Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master Thesis

Document Similarity using Dense Vector Representation

In cooperation with:
XING AG
Data Science

Ahmed Elsafty

4elsafty@informatik.uni-hamburg.de
Intelligent Adaptive Systems Master program
Matriculation No. 6641893

First Supervisor Hamburg University:	Prof. Dr. Chris Biemann
Second Supervisor Hamburg University:	Dr. Martin Riedl
External Supervisor XING AG:	Dr. Fabian Abel

Abstract

The purpose of this thesis is to analyze different document representation models for identifying similar job advertisements to improve the quality of job recommendations on XING, a social network for professionals. We test our algorithms both in an offline experiment as well as in an online evaluation. For offline experimentation, we present a method for automatically creating a training dataset based on user interactions. We experiment with different pre-processing techniques like lowercasing the input text and stemming which shows significant increase in performance. Moreover, we test four different word embedding techniques: average and weighted average of vectors using Word2Vec, Word2VecF using arbitrary context and Doc2VecC. Results show performance boost when a weighted average is used and both global and local context is included in the model training. We also analyze the performance of LDA and discover that it does not perform well on the dataset. The best results in the offline evaluation are achieved when exploiting more information like the job title along with the full job posting description. Titles are used for weighing the word vectors more accurately when building the vector representing the entire document. Finally, we implement and integrate the best performing model also in XING's job recommendation system and conduct an online A/B test evaluation with Millions of XING users. The results from that online evaluation confirm our findings from the offline experimentation and lead to an +8.0% increase of the click-through rate on job recommendations.

Acknowledgment

First of all, I would like to sincerely thank my supervisor, Dr. Martin Riedl, for his assistance, priceless guidance, and advice throughout my thesis. I would also like to thank Prof. Chris Biemann for his efforts to make the process as smooth as possible, beginning from the thesis proposal, to the very end of it.

Special thanks to my Ruben for always dedicating the time to answer my questions and assisting me with my technical problems. I would like to thank Mirko for his help during the thesis and throughout my entire master degree. Special thanks for Fabian for proposing the research problem and reviewing the thesis. I would also like to thank Daniel for his precious guidance and the entire Data Science team for providing me with a proper working environment and the needed facilities to complete the dissertation.

Finally, I would like to thank my parents for their continuous support and the tremendous sacrifices they made to ensure that I get the best possible education. Everything what I am now is because of you.

Contents

List of Figures

List of Tables

1 Introduction

Recommender Systems have received plenty of attention in the past decade. Given the possibility to interact with a catalog of millions of items on e-commerce companies such as Amazon[1], users can utilize the search functionality to query the needed items, or a system can be used to recommend items that the users may find relevant. Recommender systems are exploited to suggest movies on Netflix[2], music tracks on Spotify[3], Novels on Goodreads[4] and hundreds of different items categories from different retailers and e-commerce companies.

By exploiting certain user features and the user history of items interactions, recommender systems may use an ensemble of recommendation algorithms to provide the user with possibly relevant items. Instead of using static weights for combining the ensemble of recommender strategies, a Recommender system constantly retrains and adjusts its model. For example, a recommender system may need to balance between relevance and diversity of the recommendation list. Recommending items for which the system estimates high relevance may seem like a safe way to ensure steady revenue. However, recommendations also need to be diversified to provide the user with more opportunities and thus allowing the user to find interesting items outside of her *filter bubble* while maintaining an acceptable level of accuracy [Adomavicius and Kwon 2012]. This gives Recommender systems flexibility to changing trends as well as the ability to provide personalized results for every user.

A Recommender system that retrieves more relevant and interesting items for users helps increasing the revenue for e-commerce companies. Similarly, a recommender system in social networks that suggests more relevant news and trends would win customers away from their competitors and reduces the risk of churn.

XING[5] is a social network company with approximately 18 million users worldwide [Abel et al. 2016]. It offers, for example, functionality for exploring job offers: users can either explicitly search for jobs or can explore jobs via a job recommendation service. Around 45% of the traffic on job offers is driven by the job recommendation service. Hence, the job recommendation system is a crucial component on the XING platform that requires constant improvement.

[1] http://www.amazon.com/
[2] https://www.netflix.com/
[3] https://www.spotify.com/
[4] http://goodreads.com/
[5] https://www.xing.com/

XING's job recommendation system is an ensemble of recommender models. It is organized into four core recommender engines that identify items that are possibly relevant to the user and around 20 filters, re-ranking strategies and diversification components that focus on optimizing precision. Together, those core engines and other components exploit around 200 features.

One of XING's core recommender engine and currently the engine with the highest impact on the recommendations is the so-called More-like-this (MLT) component: it recommends the user those job postings that are similar to the ones that the user interacted with in the past. Therefore, it exploits metadata of a job posting like keywords, disciplines and industries in which the job is categorized. Such metadata is converted into a search query to retrieve similar documents from the job posting inventory and rank them.

There are multiple problems that may rise from using exact keyword matching or matching of categories for retrieving and ranking job postings. First, the document collection, with 1 Million job postings, is fairly huge and too diverse to fit into the small number of categories that XING uses, e.g. for its disciplines (22 disciplines such as *HR* or *IT & Software Development*). If one would group the jobs by their discipline category, one would thus result clusters of around 45,000 postings and it is likely that there is quite a high diversity within such a cluster, i.e. not all jobs of the cluster are similar and could be filled by the same type of candidates.

Second, strict word matching has other side effects, for instance, *J2EE Developer* will not be similar to *Software Engineer*, nor *Data Lake Engineer* to *Data Architect*. Moreover, by only using the metadata of a job, which are typically supplied by HR agents or recruiters, we lose information in the full-text description of the job offer itself that may solely identify similarity between documents and clear ambiguities.

In this thesis, we analyze to what extent the full-text description of a job posting can be used to identify similar job postings and thus enhance the aforementioned MLT-component of XING's job recommendation system. We propose to get a score of similarity based on the XING's job posting description, using Dense Vector representation to avoid strict matching. Recent word embedding techniques can learn meaningful syntactic and semantic relationship based on word occurrences in the text, which got extended to documents as well. Given the ideal model, similar job postings in the inventory will be placed in separate places in the vector space, making it easy to retrieve and rerank similar job postings.

1.1 Goal

The goal of this thesis is to explore document representation methods that allow for detecting similar job postings. In particular, we aim to solve the following problem:

> Given the full-text description of a job posting p, find a vector representation of p that allows to identify similar postings. Here, two items p_1 and p_2 are similar if both items describe a similar job role.

Hence, aspects such as the company that is offering the job, benefits offered by the company or the location of the job do not need to be taken into account by the similarity measure. The document representation strategy should be applied in XING's job recommendation system and in the MLT-component in particular. Instead of using a plain keyword- or category-based query to rank job posting recommendations, we aim to exploit vector representation of the postings' full-text descriptions so that similar job postings can be found closer to each other in the vector space and non-similar postings are distant apart from each other. A proper document representation method will eventually lead to better recommendations as XING's key recommender engine is an item-to-item based recommender algorithm that recommends those items that are similar to a user's previously liked items.

1.2 Research Questions and Limitations

In this section, we list down the main challenges, limitations and research questions involved in finding the best document representation for a given dataset of job postings.

- There is no open source dataset of similar job postings. As we want to exploit the structure and vocabulary of a narrow scoped dataset and would like to explore how state-of-the-art word embedding techniques perform in the domain of job postings, using a general open source document similarity dataset should be the last resort. Hence, we need to solve the following question: How can we build a closed domain dataset of job postings for the evaluation of the document similarity problem stated above?

- What kind of strategies and particularly word embedding techniques can solve the problem of identifying similar job postings? And what kind of pre-processing methods need to be applied on the full-text in order to allow for the best perfor-

mance? Can we possibly find a document representation that uniquely place similar documents in distinctive places in the vector space, such that querying similar job posting is as accurate as possible?

- How can we exploit additional types of information including metadata such as the title of job postings and combine several representations so that we improve the performance of the models for detecting similar postings?

- How do the strategies for representing documents by means of dense vectors perform in real life in XING's job recommendation system? How will our models impact the accuracy and quality of the job recommendations? And how do we build the architecture and implement our strategies so that computations for building the model can be done in a reasonable amount of time and that the strategies can be deployed to the recommender service that receives Millions of requests per day and up to 500 requests per second in peak times?

1.3 Thesis Structure

In the following, we briefly outline the structure of the thesis. In Chapter 2 we give an overview of current approaches for representing document vectors. In addition, we provide background details on different topic modeling, word embeddings algorithms and similarity metrics that are used in the thesis. Here, we focus on dense representations that can be achieved using variants of Word2Vec and topic models. Chapter 3 introduces the experimental setup for the topic model comparison, where we discuss the logic behind data preprocessing and creating the ground truth and the evaluation datasets. Following this, chapter 4 lists the results of the offline experiments. Here, we compare the performance of multiple semantic models, different hyperparameters and combination of features. while chapter 5 describes the workflow of the online evaluation, along with the results from conducting the experiments on live XING users. Chapter 6 concludes the thesis while Chapter 7 introduces the Future Work, which includes interesting research questions raised by the thesis that couldn't be tackled due to the time constraint.

2 Background

In this chapter, we go over the related work in the field of document similarity, moving from the basics of bag-of-word models and Latent Semantic Indexing to topic and semantic models, then we list down some of the available document similarity datasets.

Additionally, we describe the current architecture of XING recommender system, including the data collected from users and howit gets exploited by different subrecommenders to generate the job recommendations. We also describe the architecture of different word and document representation models used throughout the thesis, in order to have better understanding of the underlying training workflows and the hyperparameters used, which helps in justifying why certain hyperparameters were selected during conducted experiments later on.

We proceed by explaining word representation models like Word2Vec and its variation Word2VecF, followed by document representation models like Doc2Vec and LDA, then we discuss a word/document Hybrid model known as Doc2VecC. Finally, we present the scoring, similarity and evaluation metrics used during the thesis.

2.1 Related Work

The task of a documents similarity has been intensively studied. Bag-of-Words model [Harris 1954] was one of the earliest fixed length vector documents representations over the entire vocabulary, where two documents are considered similar if they share the same set of words. However, the word order is not significant, and non similar documents can have the same representation if the same words happen to occur in the documents pair. For instance, "X is slower than Y" would have the same document vector as "Y is slower than X".

Better performing Bag-of-Words models integrated different weighting function for the document vector. For instance, TF-IDF weights were used to allow certain words to influence the similarity calculation more than other words that appear often throughout the corpus [Salton et al. 1975]. However, the approach failed in deducing relationship between synonyms, for example, *Doctor* will not be similar to *Physician*. More problems appear when a word can be used in multiple contexts (polysemes), like *Ruby* and Apache *ZooKeeper*, which reduces the recall.

To address some of these problems, Latent Semantic Indexing (LSI) was introduced which represents a document by reducing the dimensionality of the TF-IDF vocabulary vector to a smaller condensed topic vector, such that words and documents from the same topic are mapped closer to each other. LSI reduces - but doesn't solve - problems with polysemes since words are grouped together based on the topic [Deerwester et al. 1990]. Better document representation algorithms were built on the concept of LSI like Latent Direchlet Allocation (LDA) [Blei et al. 2003], which represent a document as a distribution of topics, where every topic is presented as a multinomial distribution over the corpus vocabulary. LDA is further explained in Section 2.3.5.

Le and Mikolov [2014] introduced Paragraph vector representation which can be learned from variable-length text snippets, such as sentences, paragraphs and documents. Paragraph vectors are learned by predicting the surrounding words in the text snippets, demonstrating better performance than bag-of-words models. Paragraph vectors along with different embedding models are explained in more details in Section 2.3.

Regarding existing document similarity dataset, The 20 Newsgroups [Lang 1995] and TREC-AP [Lewis et al. 1996] are examples of document categorization dataset where the task is to correctly classify a document against 20 classes. The datasets can be used for document similarity if all documents within a given class are considered similar [Huang 2008]. Baudiš et al. [2016] introduced different sentence pair scoring datasets to tackle multiple NLP tasks, two of them are the *Semantic Text Similarity* and *Paraphrasing* datasets. Semantic text similarity (STS) datasets compares independent pairs of short sentences with a score from 0 to 5, based on semantic similarity. While Paraphrasing (Para) is a binary classified dataset collected from Question and Answer platform[1]. It states whether a given sentences pair share the same topic, asking the same question or describing the same event. Unfortunately, the available datasets discuss general topics that are not specific to job postings, recruitment, hiring or salary. Most of them are oriented towards sentiment analysis of short sentences, where the stemmed preprocessed job descriptions at XING contain 29 words on average. To the date of writing the thesis, we didn't find any open source dataset of similar job postings.

2.2 XING Recommender System

XING is a social networking platform for professionals, with a descriptive slogan "to provide a better working life that enables professionals to grow". The platform provides services for people who want to find better job opportunities to grow their careers and

[1] https://askubuntu.com/

connect them with other professionals in the field. Therefore, XING can recommend the users nearby technical events that suits their skill sets, fitting job offers, other professionals that share similar skills and interests, and more.

The more XING knows about the users, the better the quality of recommendations it can retrieve for them. Figure 2.1 shows different input sources for the recommender system. Users can create a user profile that contains professional data like previous work experiences, the skill set they possess and job opportunities they are interested in, as well as the ability to add their co-workers and acquaintances to their list of contacts in a social network perspective. The recommender system aggregates these different data sources to generate relevant recommendations.

The user interaction behavior on the platform is being tracked to analyze what kind of job postings the users are clicking, bookmarking or deleting. This would allow the system to recommend job postings similar to those the user positively annotated via bookmarks and reply intentions, or inhibit job postings similar to those the user deleted.

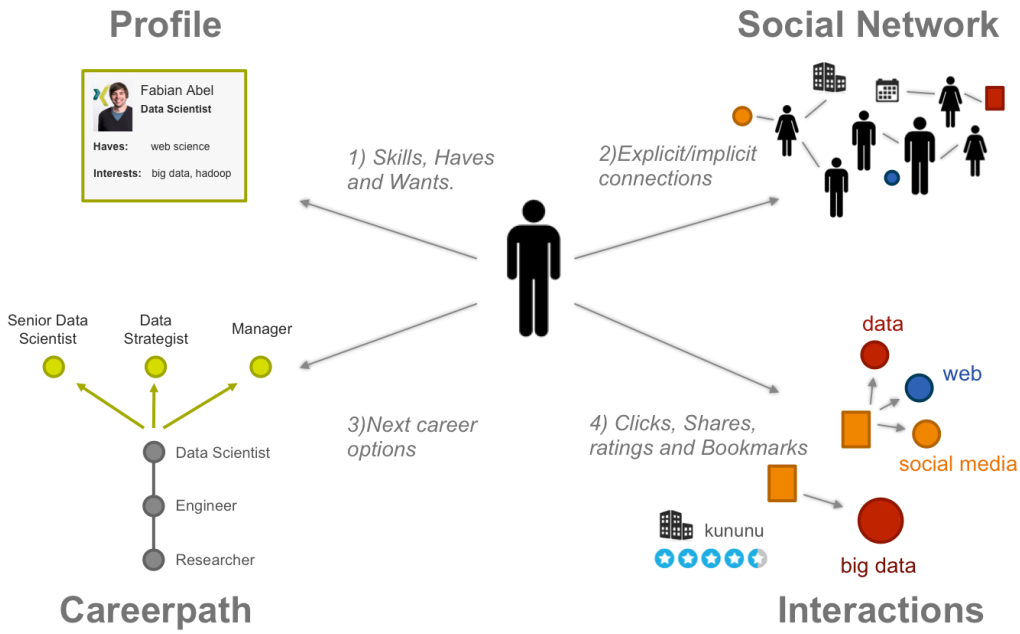


Figure 2.1: The more interactive and cooperative the users are in supplying their information on the platform, the better recommendations they receive. Like interacting with relevant jobs and supplying their current city, previous employers and their skill sets [Image Source: XING internal].

The recommender system consists of 4 separate sub-recommenders, each sub-recommender is specialized in one of the main user information branches shown in Figure 2.1, where they return a list of recommendations with a score. A final recommender aggregates all recommendation lists based on the score and co-occurrences of job postings. i.e If the

same recommended job was returned by multiple subrecommender, the recommendation score for this job gets boosted.

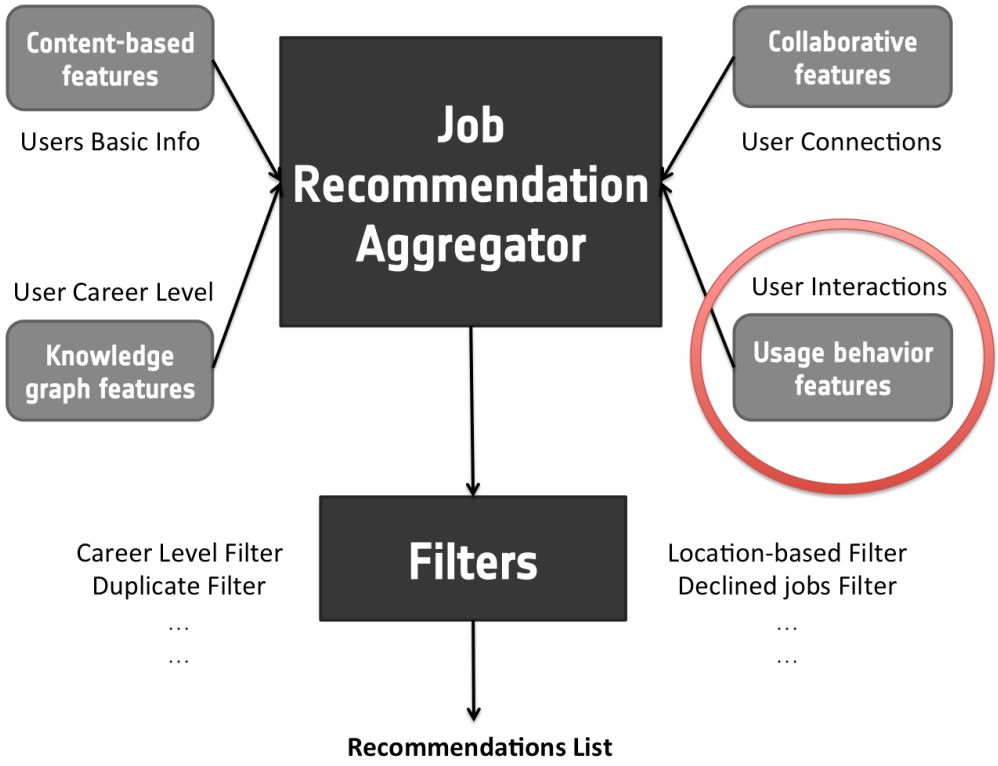


Figure 2.2: XING recommendation System consists of recommendation aggregator that combines output from 4 main sub-recommenders, along with the filtering phase, the system is used to retrieve and rerank the recommendation list for the user requesting it.

Figure 2.2 shows the architecture of the job recommender, it consists of two main parts, the recommender aggregator and the filters stage. The aggregator combines scores from different sub-recommenders and the filters removes job postings that does not meet certain criteria, like different career level or city. The sub-recommenders utilize the users' data as follows:

- **Content-based Recommender:** Exploits the user information on the platform to match profile against job posting ads. Since user profiles contain Haves, Wants and Interests as keywords (see Appendix A.1), Content-based Recommender simply matches them to the job postings' keywords supplied by the job owners and HR to retrieve relevant job recommendations.

- **Collaborative Recommender:** If a job is suited for user X, it must be suited for user Y if both users are similar in experiences and skill sets. Hence the Collaborative Recommender groups similar users together, and boost job postings for a given

user that similar users are also interested in. It also uses the user information, like contact list, to boost job postings in cities and companies where the user already knows contacts and acquaintances.

- **Knowledge Graph Recommender:** Exploits the users' previous work experiences, to predict their next job positions. Knowledge Graph Recommender consider the transition probabilities between job roles, analyzed from the user work experiences and discipline transition to have a statistical estimate of the next career level. The recommender provides the user with job posting recommendations for this estimated future career level.

- **Usage behavior Recommender:** Also known as MLT Recommender (More Like This), it retrieves job postings similar to those the user interacted with positively in the last 3 months. It uses Elastic Search to retrieve 150 job postings (on each request) based on discipline of positively interacted job postings, along with their title, and ontology ID of skills and "keywords" used in the Content-based Recommender.

After collecting all recommendation lists from the 4 sub-recommenders with predefined weights for each sub-recommender, the score is aggregated and reranked, then the recommendations are filtered out to avoid recommending job postings that user already bookmarked, or job postings in different cities or countries.

Document similarity algorithm can be applied on the MLT recommender, as the recommendation list is sorted by the Elastic Search ranking, using metadata and specific keywords from the job owners, it is prone to noisy words and it can be improved. As shown later in the thesis, strict matching shows inferior performance compared to semantics based similarity. In this thesis, we will alter the rescoring function in the MLT recommender to use a document embedding algorithm that exploits the description of the job postings.

2.3 Dense Vector Representations

In this section, we discuss multiple semantic models that are used to generate word embeddings, parameterization and different architectures, in order to justify the choice of hyperparameters and model decisions used later in the thesis.

2.3.1 Word2vec

Word2Vec represents words as vectors in a multi-dimensional vector space, where similar words tend to be close to each other [Mikolov et al. 2013a]. In order to gain such a representation, two techniques have been proposed: the Continuous Bag of Word (CBOW) model and the Skip-gram model. CBOW model predicts the current word using the surrounding words (context) in a given *window size*, and the Skip-gram model predicts the surrounding words using the current word.

Dense vector representation extracts semantic relationships based on the co-occurrence of words in the dataset. The accuracy of representation of a given two words depends on how many times the model sees these words within the same context throughout the corpus. More word and context co-occurrences during training changes the hidden representation, which allows the model to have more future successful predictions, leading to a better representation of word and context in the vector space.

Figure 2.3 shows the architecture of the CBOW and SKIP-gram models, $w(t)$ denotes the word in a sentence at index t. In a window of size N, the context ranges from $w(t-N)$ to $w(t+N)$ excluding $w(t)$, hence the term $w(t \pm 1..N)$.

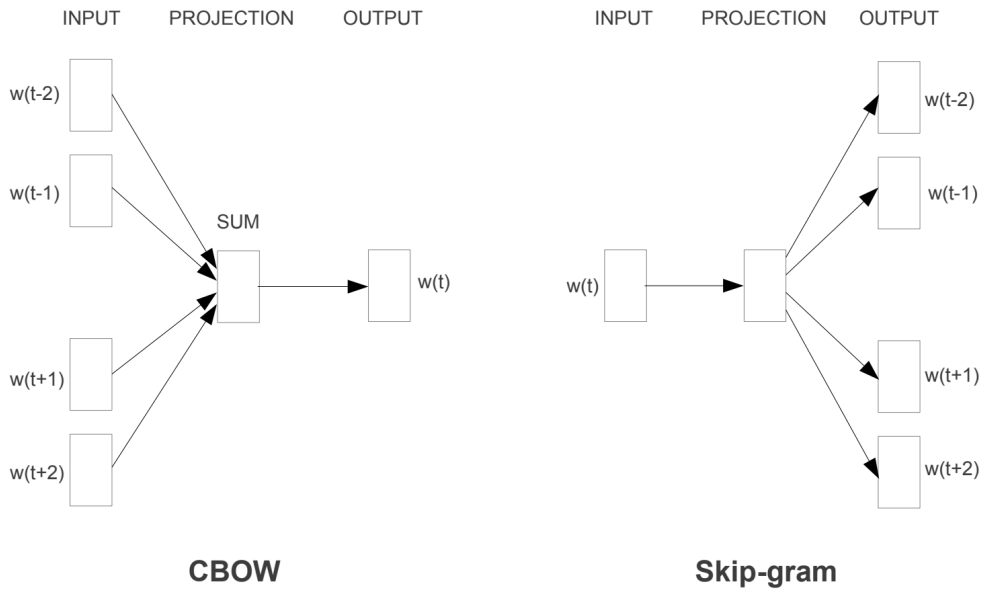


Figure 2.3: In a window of size N, The CBOW model uses the context $w(t \pm 1..N)$ to predict the current word $w(t)$, while the Skip-gram predict the context given the current word [Mikolov et al. 2013a].

The number of vector dimensions depicts the size of the projection layer shown in figure 2.3. The hidden layer is learned in unsupervised fashion during training of the neural

network like in conventional neural networks. While the projection layer is a matrix composed of the hidden layer weights combined with the input layer, where the matrix is shared for all words in the vocabulary [Mikolov et al. 2013a].

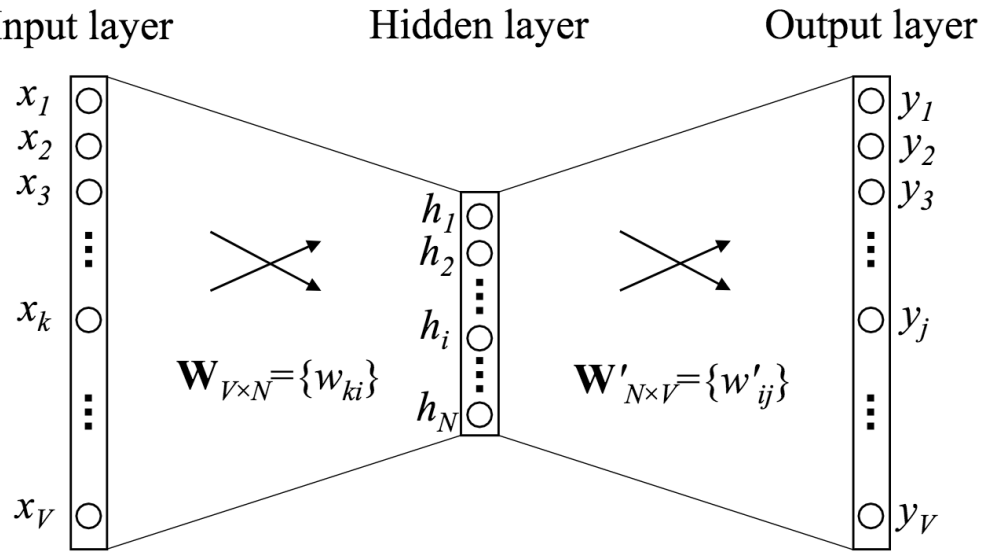


Figure 2.4: Given an input word x_i, the bigram model predicts the second word y_i in a context window of size 1 using the input and output layers of size V (vocabulary size) and hidden layer of size N. The weight matrix W_{V*N} depicts the N dimensional weights of V vocabulary words, where W'_{N*V} denotes the activation matrix needed to calculate the output. [Rong 2014].

Figure 2.4 shows the detailed architecture of the CBOW model with a window of size 1, also called a bigram model, with N number of vector dimensions depicting the size of the hidden layer with vocabulary of size V. The goal is to predict the second word in the window given the first one. The entire V words in the vocabulary is hot-encoded in the input layer, which means that given an input word, only one unit in x_1 to x_V will be 1 and the rest will be 0.

The weights between the input and the hidden layer constitutes a $V * N$ matrix W, where each row in W is an N dimensional vector representation of the one the V words in the vocabulary [Rong 2014]. The predicted output is calculated by applying Softmax function on another Matrix W'_{N*V} containing the activation vectors for every word in the vocabulary, resulting in a single output. Error is propagated back across the network to modify the weights in W and W' to reduce the error in future iterations.

Matrix W contains the word vectors which can be used directly in the latent space for similarity evaluation. Choosing the right number of vector dimensions and other hyperparameters can greatly affect the results. There are various parameters that will be

discussed in the next subsections.

Window Size

Window size limits the words used in the context during training to twice the value. For example, window size of 5 will include the 5 words to the left and the 5 words to the right for each observed word in the sentence as context. Increasing the window size may include context words that are not relevant to the current word. However, decreasing the window size can capture relations between words and stopwords which is also not preferred.

Kottur et al. [2016] and Levy et al. [2015] experimented the embeddings of their datasets using window sizes of 2, 5, 10 as error rate fluctuates across window sizes. Another reason to pick a proper window size is that it affects the performance, as using a larger windows increases the training time, because more context pairs need to be considered. Window size will be cross validated in Section 4.1 as advised by Le and Mikolov [2014].

Vector Dimensions

Melamud et al. [2016] evaluated skip-gram model on multiple intrinsic and extrinsic NLP tasks to test how number of dimensions affect the accuracy of results. One of the four extrinsic benchmarks tested was the Sentiment Analysis (SENTI [Pang and Lee 2005]) in a very similar setup to our experiment's, where word representations were evaluated in a sentiment analysis classification problem by averaging the words in the sentence to construct a sentence vector. Melamud showed that number of dimensions doesn't have big impact on the accuracy, when going from 300 to 600 dimensions (+2.6% increase). Therefore, we are going to conduct the experiment using 500 dimensions, which Levy et al. [2015] used while testing Skip-gram hyperparameters, and still conforms with the results from Melamud.

Negative Sampling

Negative Sampling is a mechanism to speed up the training time. Instead of updating the vectors of all words in the vocabulary on every iterations, only a portion of the vocabulary will be updated. By sampling random words from the vocabulary, chances that

they are similar to the current word $w(t)$ are very low. Negative Sampling aims to maximize the similarity between words that appear together in the context, and minimizing the similarity between $w(t)$ and the sampled words, instead of the entire vocabulary.

Subsampling

Subsampling, denoted as sampling rate in the C code, simply removes words that are more frequent than a certain threshold t with probability p, where f is the frequency of the word in the corpus.

$$p = 1 - \sqrt{\frac{t}{f}} \tag{2.1}$$

This approach imitates stopwords removal, as higher frequency words have higher probability of getting down-sampled from the vocabulary. In addition, Word2Vec has the **Minimum Count** parameter which removes words that occurs less than the value specific which helps filtering out noisy words.

Sampling rate of 10^{-5} was used in the experiments, it was used by Mikolov et al. [2013b] who proved its performance improvement. Minimum count of 5 is used, which is the default in the C implementation of Word2Vec.

Number of Threads

The C implementation offers parallelism while training to reduce the training time, set by number of threads. However, by analyzing the code, it appears that there is no locking between threads that write in the matrix of neural weights, some may overwrite each other. The technique is called "Asynchronous Stochastic Gradient Descent" which reduces the training time without significant effect on precision [Paine et al. 2013]. However, it may not be reproducible since the order of rewrites is random and may not be suitable for controlled experimentation. Hence, a **single thread** is used in the thesis.

Number of Training Iterations

Number of iterations depicts the number of times the model trains on the dataset, which is preferred as more iterations pushes the model towards convergence. In the ideal case,

the number of iterations should be plotted against the error [Hagan and Menhaj 1994] to decide when to stop training if the error doesn't significantly change as iterations go. However, it was technically tedious to assert the error in the C code implementation using python. According to Fallgren et al. [2016], increasing number of iterations doesn't lead to an increase in the score. Hence, 20 iterations were used in the thesis.

Since Word2Vec is conceptually a neural network, input dataset was shuffled between iterations because "faster convergence has been observed if the order in which the mini-batches are visited is changed for each epoch [Bengio 2012]".

Document Vector

As the aim is to project the documents into a vector space, we need document not word vectors to get properly placed in the space. Liu et al. [2015] and Hong and Fang [2015] suggested to use averaging of word vectors of a document as document representation. However, as more word vectors gets averaged, the more information regarding the uniqueness of the document is lost.

Huang et al. [2012] proposed a solution that involves weighted average of word vectors to represent the document, in order to favor words that are distinct to the document. Hence pulling the document vector to a point in the vector space that represents the documents' most noticeable features.

$$V(document_{k-words}) = \frac{\sum_{i=1}^{k} weight\_function(w_i)V(w_i)}{\sum_{i=1}^{k} weight\_function(w_i)} \tag{2.2}$$

Equation 2.2 describes the weighted average formula. A vector for a document consists of k words is calculated by summing all document's word vectors $V(w)$ multiplied by a weight function over the sum of all weights used. The $weight\_function(w)$ is a look-up table for word w and its corresponding weight. The weights can be a distribution of important keywords assembled by human experts, or by using a scoring function based on words distribution like *TF-IDF* (see Section 2.4.1).

2.3.2 Word2VecF

Word2VecF [Levy and Goldberg 2014] is a different implementation of Word2Vec, where the input is a file containing tuples of words and contexts (instead of a file with the entire

text), giving more control over what to feed to the network, with the possibility of using arbitrary context as shown in Table 2.1. Model embeddings can be biased by repeating specific context pairs multiple times throughout the training phase. In addition, training is noticeably faster in Word2VecF since extracting the tuples is preprocessed in advance, along with the vocabulary count and subsampling.

I	like
I	Icecream
I	Icecream
...	...
I	document_id_5
like	document_id_5
...	...

Table 2.1: By controlling the input to Word2VecF, specific tuples can be repeated to add weights/bias to the resulted embeddings. Arbitrary context can be added to add a global context

Constructing the document vector can be done with weighted average of document's word vectors. The difference here however, is adding arbitrary context words, like document Ids, we can force words in a single document to predict the document vector instead of adjacent words. This is an example of how global context can be exploited, words are no longer tied to its neighboring words in the window as much as all words in the document.

Word2Vec - Window size 10	**Word2VecF - Global Context**
selenium	bug
test	programming
manual	jenkins
testautomation	jira
testing	assurance
testmethod	unix
...	...

Table 2.2: Word2VecF shows more words that appears in the same document as "Quality Assurance", while Word2Vec shows words that can take the same or a neighboring place as "Quality Assurance".

As shown in Table 2.2, when similar words are queried for "Quality Assurance" as an example, Word2VecF tends to retrieve words that appear in the document associated with Quality Assurance, like jenkins and programming. Where Word2Vec model retrieves

words that can appear in the same location in the context window like testing and selenium.

2.3.3 Doc2Vec

Both previously described methods (Word2Vec and Word2VecF) produces word vectors, which get averaged to represent the document vector. In order to directly represent documents, Le and Mikolov [2014] introduced Doc2Vec, where every paragraph has a unique vector in the matrix D and every word has its own vector in matrix W (same local context architecture as Word2Vec). These vectors are averaged and combined to predict the next word in the context in a given paragraph.

The paragraph vector is only shared among words of the same paragraph. It can be represented as another word in the context that is fixed for all sentences and windows in the paragraph. Hence it preserves (or memorize) the topic of the paragraph. That's where the architecture name got its name "Distributed Memory" shown in figure 2.5.

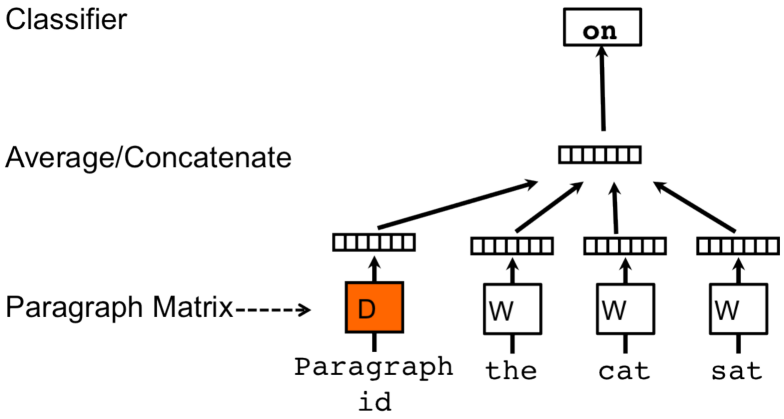


Figure 2.5: Doc2Vec Distributed Memory model uses the paragraph vector along with the local context words to predict the word $w(t)$, it also acts as a memory of the paragraph's topic [Le and Mikolov 2014]

Distributed Bag-of-Words (DBOW) however, ignores the context window and word vectors computation, as it forces the model to predict randomly sampled words in the document given the document vector as shown in figure 2.6. DBOW only updates the paragraph vector, so it needs less storage as it ignores word vectors.

The drawback of Doc2Vec is its computational complexity, as the paragraph matrix increases in size with respect to number of documents, due to the absence of negative sampling approach in updating document weights, all documents are involved in the learn-

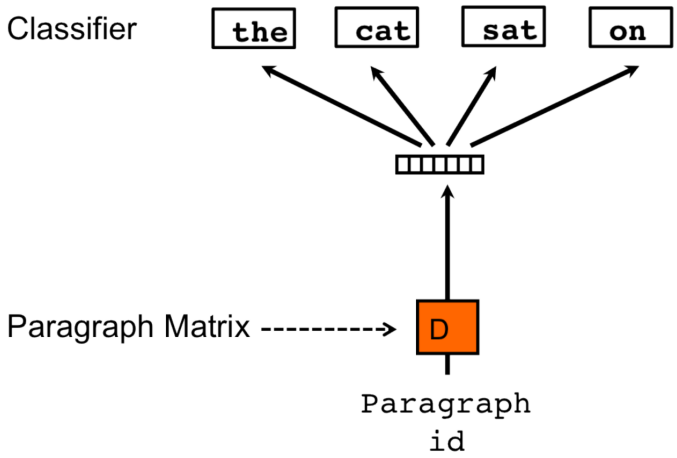


Figure 2.6: Doc2Vec Distributed Bag-of-Words model tries to predict a randomly sampled set of words in the paragraph given the paragraph vector [Le and Mikolov 2014]

ing process. Moreover, to generate vectors for unseen documents, the documents have to be added to the corpus and model training has to be restarted from the beginning, which is not feasible in the industry since it doesn't scale well [Chen 2017].

2.3.4 Doc2VecC

In order to tackle the problems in Doc2Vec, Chen [2017] presented a new approach to include global context by capturing the semantic meanings of the document during learning called Doc2Vec Corruption (Doc2VecC). Figure 2.7 shows the architecture of the model, which is very similar to Word2Vec's, except that on every learning iteration, words are randomly sampled from the document (hence, corrupting the document) and their vectors are averaged to represent a document vector, that is used to predict the current word with the help of the local context words.

The output of Doc2VecC is a vector representation of words that shares global and local semantics of the dataset. To generate a document representation, we compute the average of the word vectors. This results in a better representation than using embeddings from word2vec (see Section 2.3.1). In addition, using the average of word vectors enables the generation of document representation for documents that have not been seen during training. As the method uses only a fraction of the words inside the document to construct the document vector during training, the training time is further decreased.

Previously discussed semantic models generate a N-dimensional dense vector representation for documents, often uninterpretable and can't be visualized. We believe it would

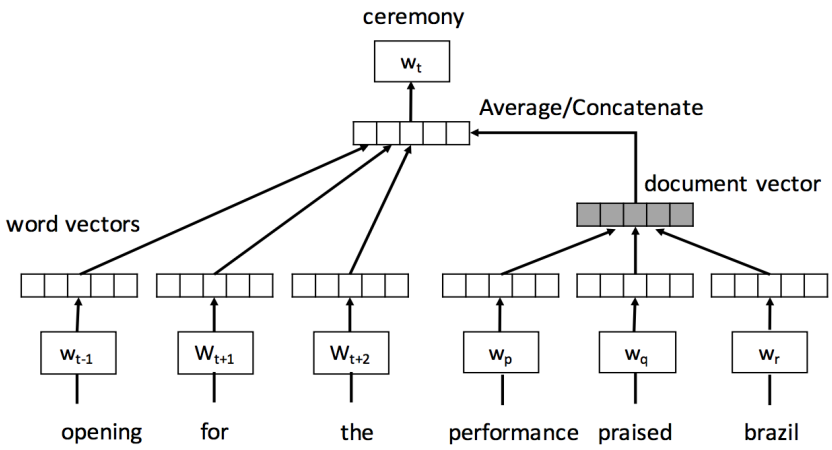


Figure 2.7: In every iteration in the Doc2VecC model, words are randomly sampled (depicted in w_p, w_q and w_r) from the document to represent the document vector which is used beside the local context to predict the current word w_t. While training, the error is back-propagated to all local and global words used [Chen 2017].

be useful to explore different type of interpretable vector representations like topic modeling.

2.3.5 Latent Dirichlet Allocation

Unlike context based semantic models, topic modeling provides a representation of documents based on the topics, implicit ideas or themes discovered across a document collection [Blei and Lafferty 2009]. Latent Dirichlet Allocation (LDA) is a topic modeling algorithm which is used to detect the distribution of topics in a certain document. LDA starts training by defining a number of topics k representing the number of underlying topics in the dataset. Each topic is represented as a multinomial distribution over all words in the vocabulary. A document (as a vector) can be represented as a distribution of the k topics [Blei et al. 2003].

As shown in Table 2.3, topics reflects a certain latent subject in the corpus. For example, *cortex*, *vision* and *neuron* can represent Opthalmology, while *infection*, *aids* and *vaccine* represents *immunology* and so on. A document vector will consist of a probabilistic distribution of all topics. For example, a document discussing white blood cells will have a document representation vector of 90% immunology, 8% chemistry and 2% opthalmology.

Each topic is also represented as a distribution over all words in the corpus. Table 2.4

computer	chemistry	cortex	orbit	infection
methods	synthesis	stimulus	dust	immune
number	oxidation	fig	jupiter	aids
two	reaction	vision	line	infected
principle	product	neuron	system	viral
design	organic	recordings	solar	cells
access	conditions	visual	gas	vaccine
processing	cluster	stimuli	atmospheric	antibodies
computer	chemistry	ophthalmology	astronomy	immunology

Table 2.3: Most weighted words within Five topics from a 50-topics LDA output trained on *Science* magazine [Blei and Lafferty 2009]. Documents can be represented as a distribution of topics (last row is annotates the topics for elaboration). For example, a document about Cataract will have a vector of 90% opthalmology and 10% immunology.

shows the distribution of words in two topics in our dataset to show the contribution of each word in the topic. For instance, the word *data* appears in the top words of both topics, however the contribution of data in Topic A which talks more about big data and statistics is bigger than its contribution in Topic B, which is related more to business intelligence and solutions.

Topic A	**probability**	**Topic B**	**probability**
data	*0.161*	business	0.230
statistics	0.095	intelligence	0.094
big	0.054	warehouse	0.079
mathematics	0.054	business_intelligence	0.078
big data	0.046	solutions	0.038
analytics	0.043	*data*	*0.032*
data mining	0.034	reporting	0.025
mining	0.030	tool	0.020

Table 2.4: Probability distribution of words in two topics in our dataset. Words with highest probability should give an interpretable theme of the topic. Topic A is more related to big data and analysis, where topic B is more into business intelligence and reports.

Unlike Word2Vec and other semantic models, LDA doesn't take words order in perspective, as it tries to summarize the document into a collection of topics in a dimensionality reduction fashion, where the output is a global representation of the document. Distributions for unseen documents can be generated using the model, and they can be compared to have a similarity value between documents.

2.4 Scoring, Similarity and Evaluation Metrics

In this section, we will go through TF-IDF as a weight function that can be integrated with weighted average semantic models. In addition, we describe two similarity metrics, Cosine and Hellinger Distances, to explain their significance to semantic and topic models. Finally, we discuss the main evaluation metric in the thesis, the F1 measure, that is used to compare the model performances during the experiments.

2.4.1 TF-IDF

TF-IDF is a scoring function which consists of the product of two parts: the term frequency (TF) and the inverse document frequency (IDF). Term Frequency (TF) of a given word is how many times the word appears throughout the document. While Inverse Document Frequency of a given word is the count of documents containing this given word. Given N representing the total number of documents in the dataset, word w appears in n number of documents. Where $f(w, d)$ is the frequency of the word w within a given document d, then the *TF-IDF* of word w is calculated as follows [Robertson 2004]:

$$\textit{TF-IDF}(w, d) = f(w, d) * log\frac{N}{n} \tag{2.3}$$

TF can't be used on its own as a weighting function as words with high *TF* scores are usually stopwords. *IDF* component should be used to normalizes the values of TF, as it increases the score of words that are distinct to the document, and punishes words that occurs in all documents. *TF-IDF* can be used for weighted average of word vectors, since words that are unique to the document will have more priority during average than stopwords and noisy ones.

2.4.2 Similarity Metrics

Once the documents are represented as vectors, similarity measures are used to compute the similarity between two given documents. Tasks like clustering or finding nearest neighbors can be used to perform using the similarity value between documents in the corpus. There are plenty of similarity measures like Jaccard Coefficient, Euclidean distance, Cosine distances and more [Bordag 2008].

Cosine Similarity

One of the standard measure for document and word similarity is the cosine similarity [Wan 2007][Dai et al. 2014][Mikolov et al. 2013a]. It shows how two vectors are related in terms of the vector angle instead of the magnitude. Given two document vectors v_1 and v_2, their cosine similarity value is calculated as follows [Shoaib et al. 2014]:

$$SIM_C(v_1, v_2) = \frac{v_1.v_2}{|v_1| * |v_2|} \tag{2.4}$$

where v_1 and v_2 are m-dimenstional vectors that represent the document, and the result is a non-negative similarity score between -1 and 1, where a similarity score of 1 means that v_1 and v_2 are identical.

Hellinger Distance

For memory performance reasons, LDA implementations sets probabilities lower than a certain threshold to zero, converting the LDA output to a sparse vector, the cosine similarity may not be the best metric for evaluation. But the sparse LDA output vector can be represented as a probability distribution, which allows us to extend vector similarity measures to include distributions similarity techniques, like Hellinger Distance [Blei and Lafferty 2007][Seroussi et al. 2011][Rus et al. 2013].

Given the distribution of topics of documents i and j, Hellinger distance can be computed as follows:

$$d(\theta_1, \theta_2) = \sqrt{\frac{1}{2} \sum_k (\sqrt{\theta_{1,k}} - \sqrt{\theta_{2,k}})^2} \tag{2.5}$$

where θ_i is the topic distribution of document i, and $\theta_{i,k}$ is the probability of the k-th topic of the document. The result, unlike cosine similarity, is a distance where 0 denotes identical distributions and 1 denotes very dissimilar distributions or documents.

2.4.3 F-measure

In information retrieval, the performance of different models can be compared using metrics like Precision, Recall and F-measure. Precision shows the fraction of retrieved documents that are relevant. While Recall measures the fraction of relevant documents retrieved from all relevant documents in the dataset.

$$precision = \frac{relevant\ documents \cap retrieved\ documents}{retrieved\ documents} \tag{2.6}$$

$$recall = \frac{relevant\ documents \cap retrieved\ documents}{relevant\ documents} \tag{2.7}$$

As precision and recall have an inverse correlation, mostly a combination is used for optimization. Recall will score 100% if all relevant documents are retrieved plus infinitely many irrelevant ones. Precision will score 100% if the model managed to retrieve - for example - only 5 relevant documents out of infinitely many relevant ones.

The F-measure (or F1-score) is defined as the harmonic mean between precision and recall. The harmonic mean is used because we are dealing with ratios and percentages, "As It tends toward the least number, minimizing the impact of large outliers and maximizing the impact of small ones [Nadeau and Sekine 2007]". In other words, both Recall and Precision have to be of a high value to have a high F1-score, making harmonic mean a better fit than the arithmetic one.

$$F1\text{-}score = 2 * \frac{precision * recall}{precision + recall} \tag{2.8}$$

3 Methodology

This chapter gives an overview over the experimental design and the offline evaluation in particular. The online evaluation is discussed in Chapter 5. In this chapter, we explain how the dataset for the offline evaluation was collected and the detailed processing steps followed to clean the job descriptions. We discuss how we built the ground truth dataset. Later in the chapter, we describe the evaluation method used for the document similarity task. We start by explaining why conventional machine learning models built on top of vector spaces are not feasible in our case, then we describe model evaluation using the entire latent vector space and its limitations. Finally, we present a different evaluation approach using Positive and Negative Sampling that avoid of the limitations of full vector space.

3.1 Experimental Setup

We hypothesize that documents are similar the closer they are to each other, and dissimilar the further they are to each other in the N-dimensional latent space. Different methods of document embeddings will generate different vector spaces, which can be analyzed, tested and compared to each other. Therefore, every other variable has to be fixed except the vector space for fair comparison. The evaluation work-flow includes multiple steps as shown in Figure 3.1.

- Documents are passed by a data cleaner module which preprocess the data, stem and lowercase the text, remove special characters, among other regular expression filters that will be explained in details later in the chapter.

- A semantic model trains on the cleaned preprocessed dataset in order to generate document representations for job postings in the latent Space. This step is the variable in our experiment, as different semantic models and hyperparameters discussed in Section 2.3 will be evaluated.

- When a given document d_1 in the ground truth is evaluated across the latent space, a set of similar documents will be retrieved according to how close they are to d_1. These similar documents can be evaluated using different evaluation datasets to get a performance measure of how well are the documents are placed in our latent space.

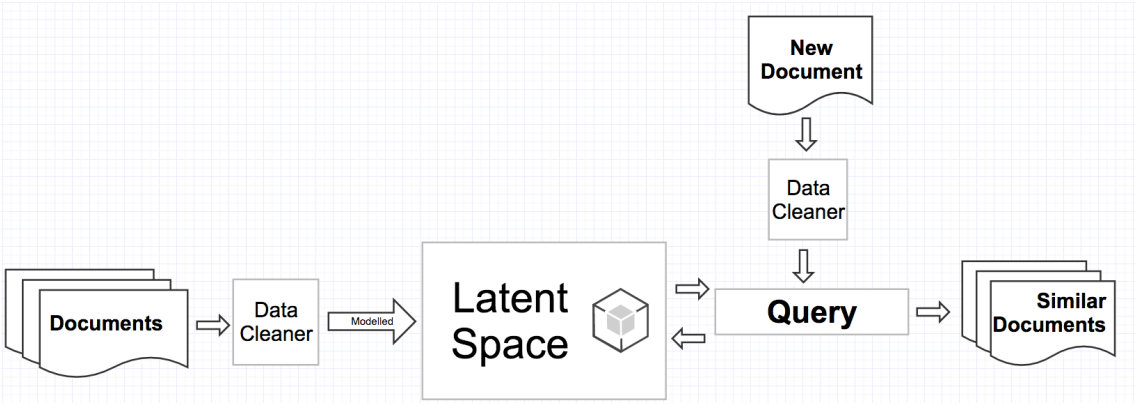


Figure 3.1: i) Documents are preprocessed before getting embedded into the latent space. ii) Given a document in the evaluation dataset, the latent space will retrieve the nearest (most similar) documents. iii) Retrieved documents can be compared to the actual similar documents in the dataset to calculate a performance score.

3.2 Offline Evaluation Dataset

In this section, we describe the data collected for our experiments and list interaction statistics like total number of bookmarks, replies, users and job postings. Furthermore, we explain the steps taken to clean the job posting descriptions like stemming, and removing URLs and special characters. Finally, we explain the ground truth construction and the intuition behind using the user interactions to build the dataset of similar job postings.

3.2.1 Data Retrieval

User interactions with the platform were analyzed and used as follows: We collected user interactions (bookmarks and reply Intentions) from March 2014 to March 2017 as (User, Job) pairs. Then, we perform several filtering steps, removing less active and excessively active users. First, we remove users and jobs that have less than two interactions. In a second step, we filter out users with overall lifetime interactions that exceed the 99th percentile of all users, which we consider as outliers.

Clicks data were not used as part of the dataset. Users can click on job postings because the title is intriguing or they want to explore new career options. Therefore, we don't consider clicking data of a given user as relevant to him as jobs he clicked, read the description and shown interest in the posting by bookmarking it for the future or informing the recruiter that he or she is interested.

Dataset	Total Tuples Count	Number of Distinct Job Postings	Number of Distinct Users
Replies	218,518	56,545	58,844
Bookmarks	397,805	102,721	79,620
Distinct Aggregation	616,323	129,156	110,417

Table 3.1: Statistics of retrieved data after cleaning outliers. Note that users sometimes bookmark and reply on the same posting, that's why the aggregation values are not absolute summation.

Most interactive users on the platform prefer German job postings, which represents 91% of the dataset collected. In theory, training a model in two language should put the words in separate parts of the vector space, as the context is different for each word in each language. In order to avoid the need of more dimensions required to represent several languages, we train all models using solely German job postings.

Portable Document Format (PDF) job postings were not used as part of the training set. Since most PDF readers parse the text line by line horizontally, which gives a wrong context since 2 columns documents are read out of order. However, they were used in the A/B Test, since the line order is not significant with the choice of model used.

Table 3.1 shows the size of the dataset after removing outliers, English postings and users with a single click and PDF jobs. Distinct Aggregation lists the number of distinct job postings and users from both datasets, since many users tend to simultaneously reply and bookmark the job posting. Finally, the dataset consists of around 129,000 job postings from around 616 thousand tuples.

3.2.2 Data Preprocessing

Before we train the models, multiple preprocessing steps were followed over the description of 129,000 available job postings in our dataset:

- HTML tags were removed using regular expressions.

- Initial models training resulted in bigger vocabulary size due to several representation of the same word. for instance, vocabulary contained *Java*, *java* and *JAVA*, which the model sees them as three different words. Hence, job postings got lowerecased.

- Removed all URLs and emails via regular expressions. When analyzing early mod-

els, there were huge bias towards HR contacts emails and job agencies that include boilerplate URLs in the job description footer.

- Numbers were replaced with a placeholder *"numb"* [Abdelwahab and Elmaghraby 2016]. In order to preserve context in the window size while training Word2Vec. For example, "since 1994" will be "since numb". In this way, numbers will always preserve its place and its relation to its neighbors in the local context. However, only stand alone numbers were removed, since numbers are important for keywords like *J2EE* and *B2B*.

- Umlauts in the German language were normalized. for example, *für* to *fuer* to help removing special characters in the next step.

- All special characters like non alphabets, question marks and bullet points were removed. There is no unified encoding for job postings as they come from different sources, and most topic modeling libraries accept only *utf-8* encoding.

- White spaces got *trimmed*, and the entire document got *stemmed* using snowball stemmer, which gave better results as shown in Chapter 4.

Early preprocessing decisions involved normalizing the top occurring alphanumeric words, like B2B to BtwoB and Html5 to Htmlfive. Due to the fact that HR sometimes stamp the job description with alphanumeric codes (e.g 000003vw). However, the step is not needed - and not used in later experiments - since both TFxIDF and Word2Vec's "minimum count" threshold would inhibit or remove noisy words.

NLTK sentence tokenizer [Stamatatos et al. 1999] was used in the early thesis experiments to identify sentences in the job posting, the approach kept useful html tags like <*br*>,<*h1*> to <*h5*> and <*li*>, <*ul*> as sentence splitters. Then the sentence tokenizer exploits commas, full stops, question marks and sentence length to identify when to end the sentence using rule based approach (hence, avoid splitting over abbreviations with full-stops like *e.g*).

The intuition was to feed the Word2vec sentence by sentence instead of one sentence document approach which - assumingly - could lower embeddings performance by adding irrelevant context. For example, end of sentence concatenated with beginning of the next sentence. However, Word2Vec favors contexts that happens more often in different locations around the corpus (with increase of number of iterations), and the issue can take place if the two sentences always comes behind each other, one can argue that these two sentences can be fitting contexts for each other. Sentence Tokenizer was not used in production to reduce computational overhead.

3.2.3 Ground Truth Construction

A open source dataset of similar job postings is not available. As we want to exploit the sentence structure and vocabulary of job postings, we can't use a general open source dataset with a broad scope to train a model that would run on a very specific narrow scope like job postings. In order to evaluate and compare different job posting embedding models, a dataset set of job postings with similarity scores has to be used. It can be in the form of a relational matrix that shows a similarity value between every document and other documents in the dataset, or simply a boolean value that shows whether two or more documents in a dataset are similar or not [Le and Mikolov 2014]. Given that creating a similarity matrix by human experts is expensive and time consuming, as experts would have to go through 129,000 jobs times 129,000 times for completeness, we use information from users at XING to create the dataset.

Assumption: If two or more users show interest in two jobs. These two jobs are similar.

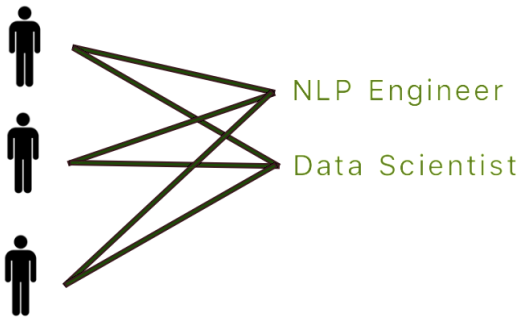


Figure 3.2: An example for users co-interactions. If three or more users bookmark two jobs for future notice, these two jobs are most likely similar

Our intuition is that users in general bookmark jobs that are relevant for them and in theory should be similar. However, this information can also be erroneous, due to misusage of the platform, like mis-clicks or saving for a friend. Hence, by selecting only jobs where several users agree on, we can increase the probability that these jobs are similar.

In order to prove the assumption, a proper representative sample should be randomly selected and asserted by human experts. However, sampling even 1% of the dataset would result in a dataset sample of 6,000 similar jobs tuples to be asserted. Due to time constraints a manual hypothesis testing would be intractable. Thus, we compare the meta-data from the job postings. For example, for 616,000 tuples of similar jobs 70.02% of them share the same discipline Id.

The co-occurrence matrix has been plotted and normalized by rows to show the distribu-

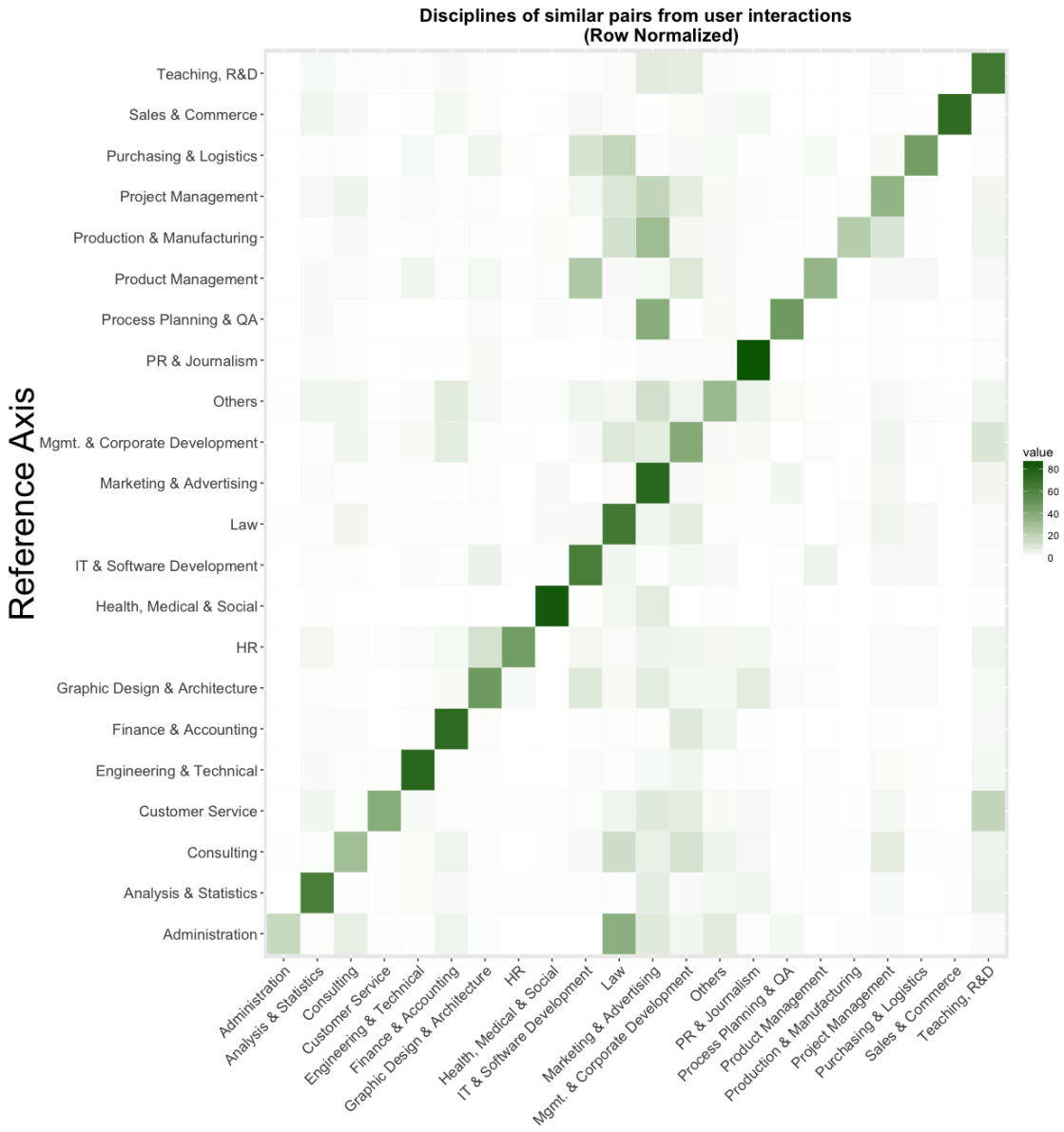


Figure 3.3: Graph shows relation between disciplines in our co-occurrence ground truth. Note that the matrix is row normalized and not symmetric. It can be read using the reference axis as follows, for example, first row's "*Teaching, R&D*" has most of its discipline distribution across *Teaching,R&D*, *Marketing & Advertising* and *Management & corporate development*. The aim is to spot any odd discipline combinations that may result from collecting the ground truth from user interactions. Legend ranges from 0% "white" to 100% "dark green" that depicts the percentage of job disciplines co-occurrences for a given discipline.

tion of disciplines without any frequency bias. As shown in Figure 3.3, most of the similar pairs have the same disciplines. We also see high correlations between Marketing, QA and Project Management, which is obvious, as these categories have high topical overlap. Further relationships can be spotted, e.g. between Consulting, Law and Corporate Management, Product management and IT, as well as R&D and Corporate development.

There are some disciplines that may not considered very similar, like *Engineering & Technical* and *Management & Corporate Development*, which is expected noise as a result of not hand-crafting the similarity pairs. Users aim for having slight diversity in the jobs they bookmark. However, such non trivial discipline combination have very low values depicted in the figure in pale green. Better solutions involve increasing the number of users who co-interacts with the job, to increase confidence, but it will decrease the size of the dataset drastically and skews the distribution of disciplines available in XING's metadata.

3.3 Latent Space Evaluation

The document embeddings models presented in Section 2.3 are used to train and embed the documents into a vector space. Then a supervised learning algorithm can be used on top of the vector space to either predict a similarity value between documents as a regression algorithm [Nagaraj and Thiagarasu 2014] or classify the job posting into a certain class as a classifier [Wajeed and Adilakshmi 2011].

Using a classifier requires the dataset to have classes or labels. The ideal approach is to classify the job into a certain category, assuming that all jobs in this certain category are similar. XING metadata like disciplines, industries and cities can be used to deduce classes but it adds more challenges. For instance, the exponential number of classes generated from metadata, using 23 disciplines and 22 different industries, will result in 506 classes. The assumption that job posting pairs from different disciplines are not similar might be a good indicator. However, for related disciplines this is often not the case, as shown in Figure 3.2.

Using a regression algorithm however requires a similarity score between job posting pairs, as shown in the similarity score attempt in Appendix A.3, getting a consistent similarity score between job postings is impossible without using a group of human experts to manually evaluate the similar pairs, making the regression algorithm approach obsolete.

3.3.1 Full Latent Vector Space Evaluation

Since a supervised algorithm is not feasible in our case, we use an unsupervised $K-$nearest neighbor algorithm. The approach was followed initially in this order:

- Train the desired model using given hyperparameters.
- Sample N jobs from the dataset, along with their similar job postings.
- Use the model to create document vectors for the sampled jobs. Insert them into the vector space.
- For each Job J_i, get K-Nearest jobs $J_{1..k}$ given that the similarity measures between J_i and every job in $J_{1..k}$ are within a certain threshold.
- F-measure can be used to compare similar job postings of J_i from the dataset (relevant) and the retrieved jobs from the vector space ($J_{1..k}$). F-measure of all N jobs can be averaged to give a performance estimate of the model.

In order to get the K-Nearest-Neighbors in a vector space, all items have to be compared to all items to calculate cosine similarity, sort by the value and then cut off at threshold. This is an expensive $O(N^2)$ operation [Maillo et al. 2017], as a sample of 5000 job postings will have to conduct 25 million similarity calculations.

A popular approach however, is to index the space using a tree data structure, such that every branch in the tree is used to search a small portion of the vector space. These branches are constructed in a hierarchical fashion to the tree root while indexing, which is a one time operation that takes plenty of time, but offers faster query times by not traversing specific branches of the tree in Nearest Neighbor search.

KD-Tree vs Approximate Nearest Neighbor

KD-Tree partitions the vector space by recursively generating hyperplanes and splitting at the points where there is a maximal variance in the data [Bentley 1975]. Unfortunately, Lee and Wong [1977] show that KD-Tree, with query complexity of $O(k.N^{1-\frac{1}{k}})$, doesn't scale well with high dimensional vectors k.

A faster way to generate hyperplanes is by using random projections to build up the

tree in what is called Approximate nearest neighbor [Arya et al. 1998], where random hyperplanes are chosen for the split. It is a tradeoff between precision of the nearest neighbor results and query time performance. Approximate nearest neighbor algorithm doesn't necessarily returns the nearest neighboring points in the vector space.

ANNOY

ANNOY [1] is a python library with Approximate Nearest Neighbor implementation from Spotify. It offers querying functionality using cosine distance that fits most of our models. It also offers faster query times with moderate resources management, since data (mmaped into memory) is shared across multiple processes.

Since ANNOY uses random projections to index the vectors, the same *Random Seed* value - used to initialize the pseudo random number generator - has to be set to get reproducible and comparable results. Number of trees used for indexing is responsible for the tradeoff between percision of nearest neighbor output and the time taken for indexing. The more trees used, the more accurate the results going to be but the longer time it will take for indexing. Finally, queried job postings from ANNOY should be filtered by a threshold to avoid getting irrelevant non similar job postings, if the job to be queried is in a distant point in the vector space.

After setting a Random Seed value, number of trees and a threshold, nearest neighbors for job postings can be queried for different vector space modeling algorithms for comparison as shown in Section 4.4.

Limitations

Jobs with titles *Java Developer, Hamburg* and *Java Backend Developer, Stuttgart* are examples of two very similar job postings but in different locations, and therefore they suit two different types of users, those who live in Stuttgart and those who live in Hamburg. According to our dataset rules, if no three users co-interacted with these two jobs, they will not be considered similar in the dataset. Same issue appears with similar job postings with relatively big time interval between them. For instance, *Accountant* (posted in 2014) and *Audit* (posted in 2017), users who interacted with the first job may have either been inactive on XING three years after or rather not interested in a similar job they interacted with three years ago.

[1] https://github.com/spotify/annoy

To conclude, the dataset contains similar postings, but it does not contain a similarity score between every job and another in the dataset. By putting all the jobs in a single vector space, similar jobs are retrieved that gets punished by the incomplete dataset that we are using. In other words, we do not want to capture the co-interaction behavior of users as much as the document similarity itself, which requires a slightly different evaluation approach.

3.3.2 Positive and Negative Sampling Dataset Evaluation

In order to avoid the limitations of using the full latent vector space, we have to evaluate the models on a dataset of narrower scope, which can be achieved by generating smaller datasets for every job posting. In order to assert their approach in information retrieval tasks, Le and Mikolov [2014] created their own dataset of triplets, consisting of Paragraph ***A***, similar Paragraph ***B*** and a third paragraph ***C*** that was randomly sampled from the rest of the collection as a non similar paragraph. Inspired by the authors, Negative Sampling in Word2Vec architecture and Cross Validation [Stone 1974]. We extended the approach to construct the positive and negative Dataset in this order:

- Set a Random Seed value to obtain to obtain reproducible results, and comparable across different models.
- For every job J_i, randomly sample N similar jobs from the dataset (Positive Jobs), and sample tN jobs from the entire dataset (Negative Jobs) to create a test-list of job postings for the given Job.
- Create K lists of randomly sampled positive and negative job postings for each job and shuffle all lists.
- Reorder every list using cosine similarity value between each job posting in the list and its corresponding job posting J_i.
- F-measure can be used to compare the lists cutout at N (retrieved), and the relevant job postings from the dataset. F-measure of all lists can be averaged to give a performance estimate of the model.

The K lists generated can be viewed as $K-$folds in cross validation, where F-measure can be averaged out on the fold level to compare models statistically, or just have an average over all folds to give a comparable performance value for the model.

By "Negatively sampling" job postings from the entire dataset, we reduce the chance of fetching similar job postings that our dataset did not capture. There is a chance of having similar job postings labeled as negative/not-similar, but that's one of the reasons K lists for each job is randomly sampled, so errors can be averaged out. Which follows the same concept of negative sampling in the Word2Vec architecture, as the model works by repetition, chances that similar words (in our case job postings) that gets sampled out from the entire corpus (dataset) is low.

During the experiments, we chose $t = 4$ and $N = 10$ to have 50 jobs per list. Increasing the ratio of negative samples to positive ones pushes the model to behave more like the Full Vector space evaluation, as more negative samples will eventually retrieve similar documents.

Unfortunately, by using negative sampling approach for picking non similar jobs, we can't be certain that all of them are non similar. Hence, the ideal model won't have 100% F1 score on this dataset.

Visualizing 500 dimensions is impossible, but using some dimensionality reduction algorithms like *T-SNE*, we can display the relationship between items in the vector space on a 2D or 3D plane. Figure 3.4 shows one of the sampled lists in Positive and Negative dataset approach, where red colored job represent the job being evaluated, and the blue colored ones represent the positive similar job postings sampled from our user interactions, and the rest of the jobs (in black) depicts the non similar jobs sampled from the entire corpus. In the figure, we can already notice multiple observations:

- Most of the blue similar jobs are already encapsulating the job being queried, where they all discuss the same topic, Java Development.

- Some of the black non similar sampled jobs are very relevant to Java development, we see *FrontEnd developer* and *Teamleader in IT Development* on close proximity to the Java cluster.

- There are multiple clusters forming already. For instance, the top right corner in the figure has a cluster of Media Management, which happened to get sampled as negative dataset for our job being evaluated.

Hence, our goal is to have a document representation where all the (blue) similar job postings are as close as possible to the one being evaluated (red), and at the same time, as further away from all non similar (black) job postings. When evaluation takes place, a nearest 10-neighbor will take place to query the 10 nearest job postings to the job being

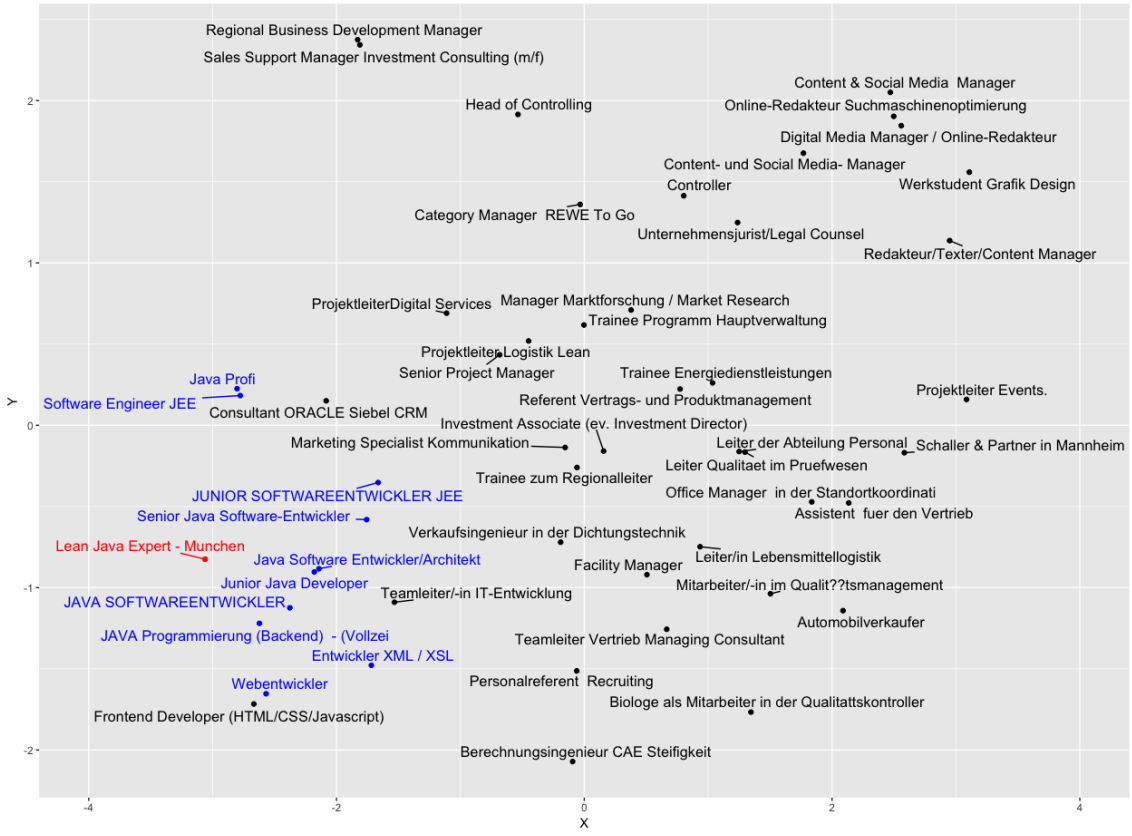


Figure 3.4: After generating job posting embeddings, the dataset can be visualized using a dimensionality reduction algorithm (T-SNE). The figure shows one of the sampled list in Positive and Negative Dataset, where the red job posting is the one being evaluated, blue job postings depicts the similar Positive job postings, and the black ones represents the negative non similar sampled jobs from the entire corpus.

evaluated. We achieve maximum F-measure if the top 10 jobs (in the example) in the sorted list are the 10 positive job postings we sampled earlier, or minimum F-measure if none of the positive job postings appeared in the top 10.

The subsequent chapter will use the two offline evaluation datasets covered to compare the performance of different model architectures, hyperparameters and feature combination attempts.

4 Offline Evaluation

In this chapter, the results are shown to justify the choices of the hyperparameters chosen, as well as the experimental results of the comparison between different document embedding models. Positive and Negative Dataset was used for all the experiments, for both parameter tuning and models comparison, where the entire latent vector space were only used in models comparison to make certain both datasets shows the same behavior. In other words, did we create any biases while creating the Positive and Negative Dataset?

4.1 Semantic Models Hyperparameters

There are plenty of hyperparameters to tune in Word2Vec model. Researchers exploit the hyperparameters that best suit their needs on their given datasets [Levy et al. 2015][Ji et al. 2016][Yao et al. 2017]. However, in order to test all combinations of the most used hyperparameters by researchers to find out the best performing ones on our dataset, we would have to conduct over 190 experiments.

First Experiment: Lowercase

We perform a sweep over the parameters in a iterative fashion: The first experiment was to find out whether **Lowercase** transformation can produce better results. 10% of the dataset were sampled and the experiment was carried out while fixing all other hyperparameters as follows: 500 Vector dimensions, window = 10, sampling rate of 1e-5, negative sampling 15 and minimum count of 5. Documents were shuffled to avoid structured bias [Melamud et al. 2016] that results from documents that may be trained in the same order.

As shown in Figure 4.1, we obtain consistent improvements for the F1 score at different ranks K than the model trained by the unprocessed text by a maximum F1 score improvement of +4.0 on the training sample.

Lowercase transformation is a common practice in Natural Language processing [Reynar 1999] and information retrieval applications [Jansen et al. 1998]. The skepticism however, came from the fact that German language uses capital letters to denote nouns. Hence,

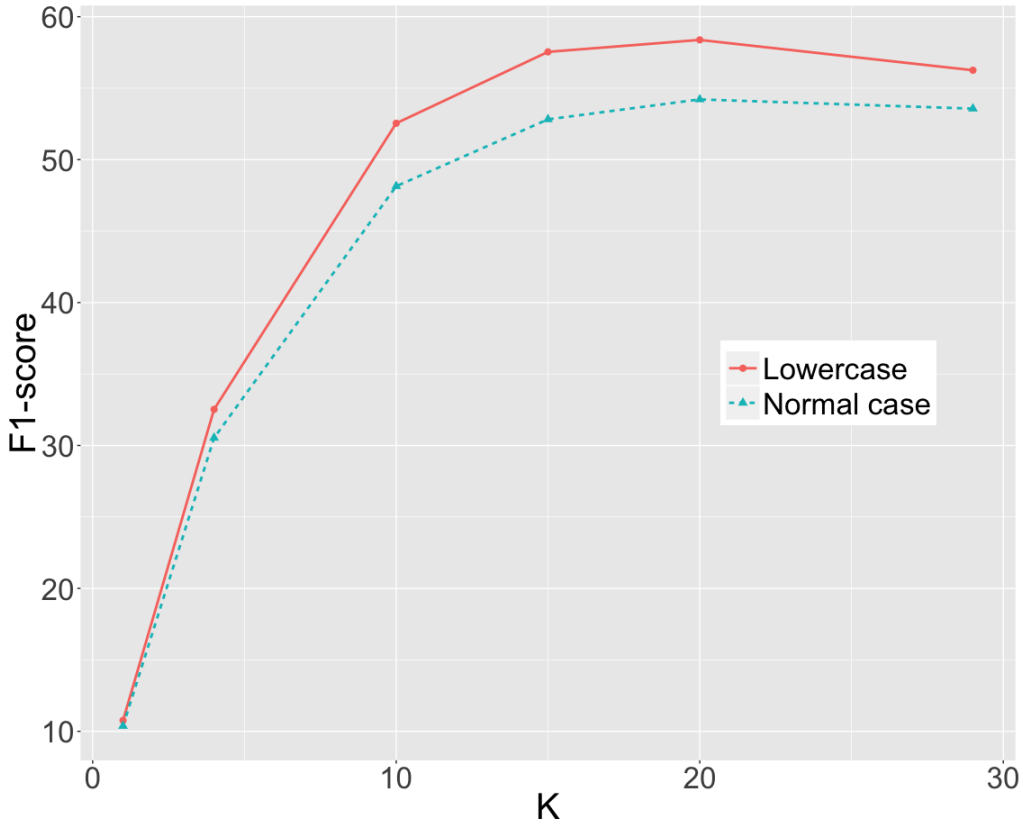


Figure 4.1: Training the model with lowercased text consistently scores better F1 values than normal case text across the K ranks of the retrieved items.

given a clean text, nouns should be the most important words to distinguish the document, which is not the case in our dataset. Multiple occurrences of the same word appears with different representations, because there is no clear format for job postings, and sometimes to direct attention to certain words, like *accountant*, *Accountant* and *ACCOUNTANT*, where the model treat them differently, with different - and similar - vector representations. However, it reduces the model's accuracy overall, since context of the same word will be shared among all its different representations.

Second Experiment: Model Hyperparameters

After lowercasing the job postings' description, another experiment was conducted to decide which Word2Vec architecture will be used (Skip-gram or Continuous Bag-Of-Words). In addition, we cross validated different window sizes *(w = 1,5,10)* in the same experiment as shown in Figure 4.2.

The graph shows significant difference between F1 scores of CBOW models (denoted as

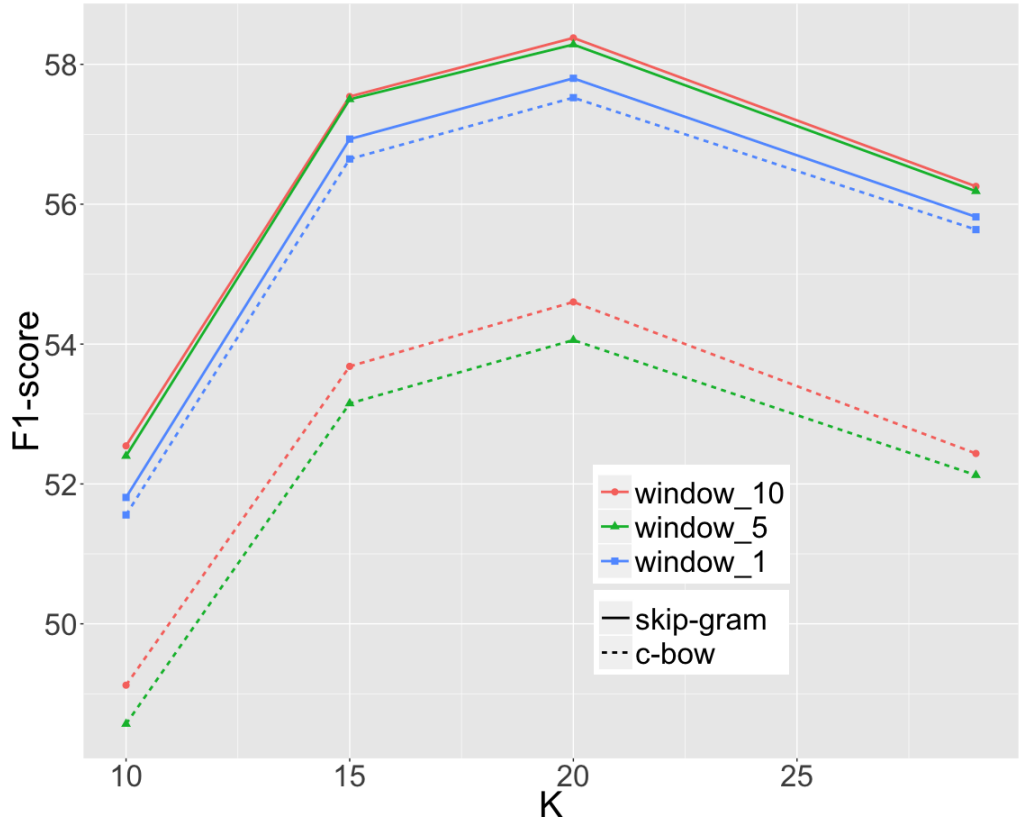


Figure 4.2: Skip gram models performs generally better than CBOW models, and the F1 score goes higher as window size increases.

dashed lines) and Skip-gram models (denoted as full lines). F1 scores also vary across different window sizes of the same model architecture. In Skip-gram model, the bigger the window, the higher the F1 scores, as a window of size 10 captures broad topical content than a window of size 1. Which is not true however in the CBOW architecture, as window of size 1 performs 3% better than the model with window size 10. We believe the drop in performance due to increasing window size in CBOW is caused by the averaging (smoothing) of context word vectors before assigning the vector for middle word, which is not the case in Skip-gram models that scales well with window size. One more reason for Skip-gram performance is its sensitivity to infrequent and rare words [Mikolov et al. 2013b] that possibly constitute the main features/keywords of the document.

Skipgram Model with window size of 5 performs slightly worse than window size of 10. A window of size 5 can be used as a performance/training speed trade-off, since adding more context to words increases training time [Mikolov et al. 2013b]. However, Skip-gram model with window size of 10 training time is afforded while experimenting, and hence the model is used to conduct the rest of the experiments. We used a window that includes 21 words (10 + 1 + 10) which is not far from the average document word count in the corpus of 29 words per document.

Levy et al. [2015] attempted to tackle the most beneficial hyperparameters in the Skip-gram model, window size was put under test with values of 2,5 and 10. Where window sizes of 5 and 10 performs equally better than window size of 2 (similar results to our experiment).

Different negative sampling values were also experimented in the paper which yielded better results for values 5 and 10 than negative sampling value of 1. Levy stated in the practical recommendations part that it's preferable to use many negative samples in the Skip-gram negative sampling architecture. Confident by the similar finds in our experiment and Levy's regarding the effect of window size, Negative sampling of value 15 was used in the thesis.

Based on the experiments conducted on model architectures and window sizes, along with the hyperparameter values discussed and justified in Section 2.3.1, Word2Vec hyperparameters used in upcoming experiments can be summarized as follows:

```
Word2Vec -size 500 -window 5 -sample 1e-5 -negative 15 \
         -threads 1 -iter 20 -min-count 5
```

4.2 Semantic Models Comparison

Once all hyperparameters are picked for the models, they can be compared to each other on the same dataset. F1 score @ 10 is picked for as comparison criteria since we have 10 positive similar job postings in each list, which can be interpreted as an average percentage of jobs in the top 10 which are actually similar. For example, a model has 70% F1 score @ 10 can be visualized that on average, 7 out of 10 jobs are similar. Note that 100% F1 score is almost impossible because of the limitations listed in Section 3.3.2.

Model				**F1 score@10**
TF-IDF				08.69 %
Word2Vec				54.84 %
Word2Vec	+ stemming			56.22 %
Word2VecF	+ stemming	+ Document Context		61.12 %
Word2VecF	+ stemming	+ Document Context	+ TF-IDF Weights	62.81 %
Doc2VecC	+ stemming			62.73 %
Doc2VecC	+ stemming		+ TF-IDF Weights	**64.23 %**

Table 4.1: F1-scores on cross-validation dataset

As baseline we use the simple $TF\text{-}IDF$, which represents documents as a vector of TF-

IDF scores of words in the description sections, then cosine similarity is used for re-ranking. This baseline performs lowest with an F@10 score of 8.69%. One characteristic can be deduced is that similar documents don't share exact words in the description, and even if they do, the shared words are not enough in frequency and inverse document frequency to distinctly identify similarity between them.

The *Word2Vec* model (Negative Sampling Skip-gram model) was tested using the hyper-parameters in Section 4.1, the score of 54.84% (with striking difference from *TF-IDF*'s) shows that using semantic similarity between words performs better than using exact words, if the words are placed properly in the vector space. Using *Stemmed* input for both training and assertion scores better than using the unstemmed text (+1.38) and help reducing the training time, since stemmed input contains smaller distinct vocabulary than the original's.

Word2VecF model when combined with arbitrary context, in this case the document ID, performs +4.9 better than using Word2Vec alone. Since Word2VecF would aim to predict the documentID instead of the context window, the context can be seen as a global context over the entire document, which should (and does) perform better than using Word2Vec local context.

By using TF-IDF scores in the weighted average, F1 score was boosted by +1.69 at 62.81%. Since all words contributes in building the document vector, TF-IDF gives more share to words which are distinct to the document, hence pulling the document vector more into their direction. Stop words with low TF-IDF score contribute very little in the weighted average, or not at all if word appears in all documents.

Doc2VecC performs the best among single model experiments using description input at 62.73% with stemming only and 64.23% with both stemming and TF-IDF weighted average. As it combines the Word2Vec local context and the Word2VecF global context (using its own document vector while training).

4.3 LDA

LDA model was trained on the corpus after filtering frequent words, that appear in more than 50% of the documents, and those with frequency less than 20. The corpus was populated with bigrams that appear more than 20 times across the dataset, and the model was trained with 1000 topics over the default values used in Gensim python library.

Before conducting the experiment, we analyzed the topic distribution to make sure the topics are interpretable. Table 4.2 shows the top words (translated to English) of some of the latent topics in our dataset. We can see in the example a topic with a very particular theme like *Finance*, where most of the words can be traced back to the field of risk management and financial transactions. Some of the topics has a broad scope like *Technical*, where words ranges from Mechanical Engineering to Technology. One of the topics handle a very specific case as shown in *Regional* topic, where the top words consist of the city *Kiel* which lies in the state of *Schleswig Holstein*. We saw that this topic contributes more in northern-German job postings.

Database	Project leader	Region	Technology	Linux	Investment
SQL	Project	Regional	Technical	System	Risk
ORACLE	Costs	Kiel (city)	Documentation	Unix	Client
Datenbank	Appointments	Schleswig (region)	Engineer	Service	Finance
Maintenance	Management	Holstein (region)	Training	Operation	Banking
Query	Planning	Rest	Further Education	Python	Risk management
Tuning	Subprojects	Resting	Mechanical Engineering	Apache	Transaction
Datawarehouse	Coordination	Schleswig Holstein	Construction	Windows	Trading
Data	responsbility	Sales Region	Design	MySQL	market
Databases	Project management	Regional	Technical	Operation Systems	Finance

Table 4.2: Top words in 6 of our underlying topics in our dataset where last row summarizes the topic in a single word. A document can be represented as a vector of 60% Databases, 30% Operation Systems and 10% Technical.

LDA was tested using stemmed input but it didn't perform anywhere close to the Word2Vec variations that includes global context. In order to reduce the computational complexity of comparing vectors of 1000 dimensions, topic modeling frameworks reduce the vector to sparse one, where dimensions lower than a certain threshold, a probability of 0.01, is set to 0. Two experiments where held using two different similarity measure as shown in Table 4.3, Hellinger distance performs way better on sparse vectors than cosine distances, yet not good enough with a score of 57.69%.

Model				**F1 Score@10**
LDA	+ stemming	+ Cosine Distance		42.55 %
LDA	+ stemming	+ Hellinger Distance		57.69 %
Doc2VecC	+ stemming	+ TF-IDF Weights	+ LDA	**65.85 %**

Table 4.3: LDA results show the striking difference between using different similarity metrics, where hellinger distance is better suited for LDA. Combining LDA and Doc2VecC performed slightly better than Doc2VecC itself.

One idea was to combine the document vector from Doc2VecC and the document vector (vector of probabilities) from LDA to construct a 1500 document vector, hoping to bias the vector towards its global features. However, since the two vectors performs best

with two different similarity measures, combining them wasn't feasible. The approach followed however, was calculating the similarity of the Doc2VecC vectors of 2 given documents, then averaging it out with the similarity of the LDA vectors of the given documents together as illustrated in Figure 4.3.

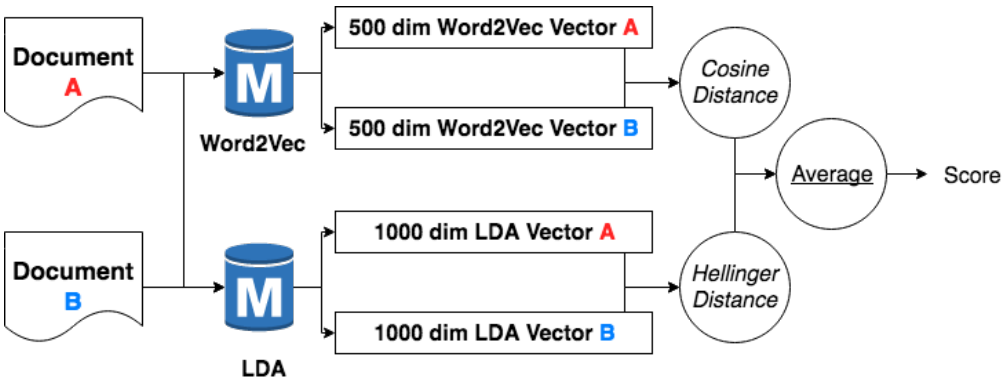


Figure 4.3: To compute the similarity between two documents in LDA and Word2Vec/F/Doc2VecC setup, the models infer the vectors for the two documents, and similarity scores are computed separately using different metrics, then averaged out to compute the final score.

By averaging the LDA vector with Doc2VecC, that already includes local and global contexts, we give more bias to the global aspect of the document when calculating the similarity score. Combining LDA to Doc2VecC showed improvement over using only Doc2VecC by +1.63. Which is the best result achieved from using the plain text in the description of job postings. However, training LDA model takes over 9 hours over a sample of 112,000 jobs, it raises a question - from an every-day usage practical perspective - whether +1.63 increase in accuracy in retrieving similar documents is worth the time and resources spent to train the LDA model in production pipelines.

4.4 Latent Vector Space

Even though using the entire Latent Vector Space is not appropriate for hyperparameters tuning, it is useful to run the models over jobs in unconstrained environment to imitate real life production pipeline, and to check for any contradictory behaviors. 10,000 jobs got sampled from the dataset and embedded in the vector space using different models then K-Nearest Neighbor is used to retrieve nearby jobs within a *threshold* = *0.5*.

As shown in Table 4.4, models' performance scores follow a similar behavior in full latent vector space as the positive and negative dataset but the ratios are different. Stemming

Model	F1 Score@threshold
Word2Vec	2.098 %
Word2Vec + stemming	2.142 %
Word2VecF + stemming + TF-IDF Weights	2.319 %
Doc2VecC + stemming	2.529 %
Doc2VecC + stemming + TF-IDF Weights	**4.237 %**
Doc2VecC + stemming + TF-IDF Weights + LDA	4.231 %

Table 4.4: F1-scores on the entire co-interaction dataset.

doesn't seem to affect the score with negligible +0.04, Word2VecF doesn't perform as good as Doc2VecC. TF-IDF has the biggest impact that almost doubles the F1-score of Doc2VecC without weighted average.

Combining LDA and Doc2VecC in the vector space doesn't seem to add any value (negligible - 0.006%). Following these observations and the results from the Positive/Negative Dataset, along with LDA computational overhead, LDA wasn't picked for the final Online evaluation.

4.5 Job Title

Instead of using the description only, the title can also be used to distinguish similar job postings. An experiment was conducted that started by building a document vector using Word2Vec (stemmed with TF-IDF weights) using the words in the title only, and use the vector for evaluation. The aim of this experiment was to avoid any computational complexity from processing the entire description if the title contains enough information to distinctly differentiate the document in the vector space.

The intuition was that Word2Vec should perform well without any help from global context since titles contains on average 4.8 words, such that word embeddings using the local context should be enough to give promising results. As shown in Table 4.5, Word2Vec using title vector only yielded 58.79%. Compared to the description vector from Doc2VecC (64.23%) in Table 4.1, the F1 score difference is significant enough that there was no point in running the experiment using the full vector space.

Doc2Vec

Since title is short, Doc2Vec training shouldn't take as much training time using titles as using full descriptions. Moreover, Since we are not using the C code, using Gensim library for training Doc2Vec, number of iterations can be asserted and visualized to track down the error rate (or in our test case, the F1 score) while training. As well as using multiple cores for parallel processing, without worrying about dirty writes, since Gensim takes into account locking between threads.

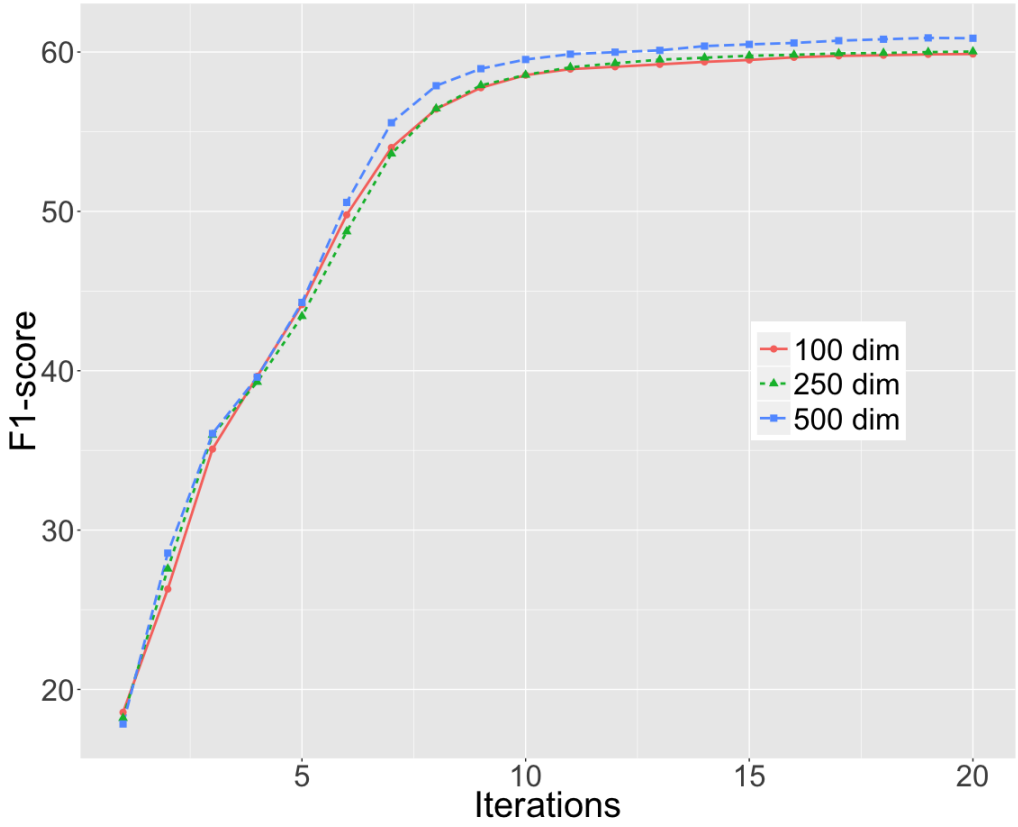
The first Doc2Vec experiment was to assert the number of dimensions, all other variables were fixed, *window size = 10* to include all the title words, default *minimum count = 5* and *sampling rate = 1e-5* and a Distributed Bag of Words model, which has the same architecture as Word2Vec's Skip-gram. The variable number of dimensions was tested with values of 100, 250 and 500.

Figure 4.4 shows the results of conducting the experiment with three DBOW models with different number of dimensions over 20 iterations. Looking at the results, the performance increase from a 100 dimensional model to 500 dimensional one is +1.36%. One could argue that it's a fair price to pay to decrease the training time by using the 100 dimensional model. One interesting remark is that the models seems to converge after 15 iterations, which also reduces the training time once the model is used in production pipelines.

The results of using Doc2Vec on title gives better results than using Word2Vec, but the results shown are plots of F1 scores on training dataset. An experiment has to be conducted to check the effect of Doc2Vec on titles it hasn't seen before. In other words, how good is the model when it infers document vectors rather than updating the neural weights of jobs titles during training.

Another experiment was conducted to test the performance of Doc2Vec models on unseen documents. However, this time we also cross validate the second Doc2Vec architecture, the Distributed Memory model (DM). Our intuition to exclude it in the first experiment was backed by the fact that DM depicts the C-BOW model in Word2Vec which already showed worse results than the Skip-gram (see Section 4.1). However, since titles are considered very short sentences, CBOW approach can behave differently and thus was worth investigating.

Two models were tested using 500 dimensions over 50 iterations, while testing the performance of the inferred vectors of unseen documents every 5 iterations. Figure 4.5 shows the results of the experiment, DBOW performs surprisingly well with 65.39%, almost as

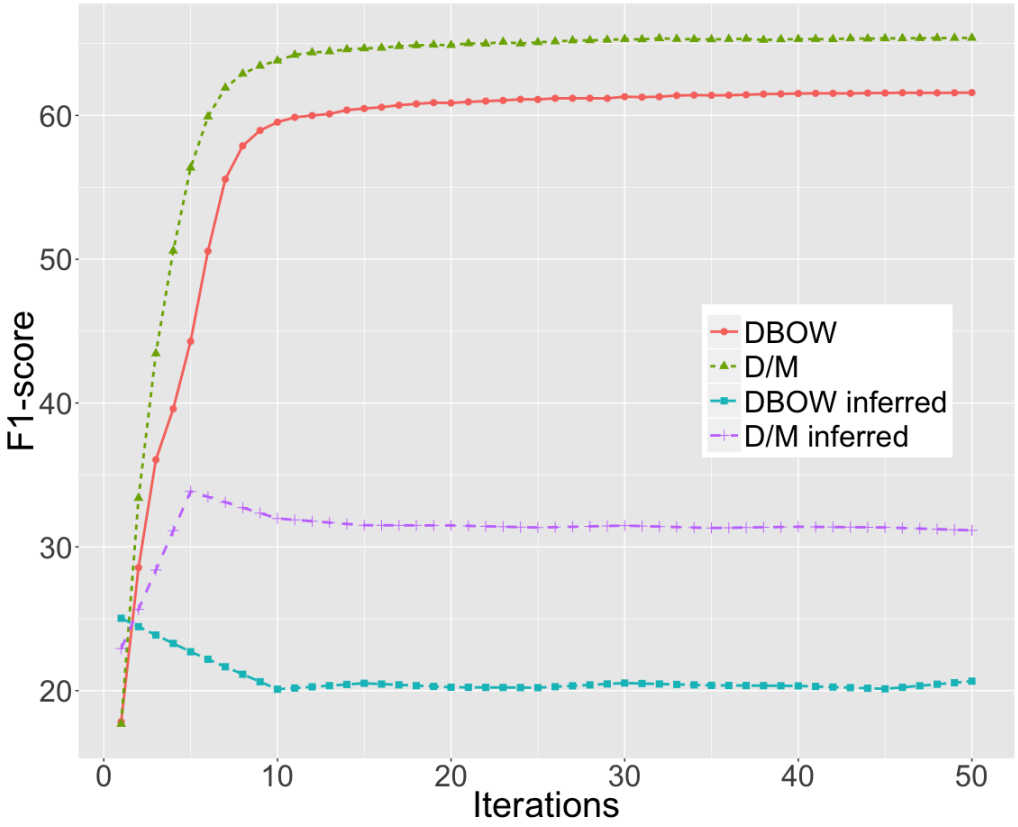


Model	f1score@10 / Iteration = 20
DBOW - 100 dimensions	59.87 %
DBOW - 250 dimensions	60.03 %
DBOW - 500 dimensions	**61.23 %**

Figure 4.4: By fixing all other hyperparameters, experiment was conducted on 3 DBOW models with different number of dimensions to test it's significance. A sample of 2500 jobs where selected as a validation dataset to test the training error across iterations. The table shows the absolute F1 scores at iteration 20.

good as both combining Doc2VecC and LDA together (Section 4.3), where D/M scored 61.59% only as expected from the previous experiment. Unfortunately, Inferred vectors for unseen documents results were drastically lower that most results in all experiments in the thesis at 31.15% and 20.66% for D/M and DBOW inferred vectors respectively.

Having a look at the $inferVector()$ in the implementation, the drop in accuracy can take place for many reasons. For instance, trying to infer a vector for a seen document will not yield the same vector back. It is a random process that tries to approximate the vector by running the training function once using the unseen document to calculate the vector, as it depends on the seed and the order of input tokens fed to the function.



Model (500 Dimensions)	f1score@10 / Iteration = 50
D/M	61.59 %
DBOW	**65.51 %**
D/M - Inferred Vectors	31.15 %
DBOW - Inferred Vectors	20.66 %

Figure 4.5: DBOW models give better performance on short sentenced documents than D/M models. However, accuracy drops to half if the model tried to infer a vector to an unseens document. The table shows the absolute F1 scores at iteration 50.

The drawback of such low accuracy in unseen documents that, Doc2Vec can't be used as a query model where it calculates title vectors on the fly in Online systems and production pipelines. The model would have to be trained overnight every time a bulk of new documents arrives.

4.6 Combining Title and Description Vectors

Since title performed very well, an experiment was conducted to show the effect of using both title and description vectors. Garten et al. [2015] proposed concatenating vectors

to prove that "hybrid representations effectively leverage strengths of individual components". In our scenario, documents can be represented by a single feature vector by concatenating title vector from Doc2Vec and description vectors from Doc2VecC, since both of them share the same distance metric (Cosine Distance). The experiment was rerun using document vectors of 1000 dimensions using both Positive and Negative Dataset and the Full Vector Space, as shown in Table 4.5.

Combining title and description vectors yielded 69.13% F1 score, +3.62 more that its individual components' scores on the Positive/Negative Dataset. On the full latent space, the concatenated vector scored 4.853%, relatively speaking, that's +14% more F1 score that the individual components score in the full latent space. We believe that title contains condensed information regarding the job title with as least noisy words as possible, and the description helps in adding extra information needed to differentiate different jobs that share similar titles.

However, since part of the algorithm uses Doc2Vec (with titles), we are constrained with pre-trained documents, as inferring title vector would drop accuracy as mentioned earlier. On the other hand, Doc2VecC model is train once (can be weekly, monthly) and generate description vectors on the fly. Another approach that involves Online evaluation has to be investigated that uses the information in the title and the Doc2VecC speed in generating vectors for unseen documents, just in case training Doc2Vec everyday is not feasible in production.

Title weighted Description

The experiment followed aimed to use the average of Doc2VecC word vectors of the description weighted by the TF-IDF values except that it also gets weighted by an arbitrary variable if the word is contained in the title. It can be visualized as follows:

$$V(document_{k-words}) = \frac{\sum_{i=1}^{k} \begin{cases} TF\text{-}IDF(w_i) * V(w_i) * P & \text{if } w_i \in title \\ TF\text{-}IDF(w_i) * V(w_i) & \text{otherwise} \end{cases}}{\sum_{i=1}^{k} \begin{cases} TF\text{-}IDF(w_i) * P & \text{if } w_i \in title \\ TF\text{-}IDF(w_i) & \text{otherwise} \end{cases}} \tag{4.1}$$

Equation 4.1 shows how the word vectors gets weighted. When constructing the document vector containing K words ($V(document_{k-words})$), all word vectors $V(w_i)$ are multiplied by the their corresponding TF-IDF values $TF\text{-}IDF(w_i)$ and a constant P if the word

appears in the title, the vectors are summed up and divided over the weights to calculate the weighted average. If no word in the description appear in the title, it behaves as a normal TF-IDF weighted average as explained earlier in Equation 2.2. Otherwise, the word appears in the title, the vector is multiplied by a the constant P that gives more weight for the given word. We already know that the title is good enough to distinguish the document based on the Doc2Vec results earlier, so giving the title words more weight when averaging pull the document vector a bit closer to the title in the vector space.

Model (+ stemming + TF-IDF weights)	**f1score@10**	**f1score@Threshold**
Word2Vec Title	58.79 %	-
Doc2Vec Title	65.51 %	4.258 %
Doc2Vec Title + Doc2VecC Description	69.13 %	**4.853** %
Description Weighted Average in title's words favor	**73.05** %	4.272 %

Table 4.5: Combining title with description resulted in the best performance in the thesis at 73% f1score@10.

By arbitrary picking $P = 5$, the approach yielded 73.05% F1 score on the Positive and negative Dataset, considered the best result we have by a + 3.92 better than the second best as shown in Table 4.5. It shows that by choosing a proper weighting function, we can achieve better results than changing the entire model. However, it scored 4.272% in the Full Vector Space, going second best after concatenating the Title Doc2Vec and the Description Doc2VecC.

As mentioned earlier, The Full vector space models the user interactions more than the similarity of documents themselves because of how the dataset were sampled out and created. One reason for the drop in the F1 score while using the weighted average approach when compared to concatenation technique is that number of dimensions was doubled (500 vs 1000 dimensions respectively), and using the same number of trees in the random hyperplane generation of ANNOY may yield different results in an unfair comparison.

4.7 Synopsis

To sum up, we conducted multiple experiments in this chapter to decide on which different preprocessing steps, hyperparameters values and models should be used to increase the quality of recommended similar job postings for users. Results showed that F1 values increase significantly by lowercasing and stemming the input text, which helps in reducing the vocabulary size and increasing the training co-occurrences.

After cross validating the model architecture (CBOW vs Skip-gram) and different window sizes (1,5,10), the Skip-gram model with window of size 10 performed the best on our ground truth. We believe Skip-gram showed better results because of its sensitivity to infrequent and rare words while bigger window size helps integrating more words while training, resulting in a global context learning behavior. Performance of different semantic models were compared to decide on the best performing model on our job postings dataset. Word2VecF model that incorporates global features while learning performs better than models that only uses local features only like Word2Vec. Where Doc2VecC that uses both global and local features performed the best in the model comparison. As document vector is constructed by average of word vectors in the document, TF-IDF is used to convert document vector to weighted average, where it consistently performed better than using normal average. Since documents contain certain words that can uniquely identify them, more weights should be given to these unique words when constructing the document vector.

LDA was used in an attempt to see the effect of topic modeling algorithms on document similarity, it didn't perform as good as Doc2VecC. However, when both models were combined in an attempt to bias the global features, they performed slightly better than Doc2VecC.

Job titles were integrated to investigate the effect of using other document features beside the description. After cross validating the model and number of dimensions, Doc2Vec Distributed Memory model with 500 dimensions were used. The model, using only the job titles, scored better than any model that integrated job posting descriptions. Combining both Doc2Vec on title and Doc2VecC description yielded even better results than only using Doc2Vec. We believe using the title is good enough to distinguish most of the jobs apart, but description contains necessary contextual information than be should integrated for more accuracy.

Overall, our strategy was to use a model that generates document vectors with the least training and evaluation time. A model that uses a weighted average of the word vectors favoring those that refer to words in the title achieved the best performance and was therefore chosen to be tested online in XING's recommender system. The details of the online evaluation are discussed in the subsequent chapter.

5 Online Evaluation

The final phase of the thesis is the A/B test, where the best performing model during the experiments along with the experimented hyperparameters are tested to evaluate their performances against the current implementation of XING Elastic Search Reranking.

5.1 Experimental Setup

The two evaluation datasets explained in Section 3.3 are considered Offline, since data were collected from an earlier time span and filtered from outliers. An Online evaluation however is testing the model in real time using active users on the platform, to assert the performance in every day situation.

A controlled experiment was conducted that splits most active users into two groups, where the first group gets job posting recommendations using the currently existing architecture known as the Control Group, and the second group gets job posting recommendations that are vectorized and re-ranked using the model currently under trial, the group is known as the experimental group. In Industry, the test is known as **A/B Test** which maps to Control/Experimental groups respectively.

An A/A test was conducted [Kohavi et al. 2009], that splits the sample of active user groups to two groups that get recommendations from the same recommender system, in order to validate the absence of any bias that may occur from choosing a poor split. Table 5.1 shows the result of the A/A test, the absolute Click-through Rate (CTR) is masked due to XING policies, by normalizing the real CTR in Group A_1 to *50%* and changing the absolute number of clicks and Group A_2's CTR accordingly to preserve ratios and obscure XING's private data.

The table shows the results of the split over 30 days, where the same users was separated to two groups based on the users' IDs. On average there are 1.4 million users who viewed 10.9 million job posting on average, and *(masked out)* 5.46 million job postings got their attention. The **Click-Through Rate (CTR)** is the core of Online evaluations, as it measures success rate of a certain algorithm or major change in the logic behind delivering services to users. It is calculated as shown in Equation 5.1, the more items that the user interacted with, the higher the CTR and the more successful is the algorithm (or less successful if less interactions with a certain group of items is intentional).

	Group A_1	**Group** A_2
Number of users	1,399,523	1,403,822
Shown	10,918,720	10,952,460
Masked Clicks	5,459,360	5,476,230
Masked CTR	50.00%	50.035%

Table 5.1: A/A output for the split function over sampled users over 30 days, with masked Clicks and CTR to comply with XING policies, resulting in negligible difference in the split.

$$CTR = \frac{\#ofclickeditems}{\#ofshownitems} * 100 \tag{5.1}$$

Since the difference in the CTR between the two groups is almost negligible (*0.02%*). The split hashing function can be used to conduct the A/B Test, and the test outcome should be credited to the different strategy applied on the Experimental Group.

The experiment was conducted by having XING employees only as the experimental group, then ramping up the ratio of all users on the platform from 5% to 50% over 4 intervals with a couple of hours in between. This way we can track missing recommendations or problems in the workflow that may affect large amount of users which can be caught and handled in time with minimal losses [Kohavi et al. 2009].

Our hypothesis is that group B, interacting with job postings reranked by our top performing model, should have higher CTR, since users will be shown job postings which are more similar to those they interacted with in the past.

The best performing model, title weighted description, was used in the A/B Test to assert the model's performance with user interactions in real time, and to validate the hypothesis that job postings similar to the ones interacted by the users will be more appealing to them. The preprocessing stage had to be rewritten in Scala to fit in XING's technology stack. The Hive, Spark, Oozie and lower level workflows are explained in details in Section A.5.

On a higher level, Figure 5.1 shows the concept of the A/B Test workflow. There are 4 main steps to generate the recommendations in the A/B test environment. It starts by a user bookmarking and positively interacting with multiple jobs on the platform. The next time the platform generates recommendations for this user, a query is generated based on the aggregated features and metadata of previously bookmarked and replied job postings. The query is sent to ElasticSearch which uses the keywords and metadata

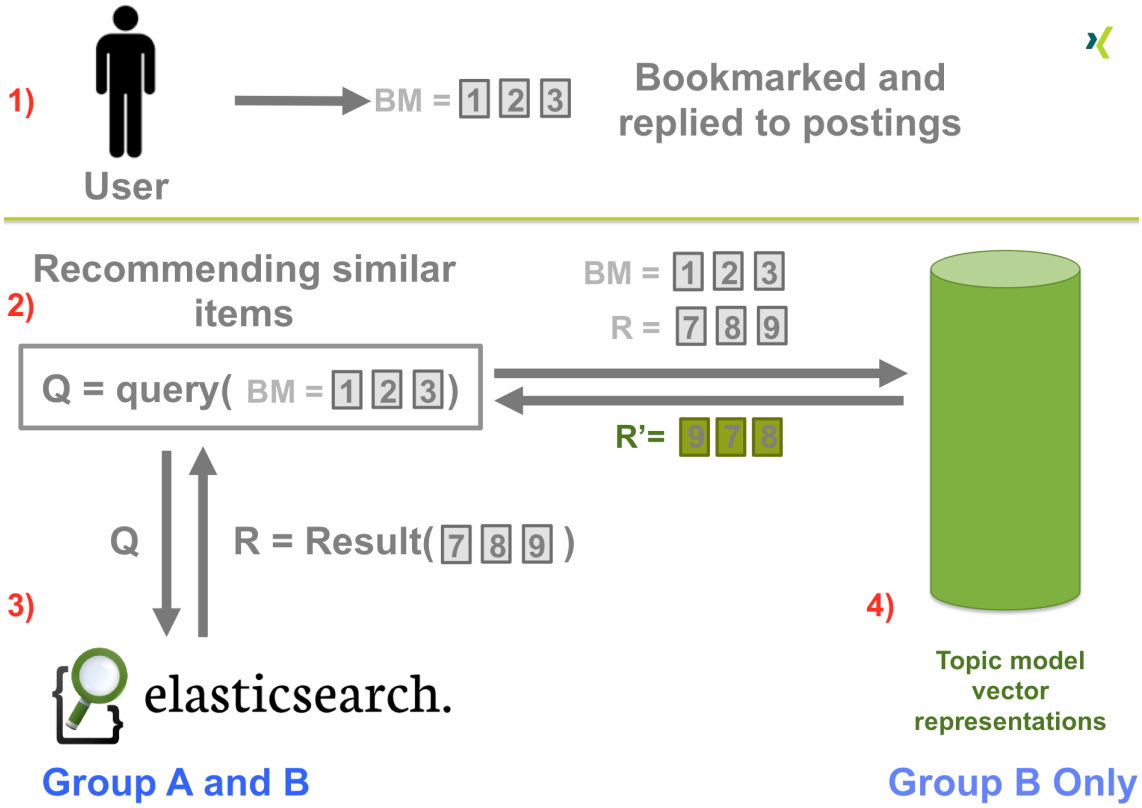


Figure 5.1: The workflow runs as follows: i) The user bookmarks and replies to job postings are denoted as set BM. ii) The recommender creates a query Q using the metadata of BM. iii) For both Group A and B, Q is issued to ElasticSearch to retrieve documents and rank them using keywords and BM metadata, resulting in a set of job recommendations R. iv) An additional step for group B is passing the results R and bookmarked/replied jobs BM to Topic model vector representation module to re-rank the documents based on similarity between BM and R, and returns the reranked list R' [XING internals].

like disciplines to retrieve and rank job postings from XING jobs inventory. Until now, this is the current implementation of More-Like-This recommender at XING and it applies to all users.

If the user ID belongs to group B in the A/B Test, an extra module receives the list of retrieved recommendations while from ElasticSearch and ignores its ranking. Then, along with the positively interacted job postings from the user, it calculate the cosine similarity between the positively interacted vectors and the retrieved job postings vectors in the latent space to re-rank the list. Hence, we explicitly compare the re-ranking performance between ElasticSearch metadata and keyword matching algorithm in group A versus Semantic and Topic model embeddings in group B.

5.2 Results

Before conducting the Online evaluation workflow, another test had to be conducted first to check whether the A/B test will give different results than the current architecture. In other words, is it worth it to run the A/B test in the first place?

The model got implemented and deployed on a REST endpoints. Which was used - along with the original endpoint - to rerank the same recommendations for 2000 of most active users on the platform. As shown in Table 5.2, the intersection of recommendations between the two endpoints does not pass 36% for all K ranks in the recommendation lists. Which shows that the changes implemented would have significant results.

Top K	Intersection		Distance From User (KM)	
	mean	**stdev**	**Original Endpoint**	**Endpoint B**
4	30.1%	32.16%	287.4	179.3
10	35.5%	27.89%	293.5	188.4
20	35.4%	25.69%	325.4	195.1
50	34.1%	21.28%	336.1	191.9

Table 5.2: A/B test pre-analysis of recommendations

Another interesting remark from the analysis is the average distance of the jobs to the User. In other words, how far are the job recommendations from the user's location/city in Kilometers (KM). The model reranks recommendations with an average distance of 30% closer to the users than the current approach in the top 4, up to 60% in the top 50. One hypothesis is that jobs in the same city are closer in the vector space to each other which are influenced by city vectors in the description.

The A/B test was conducted for 10 days using the user group split discussed in Section 5.1, the target audience for the test was a sample of users who bookmarked, interacted or rated a job posting in the last 3 months. Due to the fact that noninteracting users will not receive any recommendations from the MLT recommender, which makes the two groups get the same recommendations from other sub-recommenders.

Table 5.3 shows the results from the A/B Test, the absolute values of Clicks and CTR are masked out by normalizing Group A's real CTR to 50% and changing the Clicks and Group B's CTR accordingly to preserve the ratios without showing XING confidential numbers. The experimental group B has a very noticeable difference of +8.00% more clicks per received recommendations using the title weighted description model in a short span of 20 days.

	Group A	**Group** B
No of users	77,151	76,853
Shown	1,022,788	1,003,527
Masked Clicks	511,394	541,904
Masked CTR	50.00%	54.00%

Table 5.3: Sampled users in Group B have +8.00% higher CTR using those in control group A. The CTR and absolute clicks are masked to comply with XING policies.

5.3 Synopsis

In this chapter, an A/A test was conducted to split the users into two groups and make sure the two groups performs similarity before applying any major changes. Then the A/B test was conducted over the span of 20 days to assert the impact of the document representation model. The results of the offline evaluation could thus be confirmed in the online A/B test. Using the weighted average of word vectors while putting an emphasis on those word vectors that appear in the title of the job posting outperformed the current setup that XING was using, i.e. a ranking that is primarily based on Elastic Search's TF-IDF vectors.

6 Conclusion

To sum up the work done in this thesis, we constructed the dataset based on the positive user interactions with the platform due to the lack of a dataset of similar job postings. Our hypothesis that if 2 or more users showed interest in 2 job postings, these postings are similar. We used the constructed dataset to evaluate the performance of different document representation models and preprocessing steps. Experiments conducted during the thesis displayed better performance when the text is preprocessed via Lowercase conversion and stemming, reducing the number of vocabulary the models has to deal with, and increasing the number of word co-occurrences for training.

Different document representation techniques have been experimented with like average of word vectors from different word representation models like of Word2Vec, Word2VecF with arbitary context and Doc2VecC. The experimental process took an iterative approach in finding out the best hyperparameters for the semantic models. Experiments with topic modeling algorithms was also conducted using LDA on its own and in combination with Doc2VecC vector representation which yielded slightly better performance than only using Doc2VecC, which were not used for the Online test due to the computational complexity of LDA training in production.

We went to explore different features of the job postings that is not associated with the job description. For instance, Doc2Vec using only job postings' titles yielded better results that using any document description representation, and combining both title vector from Doc2Vec and description vector from Doc2VecC yielded the best results on our dataset. However, in an effort to eliminate Doc2Vec due to its limitations in inferring document vectors, we showed how manipulating the average weight of word vectors to favor features like job title can improve the accuracy of retrieved jobs. Since words in the document should not be equally responsible to construct the document vector, researching a better weighting function should be as important as finding the best vector representation model.

Finally, an A/B test was conducted to assert our hypotheses during the thesis. The best performing document representation along with the experimented hyperparameters were implemented in XING pipeline. In order to test it in real time over an experimental and a control user groups, where the A/B test succeeded with about +8.00% increase in Click-Through Rate, concluding that users are indeed getting more similar jobs to those they annotated earlier.

7 Future Work

This section lists down the main research questions raised by the thesis, which couldn't be tackled due to the time constraint. Future work proposes new experiments that should be conducted to investigate the impact of different preprocessing ideas and semantic model modifications on the accuracy of results.

As stated in Section 2.3.1, dense vector representation extracts semantic meanings based on the co-occurrence of words in the dataset. As more word co-occurrences within the same context throughout the corpus yield better word representations in the vector space. However, in the German language, there exist many compound words, for example *Diplomsozialpädagogik* which can be split down to *Diplom*, *Sozial* and *Pädagogik*. The model would include the compound word as a stand alone word along its subcomponents in the vocabulary.

Compound words are less frequent than their single compounds, implying weaker representation of compound words since they are not as frequent as their subcomponents. The effect of splitting compound words should be investigated which reduces vocabulary size and training time, and increase the number of co-occurrences for even better representation for the subcomponents. For this, compound splitting methods can be used like SECOS [Riedl and Biemann 2016], [Koehn and Knight 2003] or [Ziering and van der Plas 2016].

Same problem can be seen with using common prefix and suffixes in the job description, e.g *Team- und Kommunikationsfähigkeit* is read as *Teamfähigkeit* and *Kommunicationsfähigkeit*, same with *ziel- und ergebnisorientierte* and *Analysetools und -verfahren*. The current model can't reconstruct the words back from the common prefixes and suffixes, and treat the incomplete word in the text as a new vocabulary. Recovering the words to its original form will also decrease vocabulary size and increase the number of training co-occurrences.

Regarding weight functions discussed in Section 2.4.1, our intuition that TF-IDF can handle exhibiting and inhibiting words based on their importance to the document, but other methods with steeper and more rewarding/punishing weight functions should be experimented with. Average of word vectors can be limited only by named entities using Named Entity Recognition, which excludes noisy and stop words from calculating the document vector.

One of the popular applications in Word2Vec is the vector arithmetics operations on analogies, such that $King - Man + Woman = Queen$. In other words, retrieve all words

in the vector space that is similar to $King$ and $Woman$ but not similar to Man, and the answer was words closest to $Queen$. The applications of vector arithmetics on job posting dataset, if a better model is found that preserve relationships between major document keywords, can allow users to tailor there search results coming from the vector space, for example, retrieve all job postings that are similar to $Marketing$ and $Internship$ and not similar to $Computer$ to retrieve student jobs in Marketing areas that is not involved with technology.

Doc2VecC showed interesting results in both the original paper and the experiments on XING dataset (see Section 4.2). One can exploit the document vector used for training in the model by selecting top TF-IDF words to represent the document instead of just random word sample. Metadata can be used as document vector like title words and skills in the job postings.

Finally, since we are restricted by XING architecture, ElasticSearch is used for retrieval of documents and Reranking step is done using Dense Vector representation. ANNOY can be used for both retrieval and reranking, by placing all the documents inventory in the vector space, we are no longer tied to ElasticSearch retrieval rule that uses strict matching. ANNOY will be able to retrieve documents based on their text description and not their metadata, that can query relevant documents which ElasticSearch will not be able to retrieve.

A Appendix

A.1 XING Metadata

Users at XING can fill in what kind of skills they excel at, and what kind of job opportunities they are looking for as shown in Figure A.1. This helps the job recruiters to find the candidates for a given job, and help XING improve the quality of the recommendations delivered for the users.

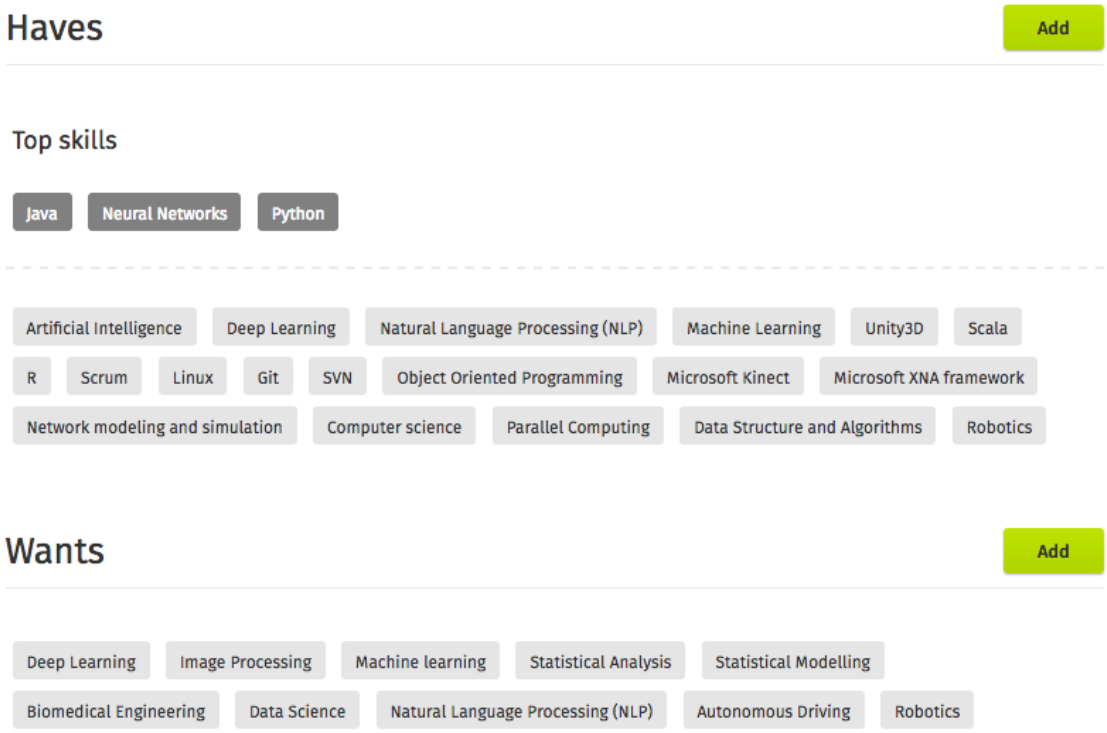


Figure A.1: Users at XING can supply the skill set they possess and what kind of job opportunities they are looking for.

A.2 Similar Postings Sample

Table A.1 shows a sample of similar postings used as a dataset during experiments. The tuples were collected from user interactions on the platform, where two job postings are considered similar if 3 or more users co-interacted with the two jobs positively, via bookmarks or reply intentions.

Job Posting A	Job Posting B
Senior Consultant (m/w) Management Beratung	Senior Consultant (m/w) Operations
Regionalverkaufsleiter (m/w)	Bereichsleiter (m/w) / Managementnachwuchs
Bereichsleiter (m/w) / Managementnachwuchs	Führungsnachwuchskraft (m/w) Vertrieb / Junior Sales Manager (m/w)
Junior Key Account Manager (m/w) im Lebensmitteleinzelhandel	Führungsnachwuchskraft (m/w) Vertrieb / Junior Sales Manager (m/w)
Senior Consultant (m/w) Management Beratung	Senior Associate (m/w)
Ingenieure (m/w) Maschinenbau und Elektrotechnik	Ingenieure, Bachelor, Master (m/w)
Entwicklungsingenieur (m/w)	Projektingenieur (m/w)
Senior Associate (m/w) Technology Strategy	Senior Associate (m/w)
Trainee MINI Produktmanagement (m/w)	Trainee Produktmanagement (m/w)
IT LEITER (M/W)	BEREICHSLEITUNG IT / CIO (M/W)
VICE PRESIDENT OPERATIONS - WELTWEIT - Strategische Verantwortung: Prozesse	GESCHÄFTSFÜHRER, MANAGING DIRECTOR EMEA - Erfolgreicher Mittelständler
Berufseinsteiger / Hochschulabsolvent - Human Resources (m/w)	Junior HR Manager (m/w)
GESCHÄFTSFÜHRER, MANAGING DIRECTOR EMEA - Erfolgreicher Mittelständler Autom	Geschäftsführer Vertrieb/Marketing m/w
BEREICHSLEITUNG IT / CIO (M/W)	Leiter IT (CIO) m/w
M&A Berater (m/w)	Senior Associate (m/w)
Strategic Business Development Manager (m/w)	Business Development Manager (m/w)
Director (m/w)	GESCHÄFTSFÜHRER, MANAGING DIRECTOR EMEA - Erfolgreicher Mittelständler
Trainer/in und Berater/in für Führungskräfte-Entwicklung	Trainer/in und Berater/in Persönlichkeits-Entwicklung
LKW- Disponent/in	disponent nahverkehr (m/w)
Consultant (w/m) Tax	Consultant (w/m) Wirtschaftsprüfung / Audit
Management-Trainee-Programm	Bereichsleiter (m/w) / Managementnachwuchs
Selbstständige Berater (m/w) für internationales Coaching-Unternehmen	Quereinsteiger (m/w) als selbstständige Unternehmer für Führungskräfte-

Table A.1: Example of similar job postings in the dataset

A.3 Job Posting Similarity Score

An experiment was conducted during the early stages in the thesis to discover features that can help deducing a similarity scores between job postings by exploiting TF-IDF approach in calculating the score. for example, Job_1 and Job_2 has 23 interested users in common. but one of those users is interested in 50 other jobs, in the same logic as IDF, this user is not interested in this specific job as someone who have 5 job interests only.

So the suggested similarity score was use absolute count of common users N as TF value,

multiplied by average of *log*arithmically scaled interaction per user u by the maximum number of user interactions as IDF:

$$similarity(Job_1, Job_2, N) = N * \sum_{u=1}^{N} \frac{log(\frac{MAX(INTERACTION)+1}{INTERACTION(u)})}{N} \tag{A.1}$$

Note that outliers should be removed before getting the maximum interactions. Also, The *log* nominator has to be 1 more than the maximum in include most diverse and active users (even if they contribute very little).

The results shown in A.2 shows the top scores, which - according to our assumption - should be the most similar in our dataset in such a way that a pair of jobs should be more similar to each other than another with lower score. The same logic implies that the lowest rows of the table should contain the least similar job postings with the lowest scores and should be filtered out if possible.

Job Posting A	Job Posting B	Score
Junior Berater(in)	Junior-Berater (m/w)	19.325092609365527
Ingenieure (m/w) Maschinenbau und Elektrotech	Ingenieure, Bachelor, Master (m/w)	17.183404979908726
Ingenieure (m/w) für eine nachhaltige Karrier	Qualitätsingenieur (m/w)	16.097931530469232
Global E-Commerce Manager (f/m) "Multi-Channe	Global E-Commerce Manager (f/m) "Marketplaces"	14.305941834880505
Objektleitung (m/w)	Technikumsfachkraft (m/w) Technologieentwicklungszentru	14.040325531164266
Business Analyst	Senior Business Analyst	13.428748543542065
Praktikum: Praktikant M&A / Corporate Finance	Praktikum Mergers and Acquisitions (M&A) / Corporate Fi	13.14698925238014
General Manager China (m/w)	General Manager China	13.132628067596858
Praktikum: Praktikant (m/w) im Bereich Person	Praktikant (m/w) im Bereich Human Resources - Recruitin	12.806322408073845
Internship: Administrative Operations (Intern	Internship: Project Management Assistant (Intern/Workin	12.64634977031052
Junior-Consultant / Consultant (m/w) Industr	Junior-Consultant / Consultant (m/w) Retail Frankfurt	12.245597040699085
Trainee SAP Consultant (m/w)	Einstiegsprogramm mind access (m/w)	12.217703151279492
Praktikum: Praktikant/in Entwicklung Fahrzeug	Praktikum: Praktikant/in Innovationsmanagement	12.194691929235548
Praktikum: Praktikant/in Digital Marketing	Praktikum: Praktikant/in Product Marketing	11.83270617345294
Head of Finance / operativer CFO (m/w) für mi	Chief Financial Officer (m/w)	11.599521713811814
Recruiter (m/w)	Sourcer / Recruiter (m/w)	11.559789351196196
Studentenjob: Trainee-Programme bei AB InBev	Studentenjob: Trainee-Programme in Bremen - General Man	11.407704832016398
Krankenpflegehelfer (m/w)	Technikumsfachkraft (m/w) Technologieentwicklungszentru	11.341945183721643
Praktikum / Studentenjob im Business Developm	Internship: Summer Analyst | London (m/f)	11.241406060754091
VP FINANCE EMEA & ASIA PACIFIC	GLOBAL HEAD OF FINANCE (M/F)	11.181253081569333
Mediengestalter/Reinzeichner	Grafikdesigner/Mediengestalter/Layouter/Reinzeichner (m	11.142771942202257
Projektmanager (m/w) Kommunikation und Market	Marketing/PR & Eventmanager - m/w	11.083593708885372
Internship: Administrative Operations (Intern	Internship: Project Management Assistant (Intern/Workin	11.06042365064935
Senior Marketing Manager (m/f) -	Kindle Content Redakteur/in	11.028797234269074
Praktikum Personalentwicklung / Academy	Praktikum HR People & Talent Management	10.965929440982546
Gesundheitsberater (m/w) in München	Gesundheitsberater (m/w), München, attraktive Bezahlung	10.895442224479737
Investment Banking Analyst - Germany	Investment Banking Associate - Germany	10.807258043490705
Praktikum: Heimarbeit Kugelschreiber montiere	Praktikum: Heimarbeit Kugelschreiber montieren	10.805448127604095
Graphic Designer / Art Director for Diverse a	Grafik Designer (Art Director)	10.762749995976673
Praktikum: Heimarbeit Kugelschreiber montiere	Studentenjob: Heimarbeit mit Kugelschreibern	10.736257755101933

Table A.2: Top scores of the similarity scoring attempt

Unfortunately, Table A.3 shows what should have been the least similar job posting pairs in the dataset. Visual inspection shows very similar job postings like *Software Test Engineer* and *Quality Assurance Manager*, *Linux Administrator* and *System Administrator with focus in Linux* and *Director Corporate Development* and Sales Manager. Job descriptions has been inspected to avoid any title bias, but job posting pairs where also very similar on

the description level.

Experiments were conducted using different similarity scores that punishes interactions on jobs only, others that punish both interactions on jobs and users, along with smoother and harsher log scales, but results were the all the same, the similarity scoring doesn't reflect the degree of similarity between job postings. Hence, the idea of having similarity score was refuted, and the dataset would contain only similar and non similar job postings.

Job Posting A	Job Posting B	Score
Verkaufsleiter (w/m)	National Key Account Manager (m/w)	0.008217335371184667
Chief Operations Officer/ Co-Geschäftsführer (m/w)	Geschäftsführer: Management In bei kleinem Champion	0.011485057489497127
Verkaufsleiter (w/m)	Director Corporate Development (m/w)	0.011801624679543049
Verkaufsleiter (w/m)	Sales Manager (m/w)	0.01186391250310543
Kommunikations-/ Marketingmanager (m/W)	PR Berater / PR Consultant (m/w) Bereich Fashion	0.014470677361283046
Apothekenaußendienstmitarbeiter / Apothekenreferent	OTC Vertriebsspezialisten im Leuchtenfachhandel (m/w)	0.01514486597033136
Head of New Sales (m/w)	Senior Consultant or Manager - Management Consulting	0.017456029024504174
Head of New Sales (m/w)	Field Marketing Professional (B2B Channel)	0.01960293681314222
Operativer Controller (m/w)	Mandat Business Controller - Karrierechance für High-Potential	0.022097250675885163
Field Marketing Professional (B2B Channel)	Senior Consultant or Manager - Management Consulting - Telco	0.023762613704010106
Vertriebsleitung (m/w) für die Vertriebsgebiete Ham	Prozessmanager (m/w) Back-Office Vertrieb	0.0239028494917414
Verkaufsleiter (w/m)	Area Sales Manager (m/w) / Gebietsleiter/in im Außendienst	0.024554500428032953
Vertriebsleiter global/internationale Kunden	Field Marketing Professional (B2B Channel)	0.02500730314817397
Software Test Engineer / Testautomatisier (m/w)	Quality Assurance Manager / Testmanager (m/w)	0.026472737978250734
Software Test Engineer / Testautomatisier (m/w)	Quality Assurance Manager / Testmanager (m/w)	0.02723568894419922
Trade Marketing Manager (m/w)	Senior Manager Business Development (m/w)	0.029883873685283294
Quality Assurance Engineer (m/w)	Quality Assurance Manager / Testmanager (m/w)	0.029890790220842376
Aerodynamik Ingenieur für Turbomaschinen oder exter	r FEM/NVH/Akustik (m/w) für den Standort Stuttgart	0.029890790220842376
Quality Assurance Engineer (m/w)	Quality Assurance Manager / Testmanager (m/w)	0.031115509737746346
Management-Nachwuchs (m/w)	Brand-Portfolio-Manager / Marketing-Manager (m/w) Cheese	0.03131193483637759
Verkaufsleiter (w/m)	Key Account Manager LEH (m/w)	0.031372259124059075
(Junior/Senior) Mobile Developer Android (m/w)	Werkstudent/Praktikant (m/w) im Bereich App Entwicklung	0.03195629030450236
Director Corporate Development (m/w)	Projektleiter Strategie & Unternehmensentwicklung (m/w)	0.03230117310892549
Key Account Manager (m/w) - Enterprise Customers	Junior Sales Manager (m/w) Handel	0.032485301714278395
Director Corporate Development (m/w)	Regionalverkaufsleiter (m/w)	0.034751070682343045
Linux Administrator (m/w)	System Administrator (w/m) Schwerpunkt Linux	0.03582204980294593

Table A.3: Lowest scores of the similarity scoring attempt

A.4 Job Description Preprocessing

Figure A.2 shows the before and after effects of applying the document preprocessing stage on text input, important HTML tags are used as sentence tokenizer like $< br >$. HTML tags are then removed entirely from the text, numbers get normalized to numb keyword to preserve the context. Websites and Emails are removed and Umlauts gets normalized to make it easier for decoding step that removes all special characters Finally, the text is lowercased and stemmed for training or generating a document vector.

```
<!—XHS:1:JOBS:6—>Ihr Aufgabenbereich<br><br>
Definition des Produkt—Portfolios f\xc3\xbcr Produkte von Drittanbietern (TPP) in \xc3\x9cbereinstimmung mit der Produktstrategie unserer Lighting—Marken<br>
Definition des Produkt—Portfolios f\xc3\xbcr Handelsware und OEM—Vertriebskan\xc3\xa4le, um die entsprechenden Kundenanforderungen zu antizipieren und zu erf\xc3\xbcllen<br>
Management des Produktlebenszyklus (OEM und Handelswaren) inklusive Road Map, Management von Neuprodukten, Produkteinf\xc3\xbchrung und Pricing<br>
Enge Zusammenarbeit mit den Produktmana—gern unserer Gesch\xc3\xa4ftseinheit Zumtobel &amp;amp; Thorn bez\xc3\xbcglich des Produktlebenszyklus im Bereich Drittanbieter<br>
Aktive Mitarbeit an Make—or—Buy—Entscheidungen<br>
Produktpr\xc3\xa4sentationen und —schulungen f\xc3\xbcr OEM— und Lighting—Kunden<br>
\xc3\x9cberwachung und Berichterstattung der produktbezogenen Umsatzentwicklung<br>
Enge Zusammenarbeit mit dem Einkauf, dem Qualit\xc3\xa4ts—und Prozessmanagement sowie verschiedenen Schnittstellen innerhalb der Zumtobel Gruppe<br><br>
Ihr Profil<br><br>
Abgeschlossenes Studium in der Betriebswirtschaftslehre oder im Marketing<br>
Mindestens 2 Jahre einschl\xc3\xa4gige Berufspraxis sowie Erfahrung auf internationalem Parkett<br>
Bereitschaft zu Dienstreisen auf weltweiter Ebene sowie zur Mitarbeit in einem kleinen, ambitionierten Team<br>
Routiniert in der Steuerung von Schnittstellen zu bzw. zwischen einer Reihe von Abteilungen und internen Prozessen<br><br>
Sie sind an der Mitarbeit in einem internationalen Konzern mit spannenden Aufgabengebieten inte—ressiert?<br><br>
Auf Ihre Online—Bewerbung unter www.zumtobelgroup.com/de/karriere.htm freut sich<br><br>
Nadine Grasl<br>
Human Resources <br>
Tel. +43 664 808 92 2111<br>
```

```
ihr aufgabenbereich
definition des produkt portfolios fuer produkt von drittanbiet tpp in uebereinstimm mit der produktstrategi uns lighting mark
definition des produkt portfolios fuer handelswar und oem vertriebskanael um die entsprech kundenanforder zu antizipi und zu erfuell
management des produktlebenszyklus oem und handelswar inklusiv road map management von neuprodukt produkteinfuehr und pricing
eng zusammenarbeit mit den produktmana gern uns geschaeftsein zumtobel amp thorn bezueg des produktlebenszyklus im bereich drittanbiet
aktiv mitarbeit an mak or buy entscheid
produktpraesentation und schulung fuer oem und lighting kund
ueberwach und berichterstatt der produktbezog umsatzentwickl
eng zusammenarbeit mit dem einkauf dem qualitaet und prozessmanagement sowi verschied schnittstell innerhalb der zumtobel grupp
ihr profil
abgeschloss studium in der betriebswirtschaftslehr oder im marketing
mindest numb jahr einschlaeg berufspraxis sowi erfahr auf international parkett
bereitschaft zu dienstreis auf weltweit eben sowi zur mitarbeit in ein klein ambitioniert team
routiniert in der steuer von schnittstell zu bzw zwisch ein reih von abteil und intern prozess
sie sind an der mitarbeit in ein international konz mit spannend aufgabengebiet int ressiert
auf ihr onlin bewerb unt freut sich
nadin grasl
human resourc
tel
numb numb numb numb numb
```

Figure A.2: HTML tags are removed, numbers normalized to numbs, lowercase filter and stemmers are applied, umlauts normalized and websites removed.

A.5 A/B Test Workflow

In order to test the impact of using Dense vector representations using online evaluation, the A/B test has to be implemented using the technology stack at XING. The stack used in the A/B test consists of the following:

- Hive[1] is a data warehouse software that uses SQL like syntax to manage datasets in distributed environments like Hadoop Distributed File System (HDFS).

[1] https://hive.apache.org/

- Spark[1] is a cluster computing framework that allows in memory data processing that's best utilized for data streaming. It uses Resilient Distributed Dataset (RDD) as primary data abstraction, it is lazy evaluated such that data is only transformed and available when needed, and it allows parallel operation on data.

- Cassandra[2] is a highly available distributed database with linear scale performance, making it ideal for read operations in run-time.

- Oozie[3] is a workflow managing system used to run and schedule Apache Hadoop jobs.

The Workflow has to be conducted every night using Oozie to generate vectors for available jobs as follows:

- **Jobs retrieval:** Hive script retrieves all available jobs from users, and all interacted jobs in the past 3 months by the users.

- **Jobs Preprocessing and Vector Inference:** Spark retrieves the jobs, the word vectors and the titles to combine them and infer the document vectors.

- **Jobs' vectors storage:** Vectors are imported to Cassandra to be used by the recommender to calculate similarity scores.

Dense Vector representation models return a text file in a form of a map of word to vector representations. Early stages failed to load the file to every driver in Spark (4 GB file), it had to be ported from HDFS to a tabular format that can be read by Hive/Spark. Which changed the logic of how Spark reads the documents to an RDD of pairs (*PostingID*, *Word*) to facilitate joins over the word vectors table.

```
1  val title:          RDD[(PostingID, Array[Word])] //Retrieves PostingIDs and cleans/lowercase/stems the titles
2  val wordVectors:    RDD[(Word, DenseVector[Double])]
3  val docIDWordPair:  RDD[(PostingID, Word)]
4
5  val docVecWithIDF: RDD[(PostingID, Array[Double])] = docIDWordPair
6    // Get title words with left join to handle empty titles
7    .leftOuterJoin(title)
8    // Identify words that appear in title
9    .map { case (docID, (word, titleWords)) =>
10     (word, (docID, titleWords.fold(false)(_.contains(word)))) }
11   // Join with Word's IDF value and Vector
12   .join(wordsIDF)
13   .join(wordsVectors)
14   // Add the title weight to the TF-IDF value with respect to titleWords
15   .map { case ((_, (((documentId, isInTitle), idfValue), vector))) =>
16     val weightedVector      = vector * idfValue
17     val weightedVectorTitle = vector * idfValue * titleBoost
```

[1]https://spark.apache.org/
[2]http://cassandra.apache.org/
[3]http://oozie.apache.org/

```
18      val weightedTitleBoost     = idfValue * titleBoost
19      (documentId, (if (isInTitle) weightedVectorTitle else weightedVector, if (isInTitle) weightedTitleBoost else
            idfValue), 1)
20    }
21    //reduce over documents to sum up the vectors, weights and wordsCount
22    .reduceByKey { case ((vec1, weight1, count1), (vec2, weight2, count2)) =>
23      (vec1 + vec2, weight1 + weight2, count1 + count2) }
24    // Don't calculate document vectors if number of words are below threshold
25    .filter { case (_, (_, _), numberOfWords) => numberOfWords >= minWordsCount }
26    // Divide document vector over the weights (Equation 6.1)
27    .map { case (documentId, (vector, numberOfWords)) =>
28      (documentId, (vector / numberOfWords.toDouble).toArray)
29    }
```

Listing A.1: RDD workflow to calculate the document vectors in production pipeline

Listing A.1 shows the Spark workflow for generating the vectors with comments. Note that $reduceByKey$ and map condition should be used since $groupBy$ is very expensive. Spark's lazy evaluation calculates data rows only when it needs them, groupBy however would have to finish transforming all the data rows before going to the next operation, creating a bottleneck at this streaming stage.

A.6 Coding pitfalls

In this section, we list some of the coding suggestions that can help decreasing execution time and save resources. The less time experiments take to execute, the more data can be squeezed in, more hyperparameters can be tested simultaneously, especially when dealing with huge amount of data like XING's.

A.6.1 Doc2Vec

Doc2Vec in Gensim library works by assigning a list of words in the document, along with the document ID. One coding pitfall is that an array of a single long String is much more memory efficient the same long String but divided over multiple cells in the array.

Early tryouts with Doc2Vec models didn't make it to convergence, because the algorithm used all the 32 GB of RAM and crashes when it requires more. A better approach is to do the preprocessing while reading from Disk. Listing A.2 shows a ReadFromDisk class that creates an iterator such that preprocessing is done only on the current line which the iterator is pointing to. This allowed the Doc2Vec to consume 4 GB of RAM at maximum.

```
1  class TitleIterator(object):
2    from nltk.stem.snowball import GermanStemmer
3
4    title_cleaner = re.compile(r"[^0-9a-z]+")
```

```
5     stemmer_de = GermanStemmer()
6
7     def __init__(self, dataset, seed):
8       self.id_tuple = [(job.id, job.title) for job in dataset.jobs.values()]
9       self.seed = seed
10
11    def __iter__(self):
12      documents = [ x for x in self.id_tuple]
13      seed(seed)
14      shuffle(documents)
15      for jid, title in documents:
16        title = re.sub(title_cleaner, "␣", title.lower)
17        title = [self.stemmer_de.stem(w) for w in title.strip().split("␣") if len(w) > 1]
18
19        yield TaggedDocument(title, [jid])
20
21  }
```

Listing A.2: Use of an iterator for Doc2Vec

A.6.2 ANNOY

Using ANNOY is straight forward, the high end programmer creates an index with the needed number of dimensions in the vector space. The vectors are then added sequentially using an numeric index in a look up fashion, i.e Index to vector as shown in Listing A.3.

```
1
2   space_indexer = AnnoyIndex(vector_dimensions = 500)
3
4   for i, key in enumerate(sorted(test_ids)):
5     if len(dataset.jobs[key].vector) > 0:
6       space_indexer.add_item(i, dataset.jobs[key].vector)
```

Listing A.3: Indexing in ANNOY

Initial implementation involved using the document ID as an index e.g 15032152 which killed the virtual machine. After tracing the code, it appeared that ANNOY uses the absolute value of the ID to locate memory for $max(ID) + 1$, which means that document IDs has to be mapped to another incremental lookup table, because retrieving the nearest neighbors from ANNOY is using and retrieving these incremental IDs. A better approach while adding indices is adding the maximum incremental index first, to allocate the memory once to reduce indexing speed.

A.6.3 Apache Hive

Our Dataset already had pairs of ($User_1$,Job_1), approximately 4 million pairs. We can group by Jobs to get Users who interacted with a given a job. But the goal was to do Set intersection between users to get jobs that 3 or more users interacted with. In other words:

```
CREATE TEMPORARY FUNCTION array_intersect;

SELECT  a.posting_id as jobA,
        b.posting_id as jobB,
        array_intersect(a.users, b.users) as common_users
FROM JobsUsersTuples a
CROSS JOIN JobsUsersTuples b
WHERE a.posting_id > b.posting_id -- to speed up by x2
HAVING size(common_users) >= 3;

Query ID = hive_2017022
Total jobs = 1
Launching Job 1 out of 1

--------------------------------------------------------------------------------
VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1            RUNNING      1          0        1        0       0       0
Map 2 ..........  SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/02  [==============>>----------] 50%   ELAPSED TIME: 7184.90 s
--------------------------------------------------------------------------------
```

Listing A.4: Dataset generation Hive Script

Hive works by splitting the data and preprocessing it concurrently using Mappers, then collecting the results using Reducers. Cross Joins on such a table would produce 4 million times 4 million rows before filtering, which is very time consuming ($O(N^2)$ time complexity). Hive would have to go through each row for every row in the table, which completely makes Map and Reduce features obsolete in such operation. In the example, only 1 mapper and 1 reducer are used, since the operation can't be parallelized.

The second approach was to create a User Defined Function (UDF) in Hive that stores the table in the mapper's memory, which transforms the problem to $O(N)$ space and $O(N)$ time complexity. As shown below:

```
class UDTFCrossJoinSetIntersection extends GenericUDTF{
private[this] var jobAConverter: PrimitiveObjectInspector  = _ // reads current row, cast it to JobID
private[this] var usersConverter: ListObjectInspector      = _ // reads current row,
                                                               // cast it to List of Users
private[this] var MemoryData: Map[Int, util.List[_]]       = _ // copy of the entire table in memory

override def initialize(argOIs: Array[ObjectInspector]) = ... // outputs to postingA, postingB, common_users
override def configure(mapredContext: MapredContext)    = ... // reads table using HCat and fills in 'MemoryData'
override def process(jobA: int, usersA: Array[Int])     =     // Joins in memory with Set intersection
  MemoryData.foreach { case(jobB, usersB) =>
    if (usersA.intersect(usersB).length > 1)
      forward(jobA, jobB, usersA.intersect(usersB))
}
```

Listing A.5: Memory UDF in Hive

Using the UDF is a simple function call over the table of pairs. However, Hive works on top of a cluster resource manager called YARN, which tries to optimize queries whenever possible. If a query can be executed locally without the need for Map/Reduce, it will execute the query locally, resulting in an error using our UDF since it's mapper dependent. The behavior should be switched off using:

```
SET hive.fetch.task.conversion=none;
```

Unfortunately, the memory consumption was too high that it affected the scheduled queries for production at XING, it wasn't reproducible and at that stage in the thesis, multiple iterations of dataset generation had to be executed, it took over 2 hours and it completely blocked the Hive queues for other operations at XING.

The final solution was generating the dataset using Map/Reduce like approach. Where we do a join over the table using the userIDs, which would "emit" 3-tuples of ($userID$, $jobID_1$, $jobID_2$). we can group over $jobID_1$ and $jobID_2$ to group all userIDs in a reduce-like fashion. In this case, we join over IDs and group over IDs, with no need for cross join. Emitting tuples and grouping them is how Hive do Map/Reduce operations on a lower level.

First operation took over an hour before terminating it manually to investigate the distribution of userIDs in the dataset. According to the distribution of userIDs (can't be listed to comply with XING policies), the number of clusters had to change from 128 to 97. Which is a prime number big enough to split the distribution of userIDs evenly. The workflow took 14.31 seconds to finish, down from over 2 hours.

```
CREATE TABLE JobUserPair (posting_id int,user_id int)
CLUSTERED BY(user_id) INTO 97 BUCKETS
STORED AS ORC;

-- filling in the tuples
INSERT OVERWRITE TABLE JobUserPair ...

SET mapred.reduce.tasks=97; --sets number of reducers

CREATE TABLE Emitter AS
SELECT  a.posting_id as postingA,
b.posting_id as postingB,
a.user_id    as common_user
FROM JobUserPair a
JOIN JobUserPair b
  ON a.user_id = b.user_id
WHERE a.posting_id != b.posting_id; -- to avoid duplicates


CREATE TABLE Reducer AS
SELECT  postingA,
        postingB,
        collect_set(common_user) as common_users
FROM Emitter
GROUP BY postingA, postingB
HAVING size(common_users) >= 3;

----------------------------------------------------------------------------------------------
VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      2          2        0        0       0       0
Map 3 ..........   SUCCEEDED      2          2        0        0       0       0
Reducer 2 ......   SUCCEEDED     97         97        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 14.31 s
----------------------------------------------------------------------------------------------
```

Listing A.6: Set Intersection in Hive

A.6.4 General Tips

- Code should be parallelized in every possible situation, number of PoolExecutors should be 1.5 times the number of cores on the machine. The function to be parallelized should return an output, not write directly in Data Structure to avoid dirty writes.

```
import concurrent.futures
executor = concurrent.futures.ProcessPoolExecutor(10)
futures = [executor.submit(example_function(), input_value) for input_value in list_of_values]
concurrent.futures.wait(futures)
```

Listing A.7: Multiprocessing in Python

- Researcher should get familiar with the Data Structure used and how to iterate them effectively. For instance, if it is no longer needed to add more items to a given Set, accessing FrozenSet is faster than a normal mutable Set.
  In the example below, list intersection can be written in 3 different ways. Using list comprehension is not suitable for big lists, but converting the lists to sets and do set intersection performs better, since intersection is based on hash and not absolute values.

```
# list comprehension
intersection = [a for a in x if a in y]

# functional way achieve better performance
intersection = filter(lambda a:a in x, y)

# converting to set intersection is even better
intersection = list(set(x) & set(y))
```

Listing A.8: 3 different ways to write list intersection

- Progress bars and logging are extremely useful when dealing with time consuming problems, it allows the Researcher to take a decision whether the experiment should continue running for few more hours or restart it with different setup.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
logging.debug("test")

def progress_bar(message, current, total, frequency = 10):
  percentage = (float(current)/total)*100
  if format(percentage % frequency, '.2f') == '0.00' or current + 1 == total or frequency == 1:
    print "%s\t\t\t\tprogress: %s %%" % (message,str(percentage))
```

Listing A.9: Progress Bar

Bibliography

Abdelwahab, O. and Elmaghraby, A. (2016). UofL at SemEval-2016 Task 4: Multi Domain word2vec for Twitter Sentiment Classification . In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016) at NAACL-HLT 2016*, pages 164–170, San Diego, California.

Abel, F., Benczúr, A., Kohlsdorf, D., Larson, M., and Pálovics, R. (2016). Recsys challenge 2016: Job recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York: ACM.

Adomavicius, G. and Kwon, Y. (2012). Improving aggregate recommendation diversity using ranking-based techniques. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):896–911.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923.

Baudiš, P., Pichl, J., Vyskočil, T., and Šedivỳ, J. (2016). Sentence pair scoring: Towards unified framework for text comprehension. In *CoNLL*, Berlin, Germany.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Blei, D. M. and Lafferty, J. D. (2007). A correlated topic model of science. *The Annals of Applied Statistics*, pages 17–35.

Blei, D. M. and Lafferty, J. D. (2009). *Text mining: classification, clustering, and applications*, chapter Topic Models, pages 71–89. CRC Press.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research (JMLR)*, 3:993–1022.

Bordag, S. (2008). *A Comparison of Co-occurrence and Similarity Measures as Simulations of Context*, pages 52–63. Springer Berlin Heidelberg, Berlin, Heidelberg.

Chen, M. (2017). Efficient vector representation for documents through corruption. In *ICLR*, Toulon, France.

Dai, A. M., Olah, C., and Le, Q. V. (2014). Document embedding with paragraph vectors. In *Proceedings of Deep Learning and Representation Learning Workshop at NIPS*, Montreal, Canada.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

Fallgren, P., Segeblad, J., and Kuhlmann, M. (2016). Towards a standard dataset of swedish word vectors. In *Sixth Swedish Language Technology Conference (SLTC)*.

Garten, J., Sagae, K., Ustun, V., and Dehghani, M. (2015). Combining distributed vector representations for words. In *VS@ HLT-NAACL*, pages 95–101.

Hagan, M. T. and Menhaj, M. B. (1994). Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6):989–993.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Hong, J. and Fang, M. (2015). Sentiment analysis with deeply learned distributed representations of variable length texts. Technical report, Technical report, Stanford University.

Huang, A. (2008). Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56.

Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882, Jeju, Korea.

Jansen, B. J., Spink, A., Bateman, J., and Saracevic, T. (1998). Real life information retrieval: A study of user queries on the web. In *ACM SIGIR Forum*, volume 32, pages 5–17. ACM.

Ji, S., Satish, N., Li, S., and Dubey, P. (2016). Parallelizing word2vec in shared and distributed memory. *arXiv preprint arXiv:1604.04661*.

Koehn, P. and Knight, K. (2003). Empirical methods for compound splitting. In *Pro-*

ceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1, pages 187–193. Association for Computational Linguistics.

Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. M. (2009). Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1):140–181.

Kottur, S., Vedantam, R., Moura, J. M., and Parikh, D. (2016). Visual word2vec (vis-w2v): Learning visually grounded word embeddings using abstract scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4985–4994.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Proceedings of the 12th international conference on machine learning*, volume 10, pages 331–339.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196.

Lee, D.-T. and Wong, C. (1977). Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1):23–29.

Levy, O. and Goldberg, Y. (2014). Dependency-based word embeddings. In *ACL*.

Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Lewis, D. D., Schapire, R. E., Callan, J. P., and Papka, R. (1996). Training algorithms for linear text classifiers. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306. ACM.

Liu, Y., Liu, Z., Chua, T.-S., and Sun, M. (2015). Topical word embeddings. In *AAAI*, pages 2418–2424.

Maillo, J., Ramírez, S., Triguero, I., and Herrera, F. (2017). knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data. *Knowledge-Based Systems*, 117:3–15.

Melamud, O., McClosky, D., Patwardhan, S., and Bansal, M. (2016). The role of context types and dimensionality in learning word embeddings. *arXiv preprint arXiv:1601.00893*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *ICLR Workshop*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Nagaraj, R. and Thiagarasu, V. (2014). Correlation similarity measure based document clustering with directed ridge regression. *Indian Journal of Science and Technology*, 7(5):692–697.

Paine, T., Jin, H., Yang, J., Lin, Z., and Huang, T. (2013). Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186*.

Pang, B. and Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics.

Reynar, J. C. (1999). Statistical models for topic segmentation. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 357–364. Association for Computational Linguistics.

Riedl, M. and Biemann, C. (2016). Unsupervised compound splitting with distributional semantics rivals supervised methods. In *HLT-NAACL*, pages 617–622.

Robertson, S. (2004). Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520.

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Rus, V., Niraula, N., and Banjade, R. (2013). Similarity measures based on latent dirichlet allocation. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 459–470. Springer.

Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Seroussi, Y., Zukerman, I., and Bohnert, F. (2011). Authorship attribution with latent

dirichlet allocation. In *Proceedings of the fifteenth conference on computational natural language learning*, pages 181–189. Association for Computational Linguistics.

Shoaib, M., Daud, A., and Khiyal, M. (2014). An improved similarity measure for text documents. *J. Basic Appl. Sci. Res*, 4(6):215–223.

Stamatatos, E., Fakotakis, N., and Kokkinakis, G. (1999). Automatic extraction of rules for sentence boundary disambiguation. In *Proceedings of the Workshop on Machine Learning in Human Language Technology*, pages 88–92.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, pages 111–147.

Wajeed, M. A. and Adilakshmi, T. (2011). Different similarity measures in semi-supervised text classification. In *India Conference (INDICON), 2011 Annual IEEE*, pages 1–5. IEEE.

Wan, X. (2007). A novel document similarity measure based on earth movers distance. *Information Sciences*, 177(18):3718–3730.

Yao, Y., Li, X., Liu, X., Liu, P., Liang, Z., Zhang, J., and Mai, K. (2017). Sensing spatial distribution of urban land use by integrating points-of-interest and google word2vec model. *International Journal of Geographical Information Science*, 31(4):825–848.

Ziering, P. and van der Plas, L. (2016). Towards unsupervised and language-independent compound splitting using inflectional morphological transformations. In *HLT-NAACL*, pages 644–653.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Wirtschaftsinformatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen - benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 25.08.2017

Ahmed Elsafty

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 25.08.2017

Ahmed Elsafty