sklearn` [BLB+13] .
- ▻ Extendable by new retrieval models

2.4 Dependencies

Besides `python3` itself, the package `vec4ir` depends on the following python packages.

- ▻ numpy [1]
- ▻ scipy [2]
- ▻ gensim [3]
- ▻ pandas [4]
- ▻ networkx [5]
- ▻ rdflib [6]
- ▻ nltk [7]
- ▻ sklearn [8]
- ▻ pyyaml [9]
- ▻ pyemd [10]

2.5 Installation

As `vec4ir` is packaged as a python package, it can be installed by via `python setuptools`:

```
cd python-vec4ir; python3 setup.py install
```

In case anything went wrong with the installation of dependencies, try to install them manually: We also recommend to install `numpy` and `scipy` in beforehand (either manually, or as binary system packages).

[1] http://www.numpy.org/
[2] http://www.scipy.org/
[3] http://radimrehurek.com/gensim/
[4] http://pandas.pydata.org/
[5] https://networkx.github.io/
[6] https://rdflib.readthedocs.io/en/stable/
[7] http://www.nltk.org/
[8] http://scikit-learn.org/stable/index.html
[9] http://pyyaml.org/wiki/PyYAMLDocumentation
[10] https://github.com/wmayner/pyemd

```
pip3 install -r requirements.txt
```

In addition, we provide a helper script to setup a new virtual environment. It can be invoked `setup.sh <venv-name>` to setup a new virtual environment in the current working directory. The newly created virtual environment has to be activated via `source <venv-name>` before performing the actual installation steps as described above.

2.6 The evaluation script

The package includes a native command line script `ir_eval.py` for evaluation of an information retrieval pipeline (See Figure 2.1). The pipeline of query expansion, matching and scoring is applied to a set of queries and the metrics mean average precision (MAP), mean reciprocal rank (MRR), normalised discounted cumulative gain (NDCG), precision and recall are computed. Hence, the script may be used *as-is* for evaluation of your datasets or as a reference implementation for the usage of the framework. The behaviour of the evaluation script and the resulting information retrieval pipeline can be controlled by the following command-line arguments:

Meta options

As a meta option (affecting other options), we allow the specification of a configuration file.

- `-c, --config` Path to a configuration file, in which the file paths for the datasets and embedding models are specified. The default value is `config.yml`.

General options

- `-d, --dataset` The data set to operate on. (mandatory)
- `-e, --embedding` The word embedding model to use. (mandatory)

- ▻ `-r, --retrieval-model` The retrieval model for similarity scoring. One of `tfidf` (default), `wcd`, `wmd`, `d2v`.
- ▻ `-q, --query-expansion` The expansion technique to apply on the queries. One of `wcd`, `eqe1`, `eqe2`.

The arguments `--dataset` and `--embedding` expect the provided keys to be present in the configuration file specified by `--config`. While the key for the embedding should be below `embeddings` in the configuration file, the key for the dataset should be below `data`. An minimal example structure would be:

```
embeddings:
    word2vec: # possible value for --embedding
        path: "/path/to/GoogleNews-vectors-negative300.bin.gz"
data:
    short-ntcir2: # possible value for --dataset
        type: "ntcir"
        root_path: "path/to/data/NTCIR2/"
        field: "title"
        topic: "title"
        rels: 2
```

The options below the respective keys specify the responsible constructor (`type`) and its arguments (`root_path`, `field`, ...). More details on the natively supported dataset formats can be found in the developer's guide. The alternatives for the retrieval model (`--retrieval-model`) are described in more detail in Section 2.8.

Embedding options

- ▻ `-n, --normalize` Normalise the embedding model's word vectors.
- ▻ `-a, --all-but-the-top` All-but-the-top embedding post-processing as proposed by Mu, Bhat, and Viswanath [MBV17] .
- ▻ `-t, --train` Number of epochs used for up-training out-of-vocabulary words.

In the special case of `--train 0` the behaviour of the script is *not* equivalent

to omitting this parameter. After building a vocabulary from the input documents, the word vectors are *intersected* with the word vectors of the embedding. More specific, missing vocabulary words are initialised with close-to-zero random vectors. Instead, out-of-vocabulary words are ignored, when the `--train` parameter is omitted.

Retrieval options

- `-I, --no-idf` Do *not* use IDF re-weighted word frequencies for aggregation of word vectors.
- `-w, --wmd` Fraction of *additional* documents to take into account for `wmd` retrieval model.

By default, `ir_eval.py` uses IDF re-weighted word centroid similarity if `-r wcd` is provided. If IDF re-weighting is not desired, it is necessary to provide the `--no-idf` argument. The `--wmd` argument refers to the fraction of additional documents to consider, when the Word Mover's distance (`-r wmd`) was chosen as retrieval model. For example, consider `--wmd 0.1` and 120 matching documents for a specific query. When 20 documents should be retrieved, the word centroid similarity is consulted to retrieve the top $20 + 0.1 \cdot (120 - 20) = 30$ documents. These 30 most relevant (with respect to word centroid similarity) documents are then be re-ranked according to the Word Mover's distance. A `--wmd` value of zero corresponds to re-ranking the top k documents by Word Mover's distance, while the highest possible value of `1.0` corresponds to computing the full Word Mover's distance without taking the WCS into account.

Output options

The output options control the online and persistent output of the evaluation script. The special option `--stats` can be provided to compute the ratio of out-of-vocabulary tokens, print the top 50 most frequent out-of-vocabulary tokens and exit.

- `-o, --output` File path for writing the output.

- `-v, --verbose` Verbosity level.
- `-s, --stats` Compute out-of-vocabulary statistics and exit.

Evaluation options

The following arguments affect the evaluation process:

- `-k` Number of documents to retrieve (defaults to 20).
- `-Q, --filter-queries` Drop queries, which contain out-of-vocabulary words.
- `-R, --replacement` Treatment for missing relevance information in the gold standard. Chose `drop` to disregard them (default) or `zero` to treat them as non-relevant.

In most cases, the defaults of *not* filtering the queries and replacing missing values by zeroes (indicating non-relevance) are appropriate.

Analysis options

The analysis process of splitting a string into tokens can be customised by the following arguments:

- `-M, --no-matching` Do *not* conduct a matching operation.
- `-T, --tokenizer` The tokeniser for the matching operation. One of `sklearn`, `sword`, `nltk`.
- `-S, --dont-stop` Do *not* remove English stop-words.
- `-C, --cased` Conduct a case *sensitive* analysis.

Please note that the analysis process, also directly affects the matching operation. The defaults are to conduct a matching operation, tokenise according to `sklearn` tokeniser [PVG+11] (a token consists of at least two subsequent word-characters), remove English stop-words and transform all characters to lower case. The `sword` tokeniser additionally considers single-character words as token. The `nltk` tokeniser retains all punctuation punctuation as tokens [Bir06].

2.7 Basic framework usage

We provide a minimal example including the matching operation and the TF-IDF retrieval model. As an example, assume we have two documents `fox valley` and `dog nest` and two queries `fox` and `dog`. First, we create an instance of the `Matching` class, whose optional arguments are passed directly to `sklearn`'s `CountVectorizer`.

```
>>> match_op = Matching()
```

In its default form, the matching is a non-fuzzy boolean `OR` matching. The `Matching` instance can be used after fitting a set of documents via `match_op.fit(documents)` via its `match_op.predict(query_string)` method to return the indices of the matching documents. However, the preferred way to apply the matching operation is inside of a `Retrieval` instance. The `Retrieval` instance expects a retrieval model as mandatory argument, besides an optional `Matching` instance (and an optional query expansion instance). For now, we create an instance of the `Tfidf` class and pass it to the `Retrieval` constructor.

```
>>> tfidf = Tfidf()
>>> retrieval = Retrieval(retrieval_model=tfidf, matching=match_op)
>>> retrieval.fit(titles)
>>> retrieval.query("fox")
[0]
>>> retrieval.query("dog")
[1]
```

At index time, the `Retrieval` instance invokes the `fit` method on the provided delegates of query expansion, matching and the retrieval model. At query time, the `Retrieval` instance consults the `predict` method of the `matching` argument for a set of matching indices. The matching indices are passed as `indices` keyword argument to the `query` method of the retrieval model. The result of the retrieval model is then transformed back into respective the document identifiers.

2.8 Employing word embeddings

Several supplied retrieval models make use of a word embedding. However those retrieval models do not learn a word embedding themselves, but take a `gensim` `Word2Vec` object as argument.

```
from gensim.models import Word2Vec
model = Word2Vec(documents['full-text'], min_count=1)
wcd = WordCentroidDistance(model)
RM = Retrieval(wcd, matching=match_op).fit(documents['full-text'])
```

Several embedding-based retrieval models are provided natively:

- ▻ *Word centroid similarity* The cosine similarity between centroid of the document's word vectors and the centroid of the query's word vectors (`WordCentroidDistance(idf=False)`).
- ▻ *IDF re-weighted word centroid similarity* The cosine similarity between the document centroid and the query centroid after re-weighting the terms by inverse document frequency (`WordCentroidDistance(idf=True)`).
- ▻ *Word Mover's distance* The cost of moving from the words of the query to the words of the documents is minimised. The optional `complete` parameter (between zero and one) allows to control the amount of considered documents. Setting `complete=0` results in re-ranking the documents returned by word centroid similarity with respect to Word Mover's Distance, while setting `complete=1.0` computes the Word Movers distance for all matched documents.
- ▻ *Doc2Vec inference* The `Doc2VecInference` class expects a `gensim` `Doc2Vec` instance as base model instead of a `Word2Vec` object. The model is employed to infer document vectors for the documents and the queries. The matching documents are ranked according to the cosine similarity between inferred vectors of the query and the documents. The parameters `alpha`, `min_alpha` and `epochs` control the inference step of the `Doc2Vec` instance.

All these provided embedding-based retrieval models are designed for usage inside the `Retrieval` wrapper.

2.9 Evaluating the model

To evaluate your retrieval model `RM` just invoke its `evaluate(X, Y)` method with `X` being a list of (`query_id`, `query_string`) pairs. The gold standard `Y` is a two-level data structure. The first level corresponds to the `query_id` and the second level to the `document_id`. Thus `Y[query_id][document_id]` should return the relevance number for the specific query-document pair. The exact type of `Y` is flexible, it can be a list of dictionaries, a 2-dimensional `numpy` array, or a `pandas.DataFrame` with a (hierarchical) multi-index.

```
scores = RM.evaluate(X, Y)
```

The `evaluate(X, Y, k=None)` method returns a dictionary of various computed metrics per query. The scores can be manually reduced to their mean afterwards with a dictionary comprehension:

```
import numpy as np
mean_scores = {k : np.mean(v) for k,v in scores.items()}
```

The metrics and therefore keys of the resulting scores dictionary consist of:

- ▷ *`MAP`* Mean Average Precision (`@k`)
- ▷ *`precision`* Precision (`@k`)
- ▷ *`recall`* Recall (`@k`)
- ▷ *`f1_score`* Harmonic mean of precision and recall
- ▷ *`ndcg`* Normalised discounted cumulative gain (produces sensible scores with respect to the gold standard, even if k is given)
- ▷ *`MRR`* Mean reciprocal rank (`@k`)

Where k is the number of documents to retrieve.

2.10 Extending by new models

In order to create a new retrieval model, one has to implement a class with at least two methods: `fit(X)` and `query(query, k=None, indices=None)`. In the following, we will demonstrate the implementation of a dummy retrieval model: the `GoldenRetriever` model which returns a random sample of k matched documents.

```
import random as RNG

class GoldenRetriever(RetriEvalMixin):
    def __init__(self, seed='bone'):
        # Configuration
        RNG.seed(seed)

    def fit(self, documents):
        # Keep track of the documents
        self._fit_X = np.asarray(documents)

    def query(self, query, k=None, indices=None):
        # Pre-selected matching documents
        if indices is not None:
            docs = self._fit_X[indices]
        else:
            docs = self._fit_X

        # Now we could do more magic with docs, or...
        ret = RNG.sample(range(docs.shape[0]), k)

        # Note that our result is assumed to be
        # relative to the matched indices
        # and NOT a subset of them
        return ret
```

The newly developed class contains the following three mandatory meth-

ods:

- *The constructor* All configuration is performed in the constructor, such that the model is ready to fit documents. No expensive computation is performed in the constructor.
- `fit(self, documents)` The `fit` method is called at index time, the retrieval model becomes aware of the documents and saves its internal representation of the documents.
- `query(self, query, k=None, indices=None)` The `query` method is called at query time. Beside the query string itself, it is expected to allow two keyword arguments: `k` resembling the number of desired documents, and `indices` which provides the indices of documents, that matched the query.

The reduction of the models own view on the documents (`docs = self._fit_X[indices]`) is important, since the returned indices are expected to be relative to this reduction (more details in the next section). These relative indices provide one key benefit. Oftentimes, the documents identifiers are not plain indices but string values. Using relative (to the matching) indices allows our retrieval model to disregard the fact that the document identifiers could be a string value or some other index, which does not match the position in our array of documents `X`. The presumably surrounding `Retrieval` class will keep track of the document identifiers for you and transpose the query's result `ret` to the identifier space.

The inheritance from `RetriEvalMixin` provides the `evaluation` method described above, which internally invokes the `query` method of the new retrieval model.

2.11 Matching, scoring, and query expansion

We provide a `Retrieval` class that implements the desired retrieval process of Figure 2.1. The `Retrieval` class consists of up to three components. It combines the retrieval model (mentioned above) as mandatory object and two optional objects: a matching operation and a query expansion object.

Upon invocation of `fit(X)` the `Retrieval` class delegates the documents `X` to all prevalent components, i.e. it calls `fit(X)` on the matching object, the query expansion object, and the retrieval model object. A query expansion class is expected to provide two methods.

- ▻ *`fit(X)`* This method is invoked with the set of raw documents `X`
- ▻ *`transform(q)`* This method is invoked with the query string `q` and should return the expanded query string.

We provide more details on the implementation of a full information retrieval pipeline in the Developer's Guide (See Chapter 3).

2.12 Combining multiple fields and models

The `vec4ir` package also provides an experimental operator overloading API for combining multiple retrieval models.

```
RM_title = WordCentroidDistance().fit(documents['title'])
RM_content = Tfidf().fit(documents['full-text'])
RM = RM_title & RM_content  # fuzzy and: x+y - x*y
R = Retrieval(retrieval_model=RM)
```

On invocation of the `query` method on the combined retrieval model `RM`, both the model for the title and the model for the content get consulted and their respective scores are merged according to the operator. Operator overloading is provided for addition, multiplication and binary `AND` operator which implements `FUZZY-AND` $x \& y = x + y - x \cdot y$. For these `Combined` retrieval models, the consulted operand retrieval models are expected to return (`doc_id`, `score`) pairs in their result set. However, in this case the result set does not have to be sorted. Thus, the query method of the operand retrieval models is invoked with `sorted=False`. Still, the combined retrieval model `RM` keeps track of its nesting, such that the outer-most `Combined` instance will return a sorted list of results.

Chapter 3

Developer's guide

The developer guide is targeted to people that want to extend or understand the functionality of the `vec4ir` package. We will go through the implementation details in a top-down manner (See Figure 3.1). We start with the core functionality (Section 3.1). Then, we describe the implementation of the matching operation (Section 3.2), provided retrieval models (Section 3.3), query expansion techniques (Section 3.4) and helpful utilities (Section 3.5) in detail. The API design of `vec4ir` is heavily inspired by the one of `sklearn` [BLB+13].

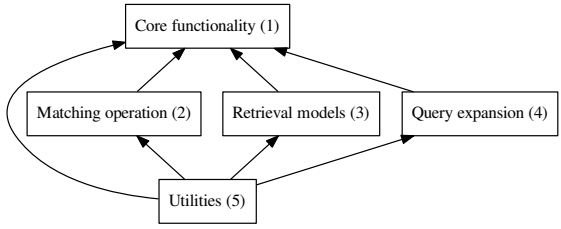

Figure 3.1. Structure of the developer guide. Arrows resemble an 'is used in'-relation.

3.1 Core functionality

The core functionality includes all functionality required for setting up a retrieval model and evaluating it. It consists of a generic information retrieval pipeline implementation, embedded vectorisation (i.e. word vector aggregation) along with the evaluation of retrieval models.

3.1.1 The retrieval class

The `Retrieval` class wraps the functionality of the matching operation, query expansion and similarity scoring.

Constructor

In the constructor, the references to the retrieval model, the matching operation and the query expansion instances are stored. Additionally, users may set an identifier name for their composition of retrieval models. The `labels_` property will be filled on invoking `fit`.

```
def __init__(self, retrieval_model, matching=None,
             query_expansion=None, name='RM'):
    BaseEstimator.__init__(self)

    self._retrieval_model = retrieval_model
    self._matching = matching
    self._query_expansion = query_expansion
    self.name = name
    self.labels_ = None
```

The fit method

Upon `fit(X[, y])`, the respective `Retrieval` instance (as a whole) fits the documents. The `fit` method is called on the delegates of retrieval model instance as well as the matching or query expansion instances (if

provided). If additional document identifiers are provided as argument `y` to `fit`, then these are stored inside the `labels_` property, such that in all further computation, the indices of `X` may be used for accessing the documents.

```
def fit(self, X, y=None):
    assert y is None or len(X) == len(y)
    self.labels_ = np.asarray(y) if y is not None else np.arange(len(X))
    if query_expansion:
        self._query_expansion.fit(X)

    if matching:
        self._matching.fit(X)

    self._retrieval_model.fit(X)
```

The query method

The `query(q[, k])` methods allows to query the `Retrieval` instance for `k` relevant documents to the query string `q`. First, in case a query expansion delegate is provided, it is called to transform the `q` according to the respective query expansion strategy. In a second step the matching operation is conducted (formally also optional) by invocation of its `predict` method. The `predict` method is expected to return a set of indices that matched the query `q`. The result of the matching operation `ind` is used in two ways. On the one hand, it is passed to the `query` method of the delegate of the retrieval model, such that it may (and should) reduce its internal representation according to the matching indices. On the other hand the view on the stored labels is reduced by `labels = labels[ind]`. The benefit of this reduction is that the relative indices `retrieved_indices`, returned by the retrieval model, can be used for accessing the labels array directly. Hence, the original document identifiers are restored. In between, we assert that the retrieval model does not return more than k documents by slicing `retrieved_indices = retrieved_indices[:k]` the top k indices.

```
def query(self, q, k=None):
    labels = self.labels_

    if self._query_expansion:
        q = self._query_expansion.transform(q)

    if self._matching:
        ind = self._matching.predict(q)
        if len(ind) == 0:
            return []
        labels = labels[ind]  # Reduce our own view
    else:
        ind = None

    retrieved_indices = self._retrieval_model.query(q, k=k, indices=ind)
    if k is not None:
        retrieved_indices = retrieved_indices[:k]

    return labels[retrieved_indices]
```

3.1.2 Embedded vectorisation

Since `vec4ir` is dedicated to embedding-based retrieval models, a class is provided that aggregates word vectors according to the respective term-document frequencies. We call this process "embedded vectorisation". The `EmbeddedVectorizer` extends the behaviour of `sklearn.feature_extration.text.TfidfVectorizer` as a subclass.

Constructor

In the constructor, the `index2word` property of the word vectors `embedding` is used to initialise the vocabulary of its super class `TfidfVectorizer`. This results in the crucial property that the indices of the embedding model and the transformed term-document matrix of the `TfidfVectorizer` correspond

to each other. All additional arguments of the constructor are passed directly to the `TfidfVectorizer` so that its functionality is completely retained. Most notably, passing `use_idf=False` disables the discounting of frequent terms over the collection (which is enabled by default). If `normalize=True` is given, the word frequencies are normalised to unit L2-norm for each document.

```
def __init__(self, embedding, **kwargs):
    vocabulary = embedding.index2word
    self.embedding = embedding
    TfidfVectorizer.__init__(self, vocabulary=vocabulary, **kwargs)
```

Fitting and transforming

The fit method call is passed to the delegate of the super class `TfidfVectorizer`.

```
def fit(self, raw_docs, y=None):
    super().fit(raw_docs)
    return self
```

In the `transform(raw_documents, y=None)` method, we also start with the invocation of the `transform` method of the superclass. The obtained `Xt` are the term frequencies of the documents. Depending on the parameters passed to the `TfidfRetrieval` super class in the constructor, the values of `Xt` are already re-weighted and normalised to unit L2-norm. In the final step we aggregate the word vectors with a single matrix multiplication `Xt @ syn0`. Thus, word frequency with unit L2-norm will result in the centroid of the respective embedding word vectors.

```
def transform(self, raw_documents, y=None):
    Xt = super().transform(raw_documents)
    # Xt is sparse matrix of word frequencies
    syn0 = self.embedding.syn0
    # syn0 are the word vectors
    return (Xt @ syn0)
```

3.1.3 Evaluation

For convenient evaluation, we provide the `RetriEvalMixin` class. A retrieval model class, that implements a `query(q[, k])` method, may inherit the provided `evaluate` method from the `RetriEvalMixin` class. The `evaluate` method is an interface for executing a set of queries by a single call and computing various ranking metrics mean average precision, mean reciprocal rank, normalised discounted cumulative gain, precision, recall, F1-score, precision@5, precision@10 as well as the respond time in seconds. The `evaluate` method expects the following arguments:

- *`X`* list of query identifier and query string pairs
- *`Y`* the gold standard, either a dictionary of dictionaries, a `pandas.DataFrame` with hierarchical index or an `numpy` / `scipy.sparse` array-like. The outer index is expected to be the query identifiers while the inner index needs to correspond to the document identifiers. A value greater than zero indicates relevance. Greater values indicate more relevance.
- *`k`* The desired amount of documents to retrieve. Except for precision@5 and precision@10, all metrics are computed with respect to this parameter.
- *`verbose`* Controls whether intermediate results should be printed to `stdout`.
- *`replacement`* The value to use for documents which could not be found in `Y` for the specific query identifier. The default value is zero. Hence missing query-document pairs are regarded non-relevant.
- *`n_jobs`* If greater than 1, the evaluation will be executed in parallel (not supported by all retrieval models).

The evaluate method of the `RetriEvalMixin` class is implemented as follows:

```
def evaluate(self, X, Y, k=20, verbose=0, replacement=0, n_jobs=1):
    # ... multi-processing code ...
    values = defaultdict(list)
```

```
    for qid, query in X:
        t0 = timer()
        result = self.query(query, k=k)
        values["time_per_query"].append(timer() - t0)
        scored_result = [harvest(Y, qid, docid, replacement)
                         for docid in result]
        r = scored_result[:k] if k is not None else scored_result

        # NDCG
        gold = harvest(Y, qid)
        idcg = rm.dcg_at_k(gold[argtopk(gold, k)], k)
        ndcg = rm.dcg_at_k(scored_result, k) / idcg
        values["ndcg"].append(ndcg)

        # MAP@k
        ap = rm.average_precision(r)
        values["MAP"].append(ap)

        # MRR
        ind = np.asarray(r).nonzero()[0]
        mrr = (1. / (ind[0] + 1)) if ind.size else 0.
        values["MRR"].append(mrr)

        # ... other metrics ... #
    return values
```

3.2 Matching operation

The matching operation is implemented in the `Matching` class and designed to be employed as a component of the `Retrieval` class. The interface to implement for a custom matching operation is:

- In the `fit(raw_documents)` method, an internal representation of the documents is stored.
- The `predict(query)` returns the index set of matching documents.

3.2.1 A generic matching class

The generic `Matching` class reduces the complexity of adding a matching algorithm to the implementation a single function. The default representation is a binary term-document matrix which should suffice for all boolean models. The representation is computed by an `sklearn.feature_extraction.text.CountVectorizer`. This process can be further customised by users, since additional keyword arguments are passed directly to the `CountVectorizer`.

```
def __init__(self, match_fn=TermMatch, binary=True, dtype=np.bool_,
             **cv_params):
    self._match_fn = match_fn
    self._vect = CountVectorizer(binary=binary, dtype=dtype,
                                 **cv_params)
```

Upon calling `predict`, the matching function `match_fn` is employed to compute the index set of the matching documents. Hence, the user can customise the behaviour (conjunctive or disjunctive) of each `Matching` instance by passing a function to the constructor. The `match_fn` function object is expected to return the matching indices, when provided with the term-document matrix `X` and the query string: `match_fn(X, q)` → `matching_indices`.

3.2.2 Disjunctive Matching

Disjunctive (`OR`) matching of single query terms is the default query parsing method for most information retrieval systems. Thus, the framework provides a dedicated function for it, which is also the default value of the `match_fn` argument for the constructor of the `Matching` class.

```
def TermMatch(X, q):
    inverted_index = X.transpose()
    query_terms = q.nonzero()[1]
    matching_terms = inverted_index[query_terms, :]
    matching_doc_indices = np.unique(matching_terms.nonzero()[1])
```

```
    return matching_doc_indices
```

First, the index is inverted and the indices of the query tokens are extracted by finding the `np.nonzero` entries in the second dimension `[1]`. Second, we look up the indices of the query tokens in the inverted index. Finally, we only keep one index for each the matching document by calling `np.unique`.

3.3 Retrieval models

3.3.1 TF-IDF

The `Tfidf` class implements the popular retrieval model based on TF-IDF re-weighting introduced by Salton and Buckley [SB88]. The term frequencies of a document are scaled by the inverse document frequency of the specific terms in the corpus D [PVG+11]:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

The $tf(t, d)$ is the number of occurrences of the word t in the document d:

$$tf(t, d) = \text{freq}(t, d)$$

The inverse document frequency is a measure for the fraction of documents that contain some term t:

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

The `Tfidf` retrieval model extends the `TfidfVectorizer` of `sklearn` the `Tfidf`. The documents are stored as an L2-normalised matrix of IDF re-weighted word frequencies. After transforming the query in the same way, we can compute the cosine similarity to all matching documents. As both the query and the documents are L2-normalised, the linear kernel yields the desired cosine similarity. Finally we use the function `argtopk` (See Section 3.5.1) to retrieve the top-k documents with respect to the result of the linear kernel.

```
class Tfidf(TfidfVectorizer):
    def __init__(self, analyzer='word', use_idf=True):
```

```
        TfidfVectorizer.__init__(self,
                                 analyzer=analyzer,
                                 use_idf=use_idf,
                                 norm='l2')
        self._fit_X = None

    def fit(self, X):
        Xt = super().fit_transform(X)
        self._fit_X = Xt
        return self

    def query(self, query, k=None, indices=None):
        if self._fit_X is None:
            raise NotFittedError
        q = super().transform([query])
        if indices is not None:
            fit_X = self._fit_X[indices]
        else:
            fit_X = self._fit_X
        # both fit_X and q are l2-normalised
        D = linear_kernel(q, fit_X)
        ind = argtopk(D[0], k)
        return ind
```

3.3.2 Word centroid similarity

The word centroid similarity aggregates the word vectors of the documents to their centroids. It is implemented in the `WordCentroidSimilarity` class. The implementation makes extensive use of the `EmbeddedVectorizer` (See Section 3.1.2) class for aggregation of word vectors. Hence, it is possible to choose between using IDF re-weighted word frequencies or plain word frequencies for aggregation. The parameters passed to the `EmbeddedVectorizer` delegate could be further extended. The current implementation limits the possible configuration to the single parameter `use_idf`. Providing `use_idf=True` results in IDF re-weighted word centroid

similarity.

```
class WordCentroidSimilarity(BaseEstimator):
    def __init__(self, embedding, analyzer='word', use_idf=True):
        self.vect = EmbeddedVectorizer(embedding,
                                       analyzer=analyzer,
                                       use_idf=use_idf)
        self.centroids = None

    def fit(self, X):
        Xt = self.vect.fit_transform(X)
        Xt = normalize(Xt, copy=False)
        self.centroids = Xt

    def query(self, query, k=None, indices=None):
        centroids = self.centroids
        if centroids is None:
            raise NotFittedError
        if indices is not None:
            centroids = centroids[indices]
        q = self.vect.transform([query])
        q = normalize(q, copy=False)
        # l2 normalised, so linear kernel
        D = linear_kernel(q, centroids)
        ret = argtopk(D[0], k=k)
        return ret
```

Please note, that the aggregated centroids need to be re-normalised to unit L2-norm (even if the word frequencies were normalised in the first place), so that the `linear_kernel` corresponds the cosine similarity. Finally, the results of the linear kernel are ranked by the `argtopk` (See Section 3.5.1) function.

3.3.3 Word Mover's distance

In order to compute the Word Mover's distance [KSK+15], we employ the `gensim` implementation `Word2Vec.wmdistance` [ŘS10]. Additionally, we make the analysis function `analyze_fn` as well as a completeness parameter `complete` available as arguments to the constructor. The `analyze_fn` argument is expected to be a function that returns a list of tokens, given a string. The parameter `complete` allows specification whether the full Word Mover's distance to all documents or only to the top k documents returned by `WordCentroidSimilarity` should be computed. The `WordCentroidSimilarity` is in turn customisable via the `use_idf` constructor argument. The `complete` parameter is expected to be a float value between 1 (compute the Word Mover's distance to all documents) and 0 (only compute the Word Mover's distance to the documents returned by WCS). A fraction in between takes the corresponding amount of additional documents into account. The crucial functionality can be summarised in the following lines of code.

```
incomplete = complete < 1.0

# inside fit method
docs = np.array([self.analyze_fn(doc) for doc in raw_docs])
if incomplete:
    self.wcd.fit(raw_docs)

# inside query method
if incomplete:
    wcd_ind = self.wcd.query(query, k=n_req, indices=indices)
    docs = docs[wcd_ind]
q = self.analyze_fn(query)
dists = np.array([self.embedding.wmdistance(q, d) for d in docs])
ind = np.argsort(dists)[:k]
if incomplete:
    # stay relative to the matching indices
    ind = wcd_ind[ind]
```

3.3.4 Doc2Vec inference

The `Doc2VecInference` class implements a retrieval model based on a paragraph vector model [LM14], i.e., `Doc2Vec` from `gensim` [ŘS10]. Given an existing `Doc2Vec` model, it is used to infer the document vector of a new document. We store these inferred document vectors and also infer a document vector for the query at query time. Afterwards, we compute the cosine similarity of all matching documents to the query and return the respective indices in descending order. The inference step considers three hyper-parameters:

▻ *`alpha`* The initial learning rate

▻ *`min_alpha`* The final learning rate

▻ *`steps`* The number of training epochs

The inference process itself (provided by gensim) runs `steps` training epochs with fixed weights and a linearly decaying learning rate from `alpha` to `min_alpha`. As the model does operate on a list of tokens, an analyser is required to split the documents into tokens. We compute the representation of the documents (as well as the query) as follows:

```
analyzed_docs = (analyzed(doc) for doc in docs)
representation = [doc2vec.infer_vector(doc,
                                       steps=self.epochs,
                                       alpha=self.alpha,
                                       min_alpha=self.min_alpha)
                  for doc in analyzed_docs]
representation = normalize(np.asarray(representation), copy=False)
```

The final normalisation step prepares the computation of the cosine similarity with a linear kernel. At query time, the cosine similarity between the vectors of the query and the documents is computed. The documents are then ranked in descending order.

3.4 Query expansion

All query expansion techniques should implement the following interface:

- ▻ *fit(raw_documents)* Fits the query expansion technique to the raw documents.
- ▻ *tranform(query)* Expands the query string.

As two implementations of this interface, we present the naive centroid based expansion as well as embedding-based query language models by Zamani and Croft [ZC16] . Please note, that for research it is strongly recommended to reduce the employed word embedding to the tokens that actually appear in the collection. Otherwise, words could be added to the query, that never appear in the collection and thus, do not affect the matching operation.

3.4.1 Centroid Expansion

Centroid expansion is a query expansion technique that computes the (possibly IDF re-weighted) centroid `v` of the query. It expands the query by the m nearest words with respect to the cosine distance to `v`. As a first step, we once again compute the centroid vector of the query using the `EmbeddedVectorizer` after fitting the collection (to build a vocabulary). Then, we employ the `similar_by_vector` method provided by the `gensim.models.Word2Vec` class to obtain the nearest tokens.

```
## inside transform method
v = vect.transform([query])[0]
exp_tuples = wv.similar_by_vector(v, topn=self.m)
words, __scores = zip(*exp_tuples)
expanded_query = query + ' ' + ' '.join(words)
```

3.4.2 Embedding-based query language models

The embedding-based query language models proposed by Zamani and Croft [ZC16] includes two techniques for query expansion (`EQE1` and `EQE2`), both rely on the underlying probability distribution based on the (weighted) sigmoid of the cosine similarity between terms. While `EQE1` assumes statistical independence of the query terms, `EQE2` assumes query-independent term similarities. The probabilities of the respective query language models are defined with respect to a `delta` function. The `delta` function transforms the cosine distance between two word vectors by a parametrised sigmoid function. We implement this behaviour in a dedicated `delta` function:

```
def delta(X, Y=None, n_jobs=-1, a=1, c=0):
    D = pairwise_distances(X, Y, metric="cosine", n_jobs=n_jobs)
    D -= c
    D *= a
    D = expit(D)
    return D
```

We compute the pairwise distances for all word vectors `D = delta(E,E)`. While both variants are implemented in the framework, we only present the computation of the `EQE1` variant:

```
prior = np.sum(D, axis=1)
conditional = D[q] / prior
posterior = prior * np.product(conditional, axis=0)
topm = np.argpartition(posterior, -m)[-m:]
expansion = [wv.index2word[i] for i in topm]
```

3.5 Utilities

3.5.1 Sorting only the top k documents

When dealing with retrieval models, a common operation is to retrieve the top k indices from a list of scores in sorted order. Sorting the complete

list and then slicing the top k documents would lead to unnecessary computation. Thus, `vec4ir` includes a useful helper function for this specific operation: `argtopk`. The function makes extensive of `numpy.argpartion`, which performs one step of the quicksort algorithm. On invocation by `np.argpartition(A, k)`, the element at position `k` of the returned index set is at the correct (with respect to sorted order) position. All indices $i = 0, \ldots, k-1$ smaller than `k` also have smaller values `A[i] <= A[k]`. Since we are interested in the k highest values, we invoke `argpartition` with `-k` and slice the top `k` values by `[-k:]`.

```
def argtopk(A, k=None):
    A = np.asarray(A)
    if k is None or k >= A.size:
        # if A contains too few elements or k is None,
        # return all indices in sort order
        return np.argsort(A, axis=axis)[::-1]

    ind = np.argpartition(A, -k, axis=-1)[-k:]
    ind = ind[np.argsort(A[ind], axis=-1)][::-1]
    return ind
```

3.5.2 Harvesting the gold standard

The `RetriEvalMixin` (as described in Section 3.1.3) allows several different data types for the gold standard `Y`. We provide a unifying function `harvest` to extract the relevance score for the desired respective query-document pair for all of the supported data types. The immediate benefit is that the data neither has to be transformed nor copied before calling `evaluate` on the implementing class. Additionally, the function can be used to obtain the whole set of values for one specific query (`docid=None`). On the one hand, the desired behaviour is to raise an exception, if a query identifier is not found in the gold standard. On the other hand, missing values for the document identifier for one specific query should not raise an exception but return the `default` value (typically zero). The latter behaviour can also be achieved by using a `defaultdict` or a sparse matrix as inner data

structure. However, we want to relief the user from those issues. The `harvest` function takes the following arguments:

- *`source`* The gold standard of query-document relevance, a two-level data structure with type either `DataFrame`, `dict` of `dicts` or two-dimensional `ndarray`,
- *`query_id`* The query identifier to use as index for the first level of `source`.
- *`doc_id`* If `None` return a list of relevance scores for the query given by `query_id`, else look up the `doc_id` in the second level of `source`.
- *`default`* The default value in case a document identifier is not found on the second level (typically zero).

In the implementation, we first access the `query_id` of the `source` and raise an exception if `query_id` is not prevalent in `source`. In a second step, we look up `doc_id` on the second level of the data structure and return the value. If the second step fails, the `default` value is returned instead.

3.5.3 Datasets

An interface for datasets

We introduce `IRDataSetBase` as an abstract base class for the data sets. Subclasses should implement the following properties:

- `docs` : Returns the documents of the corpus.
- `topics` : Returns the topics, i.e. the queries.
- `rels` : Returns the relevance judgements for the queries either as a `dict` of `defaultdicts` or as a `pandas.Series` object with a hierarchical index (multi-index).

All other options for the data set (such as the path to its root directory and caching) should be placed in the respective constructor. As an example subclass of the `IRDataSetBase`, we present the Quadflor-like dataset format.

Quadflor-like Dataset Format

For convenience we adopt the data set format and specification from Quadflor[1]. Quadflor is a framework for multi-label classification. A Quadflor-like data set consists of

- `X` The documents, either a directory of files with the base filename resembles the idea
- `y` The gold standard: label annotations for the documents. CSV file with document id in the first column, and subsequent columns resemble label identifier.
- `thes` A thesaurus consisting of a hierarchy of concepts.

In our prior work on Quadflor, the gold standard `y` was used as target labels in a multi-label classification task. We employ the data set in a different manner. We extract the preferred labels of each concept from the thesaurus `thes` and use them as queries. When the label annotations (`y`) for some document contain the specific concept identifier, we consider the document as relevant to the query.

[1] https://github.com/quadflor/Quadflor

Summary

In summary, a comparative evaluation of embedding-based document representations and query-document similarities for practical information retrieval has been conducted. A novel technique for the aggregation of word vectors via IDF re-weighted word frequencies has been proposed. The associated IDF re-weighted word centroid similarity is competitive to the TF-IDF baseline and even outperforms it in case of the news domain with a relative percentage of 15%.

For the evaluation, an information retrieval framework that simulates a practical information retrieval setting has been developed. As the framework is designed for researchers, it provides convenient evaluation is extendable by new techniques. In addition, a possible integration of the word centroid similarity into an existing information retrieval framework is provided in Appendix B.

Appendix A

Extended results

A.1 Effect of word vector normalisation

We investigate the influence of normalising the word vectors to unit length, prior to their aggregation.

Table A.1. The effect of word vector normalisation on word centroid similarity for the NTCIR2 dataset using short queries and either the title or the full-text field with respect to the evaluation metrics mean average precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative gain (NDCG), limited to k=20 retrieved documents

Field		`title`			`full-text`	
Metric	MAP	MRR	NDCG	MAP	MRR	NDCG
TF-IDF	.46 (.38)	.55 (.45)	.19 (.18)	.35 (.37)	.41 (.43)	.18 (.20)
normalised word vectors						
WCS_{GLV}	.37 (.36)	.42 (.42)	.16 (.18)	.29 (.31)	.40 (.43)	.15 (.17)
WCS_{W2V}	.33 (.34)	.35 (.38)	.14 (.16)	.33 (.35)	.39 (.43)	.13 (.15)
$IWCS_{GLV}$	.41 (.36)	.49 (.44)	.18 (.18)	.32 (.32)	.39 (.41)	.17 (.18)
$IWCS_{W2V}$	.38 (.35)	.45 (.43)	.17 (.18)	.36 (.34)	.42 (.41)	.17 (.18)
non-normalised word vectors						
WCS_{GLV}	.36 (.36)	.43 (.42)	.16 (.18)	.28 (.29)	.37 (.41)	.15 (.17)
WCS_{W2V}	.30 (.35)	.35 (.41)	.14 (.17)	.33 (.33)	.39 (.40)	.15 (.17)
$IWCS_{GLV}$	.39 (.36)	.46 (.43)	.18 (.17)	.31 (.32)	.37 (.40)	.17 (.18)
$IWCS_{W2V}$	.33 (.34)	.39 (.41)	.15 (.17)	.33 (.32)	.40 (.40)	.16 (.17)

Regarding the results for the NTCIR2 dataset with short queries (See Table A.1), we observe, that the MAP values are consistently higher in case of normalised word vectors.

Table A.2. The effect of word vector normalisation on word centroid similarity for the NTCIR2 dataset using long queries and either the title or the full-text field with respect to the evaluation metrics mean average precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative gain (NDCG), limited to k=20 retrieved documents

Field		`title`			`abstract`	
Metric	MAP	MRR	NDCG	MAP	MRR	NDCG
TF-IDF	.40 (.29)	.**51**(.39)	.**20** (.15)	.35 (.32)	.**47** (.43)	.**20** (.21)
normalised word vectors						
WCS_{GLV}	.29 (.29)	.38 (.41)	.15 (.16)	.27 (.26)	.35 (.37)	.14 (.14)
WCS_{W2V}	.30 (.26)	.38 (.38)	.15 (.15)	.30 (.32)	.37 (.41)	.13 (.14)
$IWCS_{GLV}$	.37 (.34)	.45 (.43)	.17 (.16)	.33 (.30)	.44 (.41)	.16 (.16)
$IWCS_{W2V}$	.41 (.35)	.50 (.41)	.19 (.15)	.**36** (.33)	.**47** (.43)	.17 (.16)
non-normalised word vectors						
WCS_{GLV}	.31 (.29)	.37 (.37)	.16 (.15)	.31 (.29)	.42 (.41)	.15 (.14)
WCS_{W2V}	.45 (.34)	.45 (.44)	.17 (.18)	.33 (.31)	.44 (.43)	.16 (.17)
$IWCS_{GLV}$	.39 (.34)	.49 (.41)	.18 (.15)	.33 (.30)	.42 (.39)	.17 (.17)
$IWCS_{W2V}$	.39 (.33)	.46 (.40)	.19 (.17)	.**35** (.29)	.**47** (.41)	.18 (.16)

In case of long queries on the NTCIR2 dataset, we observe a drop in MAP performance when using normalised word vectors for centroid similarity along with the Word2Vec model (compare .45 to .30). For the IWCS, the difference between normalised and non-normalised word vectors is small ($\pm$.02 MAP).

Table A.3. The effect of word vector normalisation on word centroid similarity for the Economics dataset using the title field with respect to the evaluation metrics mean average precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative gain (NDCG), limited to k=20 retrieved documents

Field		`title`	
Metric	MAP	MRR	NDCG
TF-IDF	.37 (.38)	.42 (.44)	.26 (.30)
normalised word vectors			
WCS_{GLV}	.36 (.37)	.42 (.44)	.25 (.29)
WCS_{W2V}	.36 (.37)	.41 (.43)	.25 (.29)
non-normalised word vectors			
WCS_{GLV}	.36 (.37)	.42 (.44)	.25 (.29)
WCS_{W2V}	.36 (.37)	.42 (.43)	.26 (.29)

For the Economics dataset A.3 and the `title` field, we do only observe minor differences between normalised and non-normalised word vectors.

Table A.4. The effect of word vector normalisation on word centroid similarity for the Reuters dataset using either the title or the full-text field with respect to the evaluation metrics mean average precision (MAP), mean reciprocal rank (MRR) and normalised discounted cumulative gain (NDCG), limited to k=20 retrieved documents

Field	`title`			`full-text`		
Metric	MAP	MRR	NDCG	MAP	MRR	NDCG
TF-IDF	.52 (.35)	.61 (.43)	.41 (.32)	.51 (.37)	.58 (.43)	.44 (.36)
normalised word vectors						
WCS_{GLV}	.55 (.31)	.63 (.40)	.42 (.29)	.51 (.33)	.60 (.41)	.44 (.33)
WCS_{W2V}	.54 (.33)	.63 (.41)	.43 (.31)	.52 (.35)	.57 (.41)	.46 (.35)
$IWCS_{GLV}$	.58 (.31)	.69 (.39)	.45 (.29)	.54 (.34)	.63 (.41)	.47 (.33)
$IWCS_{W2V}$	.60 (.33)	.69 (.40)	.47 (.32)	.55 (.35)	.60 (.41)	.49 (.36)
non-normalised word vectors						
WCS_{GLV}	.55 (.31)	.65 (.40)	.43 (.29)	.52 (.33)	.60 (.41)	.45 (.33)
WCS_{W2V}	.57 (.33)	.64 (.40)	.46 (.31)	.54 (.36)	.60 (.42)	.48 (.36)
$IWCS_{GLV}$	.60 (.32)	.71 (.39)	.46 (.29)	.54 (.35)	.63 (.42)	.47 (.34)
$IWCS_{W2V}$	.59 (.33)	.68 (.40)	.48 (.32)	.55 (.37)	.59 (.42)	.50 (.37)

A.2 Out-of-vocabulary statistics

During the experiments (See Section 1.4), the text is transformed into lower case and split into tokens of at least two consecutive word-characters length. For each combination of embedding model and dataset, we provide the out-of-vocabulary (OOV) ratio of the embedding models with respect to our analysis procedure (See Table A.5). The out-of-vocabulary ratio is defined as $\frac{n_{\text{embedded tokens}}}{n_{\text{tokens}}}$.

Table A.5. Ratio of out-of-vocabulary terms of the employed models GloVe (GLV), Word2Vec (W2V) and Doc2Vec (D2V) for the datasets NTCIR2, Economics and Reuters using either the title or the full-text field.

Dataset	NTCIR2		Economics		Reuters	
Field	`title`	`abstract`	`title`	`full-text`	`title`	`full-text`
GLV	.04	.05	.01	.04	.03	.02
W2V	.08	.10	.04	.20	.13	.19
D2V	.04	.06	.02	.16	.07	.14

We observe that the OOV ratios for the W2V model are consistently higher than for the D2V model, whose values are in turn higher than the ones of the GLV model (See Table A.5). However, these statistics not only depend on the vocabulary of the model, but also on the analysis process of the documents and queries. In case of the `full-text` field and the W2V model (as well as the D2V model), at least the 50 most frequent out-of-vocabulary words were numbers. On the other side, the GLV model does contain (common) numbers, which explains the lower OOV ratio values.

A.3 Up-training of missing words

We chose to evaluate up-training of missing words on the Economics dataset, since it contains the most out-of-vocabulary words (See Table A.5). We start with a pre-trained word embedding and continue training for out-of-vocabulary words. The updates for the word vectors of the original pre-trained embedding are multiplied with a lock factor. A lock factor of zero means that the original vectors are not changed. In our setting, zero epochs of up-training differs from no up-training. Zero epochs of up-training lead to random initialisation of missing word vectors. The results of Table A.6 indicate that up-training of missing word vectors on titles is not helpful. When using a lock factor of .0 or .1, the attained metric scores are merely changed. Using a higher lock factor (.5) decreases the attained scores in all metrics.

Table A.6. Effect of up-training out-of-vocabulary words on the title field of the Economics dataset using skip-gram negative sampling. The updates for already existing word vectors are multiplied with the lock factor. Hence, a lock factor of zero freezes the original vectors.

Model	Lock	Epochs	MAP	MRR	NDCG
TF-IDF	—	—	.37 (.38)	.42 (.44)	.26 (.30)
IWCS_{W2V}	—	—	.37 (.37)	.43 (.43)	.27 (.30)
IWCS_{W2V}	—	0	.38 (.37)	.44 (.44)	.27 (.30)
IWCS_{W2V}	.0	50	.38 (.37)	.44 (.44)	.27 (.30)
IWCS_{W2V}	.0	100	.38 (.37)	.44 (.44)	.27 (.30)
IWCS_{W2V}	.1	50	.38 (.38)	.43 (.44)	.27 (.31)
IWCS_{W2V}	.5	50	.35 (.38)	.40 (.44)	.25 (.30)
IWCS_{GLV}	—	0	.37 (.37)	.43 (.44)	.27 (.30)

A.4 All-but-the-top embedding post-processing

In the following, we provide the results for the application all-but-the-top embedding post-processing [MBV17]. The global mean and the first D principal components are subtracted from the word vectors. As proposed by Mu, Bhat, and Viswanath [MBV17], we chose $D = 2$ for the GLV model and $D = 3$ for the W2V model.

Table A.7. Effect of all-but-the-top embedding post-processing considering D principal components on the Reuters dataset

Model	D	MAP	MRR	NDCG
		`title` field		
$IWCS_{GLV}$	—	.58 (.31)	.69 (.39)	.45 (.29)
$IWCS_{W2V}$	—	.60 (.33)	.69 (.40)	.47 (.32)
$IWCS_{GLV}$	2	.35 (.30)	.39 (.37)	.27 (.27)
$IWCS_{W2V}$	3	.55 (.32)	.63 (.40)	.42 (.30)
		`full-text` field		
$IWCS_{GLV}$	—	.54 (.34)	.63 (.41)	.47 (.33)
$IWCS_{W2V}$	—	.55 (.35)	.60 (.41)	.49 (.36)
$IWCS_{GLV}$	2	.22 (.25)	.26 (.33)	.16 (.22)
$IWCS_{W2V}$	3	.53 (.35)	.59 (.41)	.44 (.35)

We provide the results for word centroid similarity with all-but-the-top post-processed word embeddings on the Reuters dataset (See Table A.7). All-but-the-top embedding post-processing does not improve the performance of word centroid similarity in our setting.

Appendix B

Integration into Elasticsearch

We evaluated word embeddings for information retrieval in a practical setting. We concluded that the IDF re-weighted word centroid similarity is favourable in terms of ranking quality. While our evaluation and comparison is based on a `python` implementation, we simulated a practical setting, which can be realised in Elasticsearch [1]. Since we apply the actual embedding operation on top of the token frequencies, the integration of word embeddings in Elasticsearch is possible. However, the required cost for a query-document similarity computation would be increased by summation of the word vectors of the specific document. As this is an expensive operation, it is desired to compute the (IDF re-weighted) word centroids at index time instead of query time. Therefore, we propose to implement a dedicated `TokenFilter`, which takes raw token counts as input and produces a token stream whose size matches the dimensionality of the word embedding. This output token stream resembles the (possibly IDF re-weighted) word centroid vector for the respective document. Please note, that this would effectively nullify the matching operation, since word vectors are dense. Thus, the word centroid has to be stored separately, in addition to the raw token counts and should not affect the matching operation. To obtain the desired behaviour, a dedicated `Similarity` is also necessary. Thus, the following additions to Elasticsearch are required:

`EmbeddingTokenFilter` A token filter that adds the word centroid for the document to a token stream input.

`EmbeddedSimilarity` A similarity that disregards the raw token counts but takes only the word centroids into account.

[1] https://www.elastic.co/products/elasticsearch

At index time, the documents are analysed and then the centroid is computed by the `EmbeddingTokenFilter`, while the original token counts are retained. In the analysis process at query time, we only analyse the query up to the raw token counts and do *not* apply the `EmbeddingTokenFilter`. Then the matching operation matches the documents in the index. After the matching operation, the `EmbeddedSimilarity` instance may transform the query into its word centroid representation and compute the cosine distance with respect to the word centroid of the document. A data flow

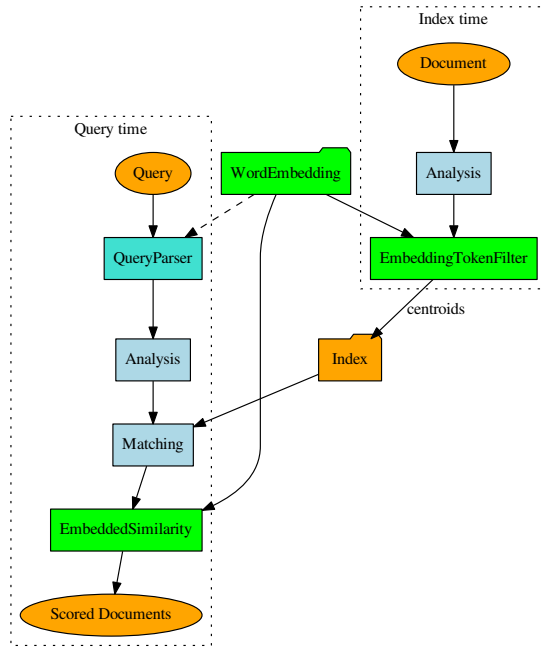

Figure B.1. Data flow graph for the integration of embedding-based retrieval techniques into Elasticsearch. Ellipsoid shapes resemble volatile raw data, while rectangular shapes resemble algorithms. Folder-like shapes represent persistent data.

graph is shown in Figure B.1. To summarise, word embeddings can be integrated into Elasticsearch by a plug-in that provides a new dedicated token filter and similarity. It is necessary, to carefully store the centroids in the index without interfering with the matching operation. At query time, the `EmbeddedSimilarity` class needs to aggregate the word vectors to their centroids, before the cosine similarity is computed. In the case of embedding-based query expansion, a dedicated `QueryParser` would be a possible future extension.

Bibliography

[AR02] Gianni Amati and C. J. van Rijsbergen. "Probabilistic models of information retrieval based on measuring the divergence from randomness". In: *ACM Trans. Inf. Syst.* 20.4 (2002), pp. 357–389. DOI: 10.1145/582415.582416. URL: http://doi.acm.org/10.1145/582415.582416.

[BA16] Georgios Balikas and Massih-Reza Amini. "An empirical study on large scale text classification with skip-gram embeddings". In: *CoRR* abs/1606.06623 (2016). URL: http://arxiv.org/abs/1606.06623.

[BBL99] Doug Beeferman, Adam L. Berger, and John D. Lafferty. "Statistical models for text segmentation". In: *Machine Learning* 34.1-3 (1999), pp. 177–210. DOI: 10.1023/A:1007506220214. URL: http://dx.doi.org/10.1023/A:1007506220214.

[BCV13] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.8 (2013), pp. 1798–1828. DOI: 10.1109/TPAMI.2013.50. URL: http://dx.doi.org/10.1109/TPAMI.2013.50.

[BDV+03] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. "A neural probabilistic language model". In: *Journal of Machine Learning Research* 3 (2003), pp. 1137–1155. URL: http://www.jmlr.org/papers/v3/bengio03a.html.

[Bir06] Steven Bird. "NLTK: the natural language toolkit". In: *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*. Ed. by Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle. The Association for Computer Linguistics, 2006. URL: http://aclweb.org/anthology/P06-4018.

[BLB+13] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation". In: *Journal of Machine Learning Research* 3 (2003), pp. 993–1022. URL: http://www.jmlr.org/papers/v3/blei03a.html.

[CP13] Stéphane Clinchant and Florent Perronnin. "Textual similarity with a bag-of-embedded-words model". In: *International Conference on the Theory of Information Retrieval, ICTIR '13, Copenhagen, Denmark, September 29 - October 02, 2013*. Ed. by Oren Kurland, Donald Metzler, Christina Lioma, Birger Larsen, and Peter Ingwersen. ACM, 2013, p. 25. DOI: 10.1145/2499178.2499180. URL: http://doi.acm.org/10.1145/2499178.2499180.

[CW08] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: deep neural networks with multitask learning". In: *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*. Ed. by William W. Cohen, Andrew McCallum, and Sam T. Roweis. Vol. 307. ACM International Conference Proceeding Series. ACM, 2008, pp. 160–167. DOI: 10.1145/1390156.1390177. URL: http://doi.acm.org/10.1145/1390156.1390177.

[Fag87] Joel L. Fagan. "Automatic phrase indexing for document retrieval: an examination of syntactic and non-syntactic methods". In: *Proceedings of the Tenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans, Louisiana, USA, June 3-5, 1987*. Ed. by C. T. Yu and C. J. van Rijsbergen. ACM, 1987, pp. 91–101. DOI: 10.1145/42005.42016. URL: http://doi.acm.org/10.1145/42005.42016.

[FDD+88] George W. Furnas, Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, Richard A. Harshman, Lynn A. Streeter,

and Karen E. Lochbaum. "Information retrieval using a singular value decomposition model of latent semantic structure". In: *SIGIR'88, Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Grenoble, France, June 13-15, 1988*. Ed. by Yves Chiaramella. ACM, 1988, pp. 465–480. DOI: 10.1145/62437.62487. URL: http://doi.acm.org/10.1145/62437.62487.

[Got16] Gregory Goth. "Deep or shallow, NLP is breaking out". In: *Commun. ACM* 59.3 (2016), pp. 13–16. DOI: 10.1145/2874915. URL: http://doi.acm.org/10.1145/2874915.

[Hie98] Djoerd Hiemstra. "A linguistically motivated probabilistic model of information retrieval". In: *Research and Advanced Technology for Digital Libraries, Second European Conference, ECDL '98, Heraklion, Crete, Greece, September 21-23, 1998, Proceedings*. Ed. by Christos Nikolaou and Constantine Stephanidis. Vol. 1513. Lecture Notes in Computer Science. Springer, 1998, pp. 569–584. DOI: 10.1007/3-540-49653-X_34. URL: http://dx.doi.org/10.1007/3-540-49653-X_34.

[Hof99] Thomas Hofmann. "Probabilistic latent semantic indexing". In: *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA*. Ed. by Fredric C. Gey, Marti A. Hearst, and Richard M. Tong. ACM, 1999, pp. 50–57. DOI: 10.1145/312624.312649. URL: http://doi.acm.org/10.1145/312624.312649.

[KSK+15] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. "From word embeddings to document distances". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 957–966. URL: http://jmlr.org/proceedings/papers/v37/kusnerb15.html.

[LM14] Quoc V. Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *Proceedings of the 31th*

International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pp. 1188–1196. URL: http://jmlr.org/proceedings/papers/v32/le14.html.

[LYR+04] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. "RCV1: A new benchmark collection for text categorization research". In: *Journal of Machine Learning Research* 5 (2004), pp. 361–397. URL: http://www.ai.mit.edu/projects/jmlr/papers/volume5/lewis04a/lewis04a.pdf.

[MBV17] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. "All-but-the-top: simple and effective postprocessing for word representations". In: *CoRR* abs/1702.01417 (2017). URL: http://arxiv.org/abs/1702.01417.

[MLS99] David R. H. Miller, Tim Leek, and Richard M. Schwartz. "A hidden markov model information retrieval system". In: *SIGIR '99: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 15-19, 1999, Berkeley, CA, USA*. Ed. by Fredric C. Gey, Marti A. Hearst, and Richard M. Tong. ACM, 1999, pp. 214–221. DOI: 10.1145/312624.312680. URL: http://doi.acm.org/10.1145/312624.312680.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. ISBN: 978-0-521-86571-5.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. "Distributed representations of words and phrases and their compositionality". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger. 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.

[MYH09] Andriy Mnih, Zhang Yuecheng, and Geoffrey E. Hinton. "Improving a statistical language model through non-linear prediction". In: vol. 72. 7-9. 2009, pp. 1414–1418. DOI: 10.1016/j.neucom.2008.12.025. URL: http://dx.doi.org/10.1016/j.neucom.2008.12.025.

[PC98] Jay M. Ponte and W. Bruce Croft. "A language modeling approach to information retrieval". In: *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*. Ed. by W. Bruce Croft, Alistair Moffat, C. J. van Rijsbergen, Ross Wilkinson, and Justin Zobel. ACM, 1998, pp. 275–281. DOI: 10.1145/290941.291008. URL: http://doi.acm.org/10.1145/290941.291008.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: global vectors for word representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, 2014, pp. 1532–1543. URL: http://aclweb.org/anthology/D/D14/D14-1162.pdf.

[PVG+11] F. Pedregosa et al. "Scikit-learn: machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[ŘS10] Radim Řehůřek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora". English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[RWH+95] Stephen E. Robertson, Steve Walker, Micheline Hancock-Beaulieu, Mike Gatford, and A. Payne. "Okapi at TREC-4". In: Special Publication 500-236 (1995). Ed. by Donna K. Harman. URL: http://trec.nist.gov/pubs/trec4/papers/city.ps.gz.

[SB88] Gerard Salton and Chris Buckley. "Term-weighting approaches in automatic text retrieval". In: *Inf. Process. Manage.* 24.5 (1988), pp. 513–523. DOI: 10.1016/0306-4573(88)90021-0. URL: http://dx.doi.org/10.1016/0306-4573(88)90021-0.

[SFW83] Gerard Salton, Edward A. Fox, and Harry Wu. "Extended boolean information retrieval". In: *Commun. ACM* 26.11 (1983), pp. 1022–1036. DOI: 10 . 1145 / 182 . 358466. URL: http://doi.acm.org/10.1145/182.358466.

[TRB10] Joseph P. Turian, Lev-Arie Ratinov, and Yoshua Bengio. "Word representations: A simple and general method for semi-supervised learning". In: *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*. Ed. by Jan Hajic, Sandra Carberry, and Stephen Clark. The Association for Computer Linguistics, 2010, pp. 384–394. URL: http://www.aclweb.org/anthology/P10-1040.

[ZC15] Guoqing Zheng and Jamie Callan. "Learning to reweight terms with distributed representations". In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*. Ed. by Ricardo A. Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto. ACM, 2015, pp. 575–584. DOI: 10.1145/2766462.2767700. URL: http://doi.acm.org/10.1145/2766462.2767700.

[ZC16] Hamed Zamani and W. Bruce Croft. "Embedding-based query language models". In: *Proceedings of the 2016 ACM on International Conference on the Theory of Information Retrieval, ICTIR 2016, Newark, DE, USA, September 12- 6, 2016*. Ed. by Ben Carterette, Hui Fang, Mounia Lalmas, and Jian-Yun Nie. ACM, 2016, pp. 147–156. DOI: 10.1145/2970398.2970405. URL: http://doi.acm.org/10.1145/2970398.2970405.

[Zha08] ChengXiang Zhai. "Statistical language models for information retrieval: A critical review". In: *Foundations and Trends in Information Retrieval* 2.3 (2008), pp. 137–213. DOI: 10.1561/1500000008. URL: http://dx.doi.org/10.1561/1500000008.