University of Zurich[UZH]

# Document Embedding Models - A Comparison with Bag-of-Words

Master Thesis September 10, 2018

**Robin Stohler**
Zurich ZH, Switzerland

Student-ID: 10-296-275
robin.stohler@gmail.com

Advisor:
**Matthias Baumgartner**

Prof. Abraham Bernstein, PhD
Institut für Informatik
Universität Zürich
http://www.ifi.uzh.ch/ddis

# Acknowledgements

First, I would like to thank Prof. Dr. Abraham Bernstein for giving me the opportunity, after the master basis module and the master's project to also write my master's thesis at the Dynamic and Distributed Information System Group of the University of Zurich.

My highest appreciation I express to Matthias Baumgartner, my supervisor, who helped me navigate through difficulties, who could turn my despair into motivation and sparked inspiration through the many exciting talks.

Moreover, I am very grateful for all the support of my friends and family who encouraged me throughout my studies.

# Zusammenfassung

Worteinbettungen haben die Möglichkeiten im Bereich der natürlichen Sprachverarbeitung und des maschinellen Lernens komplett verändert und neue Türen für viele Anwendungen geöffnet. Eine davon ist die Erstellung von Dokumenteneinbettungen mit dem Doc2Vec-Algorithmus auf Basis von Word2Vec. Diese dicht verteilten latenten Vektoren ermöglichen es, mit Text besser zu arbeiten als mit älteren Textvektorisierungsverfahren wie Bag-of-Words (BOW). In dieser Arbeit werden verschiedene Baseline-Methoden in verschiedenen Kategorien mit Doc2Vec verglichen. Um schliesslich die Nützlichkeit dieser älteren Ansätze nach dem jüngsten Umbruch im NLP-Bereich zu beurteilen. Aber empirische Ergebnisse zeigen, dass BOW, der mit einem starken Klassifikator verwendet wird, besonders in kleineren Datensätzen besser ist als Doc2Vec. Ausserdem wird ein Ansatz vorgestellt, um die Dimensionalität von BOW zu reduzieren.

# Abstract

Word embeddings changed the possibilities in the field of Natural Language Processing and Machine Learning completely, opening new doors for many applications. One is the creation of document embeddings with the Doc2Vec algorithm based on Word2Vec. These dense distributed latent vectors allow to work with text in a better, more meaningful way compared to older text vectorization processes such as Bag-of-Words (BOW). In this thesis, a variety of baseline methods are compared in different categories against Doc2Vec. To finally asses the usefulness of these older approaches after the recent upshake in the Natural Language Processing field. Empirical results show that BOW used with a strong classifier is especially in smaller datasets better than Doc2Vec. Additionally, an approach to reduce the dimensionality of a BOW is presented.

# Table of Contents

# 1

# Introduction

Bag of Words (BOW) is a widely used technique to formalize texts or documents stripping away the actual terms and constructing a vector space representation. Words are encoded as one-hot vectors where the vector space is as big as the number of unique words in the corpus. However, with the one-hot encoding problems arise such as the curse of high-dimensionality[Bengio et al., 2003], sparsity, but also semantic problems such as *term dependencies* and *vocabulary mismatch* [Manning et al., 2008]. Nonetheless, these text representations are used for the various task, e.g., spam-filtering, information retrieval, text similarity measurement or text classification. Different extensions and weighting schemes were added over the years to account for some of the problems in the BOW model.
In 2013 [Mikolov et al., 2013a, Mikolov et al., 2013b] proposed a more powerful type of word encoding model, with their algorithm Word2Vec. Word embeddings are low dimensional vectors that are trained through a machine learning (ML) method which moves similar words closer to each other in the generated vector space. The word "embedding" is borrowed from a mathematical concept, conceptually the high dimensional one-hot encoding is embedded into a low-dimensional dense vector space.
Soon the context-window based method Word2Vec was complemented by GloVe [Pennington et al., 2014b]. GloVe keeps a global statistic of co-concurrence and uses dimensionality reduction to retrieve the word embeddings. While these two methods are using different approaches the basis is similar and the word vectors are comparable in accuracy and similarity tests. Word vectors changed the field of Natural Language Processing (NLP) and started a trend in NLP with the combination of neural networks, text and the newly created word embedding models [Goth, 2016]. Other models mitigating problems of the initial Word2Vec were also created [Bojanowski et al., 2016, Ji et al., 2015].

Shortly after, the same team that created Word2Vec proposed the Doc2Vec model also to derive embeddings from whole documents or paragraph [Le and Mikolov, 2014]. While not having such an impact in the community, the implementation, Doc2Vec [1] led to a new field of embedding more than just words, but also sentences, paragraphs and documents [Pagliardini et al., 2017, Chen, 2017, Wu et al., 2017, Kiros et al., 2015, Kusner et al., 2015].

Document embeddings become convenient when tasks should be done on a document level or to asses whether two documents are similar. A feature that makes them superior

[1] also known as par2vec

compared to BOWs is the fixed length of the embedding which can be set in the learning phase. This is especially useful for downstream neural network tasks that require a fixed network topology. However, significant drawbacks are the existence of the many hyperparameters and that various models need to be combined to get good results. These hyperparameters need to be grid searched for optimal results, but these many possibilities also consume a lot of time and resources, besides not showing whether or not only local maxima or global maxima were found in the process. Likewise, the amount of data needed for these embedding models might be an issue in some cases. Unlike the BOW where every dimension in the vector space stands for a word, the dimensions in these embeddings are not interpretable, therefore, the quality of embeddings needs to be further assessed through evaluation. An often used method is the evaluation in a downstream classification task, where the accuracy gives insights into the capabilities of the embedding. Another method is to check the similarity of documents that are categorizable in groups, where documents in the same group should also be closer together. The superiority of the Doc2Vec approach could not be completely demonstrated compared to the hyperparameter free off the shelf text vectorization process BOW.

In this thesis, different baseline models are compared with Doc2Vec to answer the question of whether *BOWs belongs to the past and can entirely be replaced by document embeddings by Doc2Vec*. In this regard, different baseline approaches are compared against Doc2Vec. These assessments are based on classification tasks on different datasets with discriminative features such as number of classes, number of vocabulary, document size and number of documents. For these assessments, similarity checks were completely ignored due to some of the critiques stated in Section 2.3 and the focus on classification tasks.
The following hypotheses are tested to answer the final research question:

- Doc2Vec results from [Mikolov et al., 2013b] can be reproduced.
- Pre-processing is important for higher accuracy since punctuation and stop-words do not help on the classification problem.
- The classification accuracy correlates with the document length.
- Document embeddings outperform baseline methods in typical benchmark setup (classification task).
- Document embeddings save effort since precomputed embeddings can be used in many applications.
- Document embeddings save effort since the same classifier can be used in many applications.
- Document embeddings save time since they reduce the complexity of the documents to reduce the work for the classifier.

This thesis will also provide advice for practitioners on which method(s) to use depending on the scenario, and what the expected outcome will be. As a side result, a

simple neural network decoder is proposed to simplify the complexity of BOWs to a fixed length to reduce the heavy lifting of classifiers in downstream tasks.

The remainder of the thesis is structured as follows: In Chapter 2, related work in the field and background knowledge is discussed. Chapter 3 talks about all applied methods in this thesis and their background. The Chapter "Experimental Setup" 4 introduces the datasets, the various experiments, some details about them and answers the questions, what was measured, how was it measured and why was it measured. Chapter 5 follows the structure of the experiments Chapter and lists all result from the various experiments. Finally, the thesis is completed through the Limitation, the Future Work and the Conclusions.

# 2

# Related Work

In this thesis, the focus is on word and document embeddings, and how they compare to simpler approaches like the text vectorization process BOW. Since the upcoming of Word2Vec [Mikolov et al., 2013b, Mikolov et al., 2013a] and later GloVe [Pennington et al., 2014b] a trend in NLP in combination with ML broke out changing the way computers could now work with texts [Goth, 2016]. After that, a broad series of embedding methods that built upon the ideas of Word2Vec and GloVe were proposed where problems of prior models were mitigated. However, these models also opened up doors for other approaches like the many deep-learning methods that could now use a better representation for words other than only sparse vectors or matrices for texts. Following this, some other concepts are discussed which are important in the field and related to the before mentioned algorithms. The most popular approaches based on words are reviewed in Section 2.1, older and newer methods to represent documents are discussed in Section 2.2. Since some of the mentioned methods are based on word vectors, some of the critis in this field are discussed in Section 2.3. In Section 2.4 some of the possibilities to evaluate these hard to understand vector representations are discussed. Finally, Section 2.5 concludes the chapter with the cutting edge research currently in the field of NLP the neural network encoder.

## 2.1 Word Embeddings

The idea of a distributed word representation existed already since 1986 [Hinton, 1986]. [Bengio et al., 2003] proposed a neural probabilistic language model that could learn dense distributed word vectors but the time was not yet right, the machines not strong enough and no big corpus publicly available. [Collobert and Weston, 2008] went one step further and proposed their deep neural network model trained on a large enough corpus and showed that these word vectors are powerful in many downstream tasks.

As mentioned Mikolov et al. built upon many already known concepts but improved them. The idea behind Word2Vec is that words that occur in similar situations are also similar. The algorithm scans a corpus and maps words that occur together in a fixed size window closer together.

Global Vector or GloVe is probably the second most known way to derive word embeddings from a corpus. Instead of scanning through the corpus with a window, a

co-occurrence-matrix, that counts all words occurring together in a specified word radius, is calculated at first. Vectors are generated With the help of matrix factorization and an objective function that tries to decrease the difference within the dot product of the vectors of two words and the logarithm of their count of co-occurrences, the vectors are generated. According to [Pennington et al., 2014b] there are two worlds of creating word embeddings one world uses matrix factorization like LSA and the other world looks at a local context window as Word2Vec. While GloVe tries to take the best from both of these two worlds and combines it.

Next to GloVe and Word2Vec that create word embeddings based on a pairwise co-occurrence, either through a co-occurrence matrix or a window-based method. WordRank from [Ji et al., 2015] creates its word embeddings through a ranking. WordRank learns the word embeddings for each target word with all its context words ranked by relevance. WordRank keeps a window based training framework, but instead of pairwise comparison, the ranking is used. This ranking makes WordRank much more durable and robust against noise in corpora, making it possible to get good results even if the corpus is small. Nonetheless, this approach is not excelling the other methods in large corpora.

Fast Text is an addition to Word2Vec. Instead of learning the words as whole the words are treated as a bag of characters, and these are split up in n-grams or sub-words. The word boundaries are ignored. To illustrate this: All trigrams of "pear" are "_pe", "pea", "ear" and "ar_". Sub-word embeddings embedding are learned in a Word2Vec fashion. After the subword embeddings are learned, each word gets the sum of all its sub-words as word vector. In this way, rare words or words not in the training set can be better represented or approximated because of n-grams also appearing in other words [Bojanowski et al., 2016]. But this also leads to the strange effect of giving words that happen to have similar characters also a related meaning, even though this effect is reduced because of the sum of the n-grams.

## 2.2 Document Vector Representation Models

Doc2Vec was introduced by [Le and Mikolov, 2014], the same team which also created the Word2Vec algorithm. Doc2Vec extends Word2Vec, using similar adapted approaches as Word2Vec to also learn document embeddings. In [Le and Mikolov, 2014], three different evaluation tasks were performed. Two of the tasks were classification tasks and one information retrieval task. Two movie review datasets Standford Sentiment Treebank Dataset and IMBD were used for the sentiment classification task and another, unspecified, data set to compare queries for the information retrieval task. In [Dai et al., 2015], a performance test with two different publicly available corpora, Wikipedia and arXiv, was made. Qualitative results evaluated, where document similarities of the Doc2vec embeddings are compared to LDA topic vectors using cosine similarity for Doc2Vec and Hellinger Distance for LDA. Also, vector arithmetic with "Lady Gaga" - "American" + "Japanese" was performed to retrieve any famous Japanese singer, which was successful. Quantitative results were gained from triplets where two articles were closer to each other

than another randomly picked article based on the structure of Wikipedia. Doc2vec outperformed the different three approaches LDA, averaged word embeddings and BOW. In [Lau and Baldwin, 2016] Doc2Vec embeddings are compared with averaged Word2Vec, an n-gram model and two competitor models Skip-thought [Kiros et al., 2015] and the Paraphrase Database[Ganitkevitch et al., 2013].

A simpler method to derive vectors from text is BOW that counts word occurrences and creates from that a sparse vector. Where words are arbitrarily assigned to indexes in the vector. The BOW approach suffers from two problems: *Term dependencies* and *vocabulary mismatch* [Manning et al., 2008]. *Term dependency* is the effect that for a BOW the plural and singular of the same word is something completely different and in no way related to each other. In contrast, the two-word embedding models Word2Vec and GloVe for map singular and plural close to each that subtracting one from the other would return nearly-null [Rogers et al., 2017] which is a vector space analogy for a synonym. The other problem, *vocabulary mismatch*, occurs when two documents are compared from different disciplines. It is expected that stop words are shared but most of the interesting words that would help to classify a document are not. First, blows up the vector space of the BOW, and second, for a BOW, the two documents can only be similar or dissimilar. Besides, models that build upon word embeddings a document about ML and software engineering can be understood as not the same but similar because most of the words are expected to co-occur together, and therefore they should also be mapped closer together in the vector space.

Latent Semantic Analysis (LSA)[1] is a method that builds a TF-IDF weighted matrix with columns as documents and rows as words. This matrix is then simplified with singular value decomposition (SVD). The application of SVD returns three matrices $UDV^T$, where the columns of $U$ and $V$ are orthonormal and the matrix is $D$ diagonal with positive real entries, the singular values. This reduces the number of rows (words) while keeping the columns constant. LSA is often used in information retrieval or document classification tasks [Landauer et al., 1998].

Latent Dirichlet Allocation (LDA) is a generative probabilistic model of a corpus. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. Before running the algorithm, the number of topics is set and then documents are represented as a mix of these latent topics. The mix of these latent topics could then be used as document vectors, the vector size can be changed through the number of topics that should be extracted. A benefit that LDA has over other vector space models is that the vector dimension is better understandable than one of the many dimensions of other word or document embedding approaches. The LDA method was also used as a baseline in [Dai et al., 2015]. LDA can also be applied to images where images do not contain abstract topics but skies, mountains, meadows etc. [Blei, 2003].

The Word Mover's Distance (WMD) is a different approach to calculate document similarity next to the cosine similarity. The intuition behind this method is that the word vectors of two documents are compared with each other and a distance is calculated

[1] Also known as Latent Semantic Indexing(LSI)

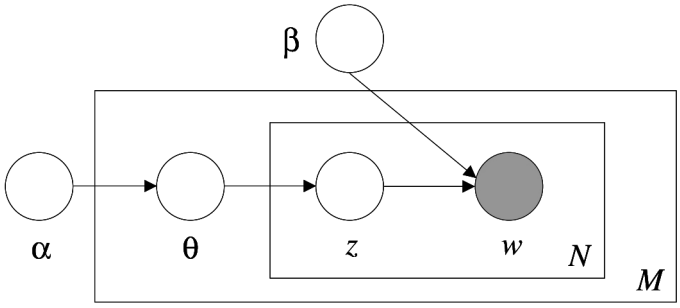



Figure 2.1: A typical graphical representation of LDA. The boxes are called "plates" and they are representing replicates. The outer plate stands for the documents, and the inner plate represents a repeated choice of documents and words in a document. [Blei, 2003]

in order to move the words from one document in the space of the other. The Method relies on the Earth Mover Distance and learned word embeddings of the document. The use of word embeddings helps this approach to be relatively word independent. This means that two texts, one about ML, the other one about software development, will not share many words, however since the words in these texts might co-occur in other documents, the word embeddings are not too far from each other which would result in a closer distance. In contrast to a BOW with cosine similarity where the similarity is solely based on the extent shared words. According to the author, advantages of this method are that it is hyperparameter-free and it showed good performance in information retrieval and classification tasks [Kusner et al., 2015].

Doc2VecC is another approach to learn document embeddings, the C in the name stands for corrpution. Unlike Doc2Vec, Doc2VecC does not learn the document embedding but rather averages aof word embeddings and optimizes these word embeddings to create good document embeddings. The corruption model is a regularization process that favors rare words over common or non-descriptive ones forcing these embeddings to be close to zero. In this way the process avoids over-fitting, and also reduces the computational cost. [Chen, 2017]

StarSpace is, according to the authors, a general-purpose neural embedding model that can solve a wide variety of problems: like text classification, information retrieval or web search, collaborative filtering-based or content-based recommendation, embedding of multi-relational graphs, and learning word, sentence or document level embeddings. Entities are represented as a bag of discrete features (bag-of-features). Interestingly the StarSpace model can compare entities of different types together since entities can be split up into finer grained entities (like a document is a bag of sentences, a sentence is a bag of words, a word can be a bag of letters etc.) This is an promising multipurpose approach which could become fascinating for various data science tasks.[Wu et al., 2017]

In 2015, [Kiros et al., 2015] presented their encoder skip-thoughts. Skip-thoughts is

inspired by the Skip-gram model from Word2Vec, but instead of creating word vectors and predicting the surrounding words, this encoder tries to encode sentences to a vector and predict on a sentence level surrounding sentences.

Sent2Vec follows a similar approach comparable to Doc2Vec of [Le and Mikolov, 2014] or the Continous Bag-of-Words (CBOW) of [Mikolov et al., 2013b] but it uses compositional N-gram features. A sentence embedding is the average of all word and n-gram embeddings in one sentence, but these word vectors are especially trained to return meaningful averaged document vectors [Pagliardini et al., 2017].

## 2.3 Critique of word embedding evaluation methods

Since some of the discussed models are based on word embeddings, it is essential to include the critiques of these models following a review of three papers criticizing the advertisement and the claims made by the word vector community.

Analogies are often used to describe the benefits or the added value of the vector space representation of words in word-embedding methods like Word2Vec. These vector models suggest that words like "debug" and "debugging" should have a similar vector distance and direction as "read" and "reading." In [Linzen, 2016], different methods that compared the relationship of four words and their corresponding vector arithmetic are evaluated. Following the famous example $King - Men + Woman = Queen$ illustrated in a standard formula in equation 2.1.

$$a^* - a + b = b^* \tag{2.1}$$

Where $a$ and $a^*$, and $b$ and $b^*$ are semantically related and the retrieved result $x^*$ should be the same as $b^*$. While the vanilla version of this vector calculation does not perform well at all, simpler methods that ignore $a$, $a^*$, or $b$ in the set of possible perform better. The evaluated methods are:

- The previously stated **Vanilla** (Eq. 2.1), where nothing was changed.
- The same as above **Add** where $a$, $a^*$, $b$ are ignored.
- **Only-B** where just the closest neighbor of $b$ without $a, a^*, b$ were considered.
- **Ignore-A** $a^* + b$ where a is completely ignored.
- **Add-Opposite** $-(a^* - a) + b$
- **Multiply** illustrated in equation 2.2.
- Reversed **Add**
- Reversed **Only-B**

$$x^* = \underset{x' \notin \{a, a^*, b\}}{\operatorname{argmax}} \frac{cos(x', a^*)cos(x', b)}{cos(x', a)} \tag{2.2}$$

The authors explored that the accuracy of this relationship is highly dependent on category of the analogy. Common capitals (Athens, Greece), All capitals (Abuja, Nigeria), Nationalities (Albania, Albanian), Gender (boy, girl), Singular to plural (banana, bananas) and Adjective to comparative (bad, worse) worked good. However, Adverb to adjective (amazing, amazingly), Adjective un-prefixation (acceptable, unacceptable) and currencies(Algeria, dinar) do not work that well. The method **Multiply** returned slightly better results than the **Add**.

In [Finley et al., 2017] the authors measured Reciprocal of Rank (RR) instead of accuracy which is supposed to act as a "softer" version of accuracy. Acuracy is not able to account for analogy difficulty since some analogies like singular and plural are easier as singular and plurals are in fact the closest words in the vector space to each other. RR can also be helpful to measure and quantify near misses which would be reduced to zero when using just accuracy. In the paper, the RR values of two methods a baseline similar to the **Only-B** mentioned above but they also included $a$ to the formula for RR illustrated in equation 2.3 and an alternative approach the RR of a similar calculation like the **Add** as mentioned earlier.

$$rank_{base} = min(rank(\underset{w \in V}{\operatorname{argmax}}(w \cdot w_2)), rank(\underset{w \in V}{\operatorname{argmax}}(w \cdot w_3))) \tag{2.3}$$

The analogy tasks were split into four groups: Inflectional morphology, Derivational morphology, Lexical semantics and Named entities. Both results per analogy task were plotted on both sides of the graph and connected through lines, illustrated in Figure 2.2. Interestingly, most of the steep lines belong to well-known examples while it seems that other analogies do not work the same way. All these examples in papers that show how good these vector arithmetic works might just be corner cases that work much better then most others. The authors state that vector arithmetic with word embeddings works best when the co-occurring words are limited to only a small number of words. This is reasonable because otherwise the result space has a wider range of possible solutions space [Finley et al., 2017].

[Rogers et al., 2017] did research in this field of proportional analogy problems ($(King - Man) + Woman = Queen$ ). They split the analogy tasks into the same four groups and compared the results of different methods for these four groups similar to the previous paper. They follow the example of "marry" and "remarry" and critiqued several weak points of these analogy tasks. First, they looked at the 3CosAdd (referred to as **Add** earlier). They used different verbs e.g., do = $a$, and redo = $a*$ , $b$ was marry and the expected result should be $b*$ =remarry. Interestingly, in the example shown remarry was never reached even though it would have been close to the returned result. A possible reason why this test failed might be because of a messy corpus. However, other problems are also apparent. Semantics and linguistics are messy and often not quite as simple and regular as a scientist might wish. The arithmetic used to calculate its proportional analogy is commutative and could be used the other way round like

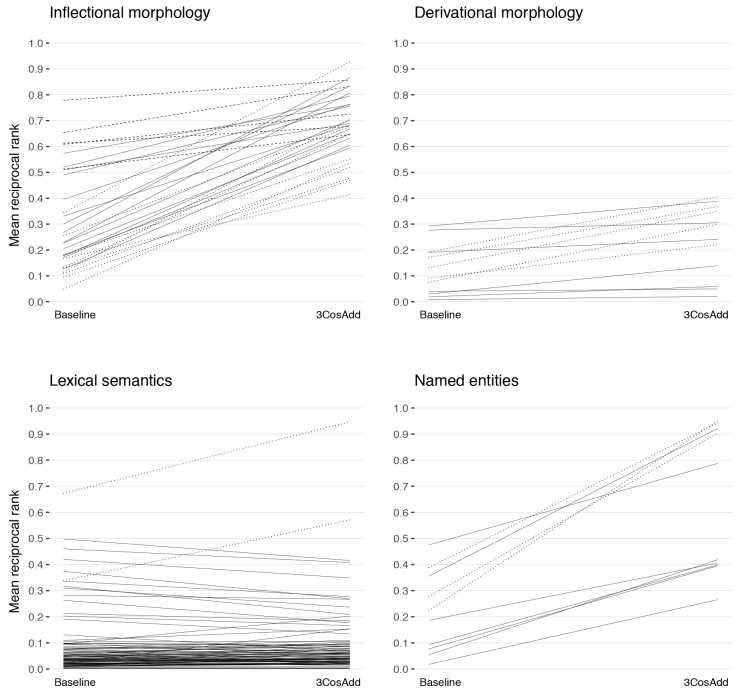



Figure 2.2: Mean reciprocal rank shifts of the four sub categories using the baseline and **Add** [Finley et al., 2017].

$(Woman - Man) + King = Queen$ which only remotely makes sense. Another example here is $Man - Woman$ which should result in "femaleness" or "maleness". This semantic feature defies definitions, which only are applicable for some parts of the words and imposes an unrealistic binary opposition. The last problem mentioned is that even analogies have their ambiguities. The example from the paper is "A trout is to river as a lion is to ____." Some could say "den" other could say "savanna" depending on the understanding of what is the river actually for the trout, is it a home or a habitat.

As illustrated with the three papers, above analogies and language in general are indeed more difficult than simple vector arithmetic. The ambiguity of language and the fact that not everything can be measured in co-occurrence is also a factor. Another flaw of the vector embeddings is that these base model cannot distinguish between different concepts of the same word as a bank can be a financial institut but also a furniture.

## 2.4 Evaluation of Embeddings

Evaluation of the embeddings is necessary, unlike features engineered by humans, e.g., counting words, counting punctuation signs and others, these n-dimensional vectors do not contain interpretable figures, but order words or documents on high-dimensional dense vector spaces closer together or further apart. Therefore, evaluation is needed to

help asses the quality of an embedding or to be able to compare different approaches with each other. These quality assessments come in two flavors: Intrinsic and extrinsic evaluation.

**Intrinsic Evaluation** Intrinsic evaluation is the evaluation of the output with a specific intermediate task, in contrast to the task the model was built for (downstream task). For word vectors, this could be an analogy completion task as shown in Table 2.1 or similarity checks. These tasks are simple and therefore fast to compute. They help to understand the produced vector spaces, but there should also be a positive correlation of success rate with the real downstream tasks to determine the usefulness of the system, which can be hard to find or hard to evaluate whether such a correlation exist. Another problem is that for intrinsic evaluation datasets often have to be created or special annotated datasets have to be used.

**Analogy evaluation:**

Table 2.1: Table with word analogy examples

| **Analogy task** | **Example** | **Modus** |
|---|---|---|
| Capital City 1 : Country 1 : : Capital City 2 : Country 2 | Chicago : Illinois : : Houston Texas | semantic |
| Gerund 1: Past Participle 1 : : Gerund 2: Past Participle 2 | dancing : danced : : decreasing decreased | syntactic |

**Correlation evaluation:**
Another method of intrinsic evaluation of word vectors would be the correlation between similarity in the embedding space and one from human labeled data on how similar words are from 0-10 where zero is not similar and ten means very similar. WordSim a famous word similarity data set was created by [Finkelstein et al., 2002].

**Extrinsic Evaluation** Extrinsic evaluation is the evaluation of the real tasks the system was built for. Unfortunately, the final assessment can be slow to compute. Optimization for an extrinsic task might not give answers to which subsystem is not performing well, since the model creation part and the classifier are involved. Hence, an intrinsic evaluation is needed. Examples of extrinsic tasks depend on the problem that should be solved through ML, but usually, it is a classification problem. This can be handled through linear or nonlinear classifiers.

**Evaluation of Document vectors** Evaluation of document vectors is essential because these vectors are not human readable or even nonsensical for a human reader. Therefore, the features generated from documents have to be fed into another machine learning task to evaluate the real benefit and performance in these tasks. Compared to word vectors, where high cosine similarity of words can be assessed easily whether or not this high value makes sense, document vectors are much more complicated because

they can be understood as a compound of words in a text and an encoding that transforms these words into a vectors. This makes it much harder to compare these document vectors easily. Following below, different tasks used in the field are described.

**Classification tasks** A classification problem tries to add a label to input data. The input data might be a sequence, a text, an image or anything else that has a data representation. A classification problem is a supervised learning method. The data that is used to generate the model is split in two, sometimes three parts. A bigger part of the data is used as training data and the rest as test data. The test data stays untouched while the training data can be augmented, or noise added in order to create a more robust system. The model is fed with the training data. In the learning process it tries to derive rules from this data to successfully predict the labels of the training set. As soon as the training is finished, the test or evaluation data is fed to the model, which then tries to predict the classes.

The accuracy of the training and test datasets are compared. In case the model fits the training sample perfectly but underperforms in the test set, the model overfits the training data. Overfitting is not what the model is intended to do. A remedy for over-fitting is regularization which makes the model more applicable in a wider range of applications also for other sets or the real world. There exist a wide range of Classifier methods such as decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and naive Bayes classifiers.

N-fold cross-validation can be used to make the classification task independent of the train-test split. Here all the data is split into N pieces, and each of the pieces acts once as a test set and the rest as the training set, the average of the N computed accuracies is then the accuracy of the classifier. To prohibit the possibility of unequal distribution of classes which might result in wrong decisions of the classifier the distribution of classes can be stratified such that each split can represent the class distribution of the whole data set.

**Sentiment analysis** Sentiment analysis is a supervised learning problem, and in principle a classification task. The goal is to successfully predict the sentiment, usually of texts. The classification can be either just positive or negative, binned steps of the continuum between positive and negative. In sentiment analysis, single words (also called unigrams) play a smaller role but n-grams are more critical since the existence and position of a single ”no” can change the whole meaning of a sentence.

**Semantic Relatedness** Semantic relatedness is an intrinsic evaluation which is important for document embeddings because similarity results for words are better understandable than similarity results of texts, because text are longer and the extend of relation of two text is harder to asses than whether ”toad” or ”frog” should be close or far from each other.

Possible methods are sub sampling of groups where some documents should be close and one should be distant. Another possibility is to query a document to get the nearest

neighbors and asses wheter they are close or not. But these methods all rely all on the knowledge of the group or class.

Another method that was used by [Dai et al., 2015] was the vector arithmetic approach similar to the famous King and Queen example but with the embeddings of documents.

## 2.5 Neural Encoder

**Encoder and Decoder Architecture** Encoder-Decoder is a deep neural network existing in different flavors such as simple dense-NN, CNN, RNN, LSMT and GRU. An auto-encoder is a special variant of the encoder-decoder architecture where the input is the same as the output. Figure 2.3 shows an example architecture that simplifies the output and then blows it up again to the original size. The encoder takes as input features and transforms them into a feature map, a vector or a tensor. The decoder tries to match the output of the encoder with the initial input or the wanted output. Encoder-decoder is an unsupervised ML method. These systems try to minimize the loss between the input and the intended output. This technique is used for image de-noising, dimensionality reduction or general compression of data, but in NLP these systems can be used for translation, summarization or chat bots.

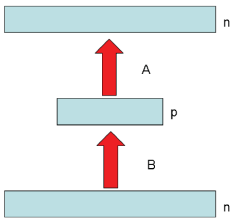



Figure 2.3: An n/p/n Autoencoder Architecture [Baldi, 2011].

**Attention Model** The attention model is a novel model in ML developed by [Bahdanau et al., 2014 that in essence tells the ML algorithm how much attention should be paid to each input parameter and at which time step. Coming from an encoder-decoder architecture where a text is encoded into a fixed length vector and then decoded into another language, in the decoding phase the network is forced to generate a massive sequence of words from only one encoded input data. This generally works well for shorter sentences, but the performance decreases the more prolonged the sentences get.

The attention model here acts how even a human translator would work, translating one part of a sentence after another, combining everything for the final translation, where also each of the previously translated parts adds information to the result of the current part. In Figure 2.4 a graphical example is illustrated. The amount of attention $y^{<t>}$ should pay to $a^{<t'>}$ is $\alpha^{<t,t'>}$, where $\alpha^{<t,t'>}$ is a soft-max function of $e^{<t,t'>}$ and $e_{ij}$ generated by the alignment model:

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.4}$$

$a$ is a neural feed-forward network that is trained together with all other components of the proposed system.

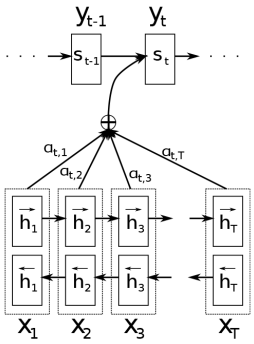



Figure 2.4: A graphical illustration of the attention model trying to generate the $t$-th target word $y_t$ given a source sentence $(x_1, x_2, ..., x_T)$ [Bahdanau et al., 2014]

Unfortunately, a weakness of the attention model is that the resources used grow quadratic with the length of the sentence.

Attention is not only used in machine translation but can also be used for other machine learning tasks like image recognition, where some parts of an image are more important than other parts to determine what is actually on that picture [Xu et al., 2015].

**Universal Sentence Encoder** [Cer et al., 2018] created a pre-trained off-the-shelf encoder that can encode text on a sentence level also using the attention model. This encoder was trained through multi-task learning. Multi task learning with an encoder can be understood as a tree, where the stem is the encoding part and the branches are various ML tasks. They mutually optimize the encoding in order to better perform on their tasks, while the encoder learns the function to derive this encoding from the input.

# 3

# Methods

In this chapter, all the methods used for the experiments are explained starting with text preprocessing, vector space models, used word embedding algorithms, document embedding algorithms and evaluation of the latter two.

## 3.1 Corpus Preprocessing

Text pre-processing is important in order to reduce artifacts of words that do not help but rather obscure any textual task. It also helps to divide the text into smaller portions to reduce the complexity.

**Tokenization** Tokenization is the first of many text preprocessing tasks, and it is therefore essential for all the tasks that follow it, e.g., normalization and parsing. Text tokenization is the process of splitting up the text into chunks (the so-called tokens). These tokens might be words, sentences, paragraphs or documents depending on the needs of the analysis or the downstream application.

**Normalization** Text normalization is a text pre-processing step that includes cleaning text, case conversion, correcting spellings, removing stop words and other unnecessary terms, stemming, and lemmatization. Following below are all the intermediate steps involved in the text normalization process in detail:

**Cleaning** Cleaning describes the process of stripping away unnecessary parts of the text to get better results in downstream tasks due to less ambiguity, which could not be resolved by the computer otherwise. Cleaning can be split into three main tasks: the removal of stop words; the normalization of the cases; and the removal or escaping of punctuation marks. Stop words examples are "how", "to", "the", "you", "a", "lot", "of" just to mention some of them. These words are meant not to give sufficient information to the text and therefore should be removed. The normalization of the cases is an easy task and changes the whole text to lower case. If not, words written with a capital first letter would not be counted as the same word as the one in lower case. The last step is either deleting punctuation marks in the text or escaping these with spaces around.

To reduce the ambiguity between the same words with and without them. An example here would be the sentence "How old are you?". If split by whitespace this sentence the last token is "you?". However, a better result would be to get two tokens from it "you" and "?". In this way, we do not add any unnecessary new words to the corpus that do not occur often and therefore do not help to provide information for any downstream ML tasks.

The tollowing two methods, stemming and lemmatization, can break the "curse of dimensionality" which means that the larger the dataset or the corpus gets, the final representation of the data becomes.

**Stemming** Stemming is the process of reducing morphological variations like inflections and derivations to their stem (also called root form). It is important not to mix the root stem with the root word since the stem might not be a correct lexicographical word but the root word always is. There are different kinds of morphological disparities, e.g., the plurality (woman versus women), gender (bride versus groom), tense (sink versus sunk), conjugation (to sink versus he, it sinks), etc. There are various alternatives to applying stemming. One possibility is to create a lookup-table as a dictionary to translate the forms to their stem. Another option is the Suffix-stripping algorithm by [Porter, 1997] which removes common endings like 'ed,' 'ing,' 'ly' etc. While this approach is fairly simple, more complex cases such as 'goose,' 'geese' might not be covered. Depending on the language stemming may be more or less complicated. In English, this task is relatively easy because of no real cases (similar to French), and few inflections or uniform inflections of verbs, in contrast in German this task is much more difficult because nouns and also verbs are inflected, as well as highly irregular.

**Lemmatization** is another approach to derive roots from words next to stemming. For lemmatization, the task is to acquire the lemma the correct lexicographical word based on the input word. Lemmatization uses unlike stemming a mostly human curated dictionary to look up words and replace it with the correct root or also called lemmas. For lemmatization Part of Speech, tagging is also essential to derive the right word. A common and widely used vocabulary in English is WordNet [Miller, 1995].

**Parsing** Parsing adds a syntactical structure to words in a text and explains further how the word is used in the text and specific context. The algorithm that executing the structural analysis and applying categories to words is called parser. Some examples of these categories are verbs, nouns, prepositions, adjectives, etc. Parsing is a difficult task because of the **lingual ambiguity**. The sentence "I saw the man with the glasses" might be understood in two different ways.

- Through the glasses, I saw a man.
- I saw a man, he wore glasses.

Parsing might build a structural tree and try to solve the occurring problems of ambiguity.

**Filtering Infrequent Words** Depending on the task that has to be fulfilled cutting off words that do not often appear in the corpus might be beneficial. The reason why they do not often appear in the text is that these words contain typos, are names, or rare long words borrowed from Latin or Greek, e.g., "electroencephalographic". These words only blow up the vocabulary without providing much predictive power, to any downstream ML tasks. Therefore, they should be filtered out.

## 3.2 Word Embeddings

**Word2Vec** Word2Vec builds upon the idea that words in a similar context also have a similar meaning. At the beginning of the algorithm, the vectors for each word are randomly initialized. Then, the algorithm starts to pass through the corpus with a center word and a context window around that center word. The dot product of the center word with the context words is calculated, and the result is minimized using Stochastic Gradient Descent (SGD). Whenever two words co-occur in a window together their distance to each other is shortened, the more often this happens, the closer the two words get. Negative sub-sampling was added to address the problem that an extensive corpus would just put all the words together because they might all co-occur. The sub-sampling maximizes the distance of randomly sampled words, not in the context window. This process ensures that not all the words end up in the same place in the vector space. The sub-sampling is based on the word frequency such that more frequent words also get more frequently sub-sampled in this way the algorithm addresses the problems of stop words.

When the training is complete, words with similar meanings are mapped onto a similar vector space, for example, "banana" and "apple" should be closer together then "banana" and "philosophy."

Following both of its variants are described:

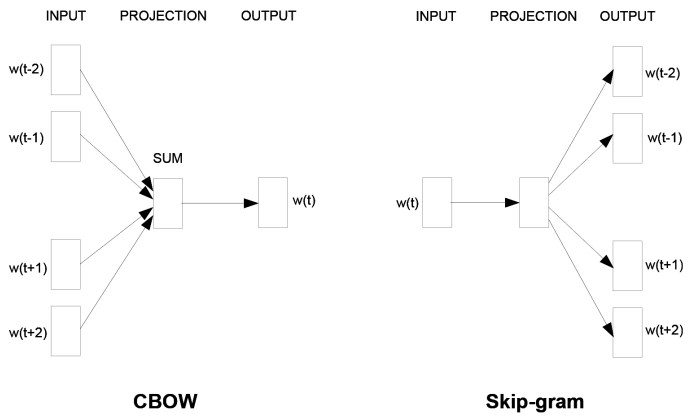



Figure 3.1: Explanatory image of Skip-Gram and CBOW [Mikolov et al., 2013a]

**Skip-Gram (SGNS)** In the SGNS algorithm, word embeddings are created by looking at one word and maximizing the chance of predicting the surrounding words. If we have the sentence "A fox jumps over the lazy dog" and a window size of seven, if "over" is used as input, SGNS should predict: "a", "fox", "jumps", "the", "lazy", "dog".

**Continous Bag of Words (CBOW)** CBOW follows a similar approach to SGNS but the other way round. Looking at some words before and some after a center word, this center word tries to be correctly predicted. The context word vectors are aggregated through the sum of these vectors in order to get only one vector.

**Global Vector (GloVe)** Unlike Word2Vec, which focus on local properties of sentences and words, GloVe uses a global count statistic in the form of a co-occurrence matrix where words of a certain window size occurring together are counted. In the GloVe paper, it is argued that meaning is highly related with the co-occurrence of words. Taking their example "ice" and "steam" are two different states of water. It is expected that both words often co-occur with water. In contrast to the word "cold," it is not expected to be seen together with the word "steam" often. GloVe formalizes this co-occurrence of words, where it tries to optimize the dot product between a word vector and its content vectors to be as close as the co-occurrence probability as possible.

$$w_i^T w_j^{'} + b_i + b_j = \log X_{ij} \tag{3.1}$$

where $X$ is the co-occurrence matrix and $X_{ij}$ the co-occurrence of $w_i$ and $w_j$ in the specified window. $b_i$ and $b_j$ are scalar bias terms associated with the corresponding word.

$$J = \sum_{i=1} \sum_{j=1} f(X_{ij})(w_i^T w_j^{'} + b_i + b_j - \log X_{ij})^2 \tag{3.2}$$

Objective function $J$ tries to minimize the sum of squared errors of the previous function. [Pennington et al., 2014b]

Interestingly, similar to Word2Vec also Glove has a way to deal with frequent words. The input for this function is $X_{ij}$, which represents the number of times when $i$ is in the context of $j$.

$$f(x) = \left\{ \begin{array}{ll} \frac{x}{x_{max}}^{\alpha} & \text{if } x_{max} \\ 1 & \text{otherwise} \end{array} \right\} \tag{3.3}$$

The results were achieved when $\alpha = \frac{3}{4}$ and $x_{max} = 100$.

## 3.3 Document Embeddings

A document embedding gives a document a representation in a latent space. The goal of document embeddings mainly is to work as input for various downstream machine learning tasks like sentiment analysis or information retrieval.

**Doc2Vec** Similar to Word2Vec this approach comes in two flavors. The Doc2Vec version of the Word2Vec CBOW is called Paragraph Vector Distributed Memory (PV-DM), and the Doc2Vec version of the SGNS is called Paragraph Vector Distributed Bag of Words (PV-DOW).

**PV-DM** PV-DM tries to predict the focus word given document and context. As mentioned in [Le and Mikolov, 2014], PV-DM should perform consistently better the following approach PV-DOM.

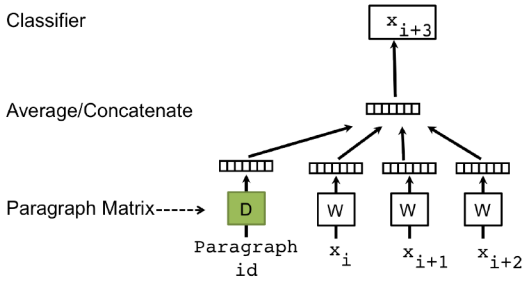



Figure 3.2: PV-DM [Dai et al., 2015]

**PV-DOM** This algorithm predicts the probability of the context with the given document. It is similar to SGNS from Word2Vec, but instead of focusing on the focus word the document is conditioned.

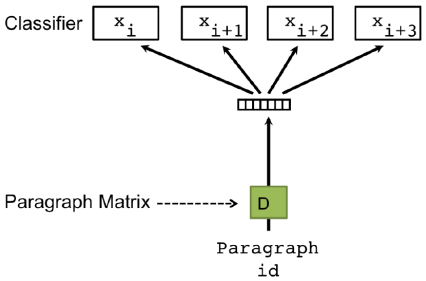



Figure 3.3: PV-DOM [Dai et al., 2015]

**Word Embedding Centroids** Document representations can also be derived from a function of all the word vectors contained in a document. In this approach, word vectors pre-trained or learned can be summed up, averaged, the minimum or the maximum for each dimension in the vector space can be taken. The term word embedding centroid was coined by [Brokos et al., 2016]. But this method is often used as a baseline model to construct document embeddings from word embeddings [Weston et al., 2014, Dai et al., 2015, De Boom et al., 2016].

[De Boom et al., 2016] suggested to take the minimum, maximum, sum or average of as aggregation function to retrieve basic document embeddings. They slso suggested approaches that weight word vectors with TF-IDF.

The methods to construct the document vectors are illustrated below. The examples are illustrated with $\vec{v}$ as the resulting document vector and $\vec{a}$, $\vec{b}$ and $\vec{c}$ as the word vectors. The example is of dimension 3, and the text contains three words.

**SUM**

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{pmatrix} a_1 + b_1 + c_1 \\ a_2 + b_2 + c_2 \\ a_3 + b_3 + c_3 \end{pmatrix} \tag{3.4}$$

**AVG**

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{pmatrix} \frac{(a_1+b_1+c_1)}{N} \\ \frac{(a_2+b_2+c_2))}{N} \\ \frac{(a_3+b_3+c_3))}{N} \end{pmatrix} \tag{3.5}$$

**MIN**

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{pmatrix} \min(a_1, b_1, c_1) \\ \min(a_2, b_2, c_2) \\ \min(a_3, b_3, c_3) \end{pmatrix} \tag{3.6}$$

**MAX**

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{pmatrix} \max(a_1, b_1, c_1) \\ \max(a_2, b_2, c_2) \\ \max(a_3, b_3, c_3) \end{pmatrix} \tag{3.7}$$

## 3.4 Bag of Words (BOW)

BOW is a vectorization process for text. This process gives the text a sparse representation in a vector space. The vector space has the size of the unique words contained in a corpus. Words are counted in a document where each word is represented as a one-hot vector, with a 1 for the current word and the rest of the vector is zeros. The multiplication product of the word vectors with their occurrence is then summed up, resulting in the final BOW for the document, a single vector. Similar to document embeddings, this model does not account for the word order. To compare several documents, large sparse vectors are needed, which makes it difficult to be used for ML tasks. Because fixed input length is a preferable feature for various ML algorithms. An example here are neural networks where the input size might be necessary for subsequent hidden layers.

**TF-IDF** is the abbreviation of term frequency-inverse document frequency. The frequency distribution of words in a natural language text follow a Zipf or Power Law distribution and consequently a small number of words occur often while most other words are located somewhere in the long tail of the distribution. An untreated BOW model weights common words higher than rarer words. Unfortunately, frequent words do often not contribute in elaborating the specific context of a sentence, and therefore they might act as noise. TF-IDF accounts for this issue and introduces the document frequency. This helps to weight words higher that only occur frequently in a specific document while generally frequent words are weighted lower or even canceled out if they occur in every document.

**Term Frequency** Term frequency is the count of each unique term of a document. One problem with the term frequency is that it is highly correlated with the length of the document. Longer documents also have higher document frequencies then shorter ones. A simple mitigation of this problem is the weighting of the term count that all term counts sum up to one.

Let $tf_{t,d}$ be the frequency of a word $w_t$ in document $d_d$ and $df_t$ be the document frequency of word $w_t$.

**Inverse Document Frequency** While term frequency is calculated for a specific document, inverse document frequency is calculated among all documents. It cancels words that occur in all documents out and decreases the weight of frequent wordsr.

The inverse document frequency is defined as $idf_t = \log 2\frac{N}{df_t}$, where $N$ is the total number of documents.

Finally, TF-IDF is defined as:

$$tf\text{-}idf_{t,d} = (1 + \log 2 tf_{t,d}) \cdot \log 2 \frac{N}{df_t}$$

Other modes of TF-IDF normalize either $tf$ or $idf$ to diminish the influence of the document size.

**Used Weighting Schemes** Following the four different weighting schemes for the word representation in a BOW applied in the experiments :

**TF-IDF** In this mode, the numbers in the BOW document vector are weighted according to the TF-IDF.

**Count** In this mode, the raw count per words is used in the vector for each text.

**Freq** In this mode the all the numbers for each word occurring in a document sum up to 1.

**Binary** In this mode, if the word is in the text, a one is in the vector otherwise a zero. This mode is especially helpful when used together with ML tasks since this can be used as a multi-label classification problem, where each word in a text is an additional label.

## 3.5 Neural Auto-encoder

In addition known baseline methods, a method was created to get document embeddings. The new process used a simple deep neural network auto-encoder setup. Auto-encoder have perfect features for this task, because auto-encoding is an unsupervised machine-learning method, where input and the same output can be used to train the encoder decoder architecture. The goal is that the encoder learns how to encode some data to compress it to use less space and then the decoder determines the step to reverse this compression again to get the original output. The result can be compared with the initial output, and both the encoder and the decoder can be trained to achieve the best possible outcome. This is precisely what most embedding methods do in a different more word and co-occurrence centric fashion.

To be able to feed text into the auto-encoder the binary vector encoding from BOW was used. Here, words were used as a multi-class problem where the encoder encoded the sequence until the sequence consists only of 300 numbers and then the decoder tried to predict each word that initially was part of the text from the compressed numeric vector. After this auto-encoder was trained sufficiently, the encoder is used to encode new documents. The encoding can then be used for any ML tasks similar to learned Doc2Vec embeddings.

# 4

# Experimental Setup

In this chapter all the background for the experiments is shown, starting with the presentation of the different datasets that were used for benchmarking reasons. Then all details about the implementation of the code for this thesis are explained, concerning software but also hardware. In the Methods section, the final methods and their hyperparamters used for the Experiments are listed. Then the specific evaluation method with the neural classifier is shown. Finally in the Methodology section all experiments are listed, including their contribution to answer the hypotheses.

Table 4.1: The different statistics for each dataset.

| | **CR** | **IMBD** | **MPQA** | **MR** | **NG20** | **SST1** | **SST2** | **SUBJ** | **TREC** |
|---|---|---|---|---|---|---|---|---|---|
| Number of documents | 3774 | 50000 | 10606 | 10662 | 18305 | 160128 | 79654 | 10000 | 5952 |
| Mean number of words per document | 20 | 265 | 3 | 21 | 174 | 7 | 9 | 24 | 10 |
| Number of classes | 2 | 2 | 2 | 2 | 20 | 5 | 2 | 2 | 6 |
| Number of Words | 5341 | 113321 | 6234 | 18758 | 77610 | 17828 | 17232 | 21322 | 8759 |
| Words in Pretrained embedding | 5296 | 89147 | 6199 | 18001 | 64499 | 17618 | 17038 | 20463 | 8675 |
| Word in pretrained embedding ratio | 99.16 | 78.67 | 99.44 | 95.96 | 83.11 | 98.82 | 98.87 | 95.97 | 99.04 |

## 4.1 Datasets

The evaluation was carried out on a series of datasets: Customer Reviews (CR), Movie database (IMDB), Multi-Perspective Question Answering (MPQA), News Group (NG20), two versions of the Stanford Sentiment Treebank (SST1 & SST2) a Subjectivity dataset (SUBJ) and a question classification dataset (TREC). The datasets were selected such that they cover a broad range of document sizes, number of classes, corpus size, and vocabulary size. The detailed statistics of each dataset can be found in Table 4.1. A table with the references and a short explanation for each dataset can be found in Table 4.2. A Table containing an example of each dataset and a corresponding class is listed in Table 4.3.

Table 4.1 shows that the document length is less than 25 for most of the datasets. IMDB and NG20 are the only two datasets that contain significantly longer documents.

This characteristic is highlighted in Figure 4.1, which plots the histogram of document lengths. These two datasets show a bigger variance in length while all the other datasets have one high peak around the mean document length of the dataset.

Basic preprocessing was conducted on each of the datasets. It included the lowercasing of all words, splitting most of the punctuation signs from the words, splitting short forms of negations, verbs or genitive "'s" from the word they were attached to. Then the text was tokenized.

**Frequency distribution of the Datasets** Figure 4.2 shows the word frequency, word rank plot in log-log scale. Text follows a Zipf, Power Law, or Pareto distribution. These laws or distributions describe an imbalanced distribution. The pareto principle says that 80% of the wealth is owned by 20% of the people. These laws may be translated into texts, that only a few words occur very often while other words tend to be rarer. A theory that tries to explain the Zipf distribution is called "principle of least effort", which states that animals, machines, and humans will tend to choose the path of least resistance, in texts this results in the mentioned distribution. The Zipf distribution is not only a feature of English but any language. The plotted distributions show this with the most occurring words often are "a", ".", "and", "the", "of" and many more. A perfect Zipf distribution follows a straight sinking line.

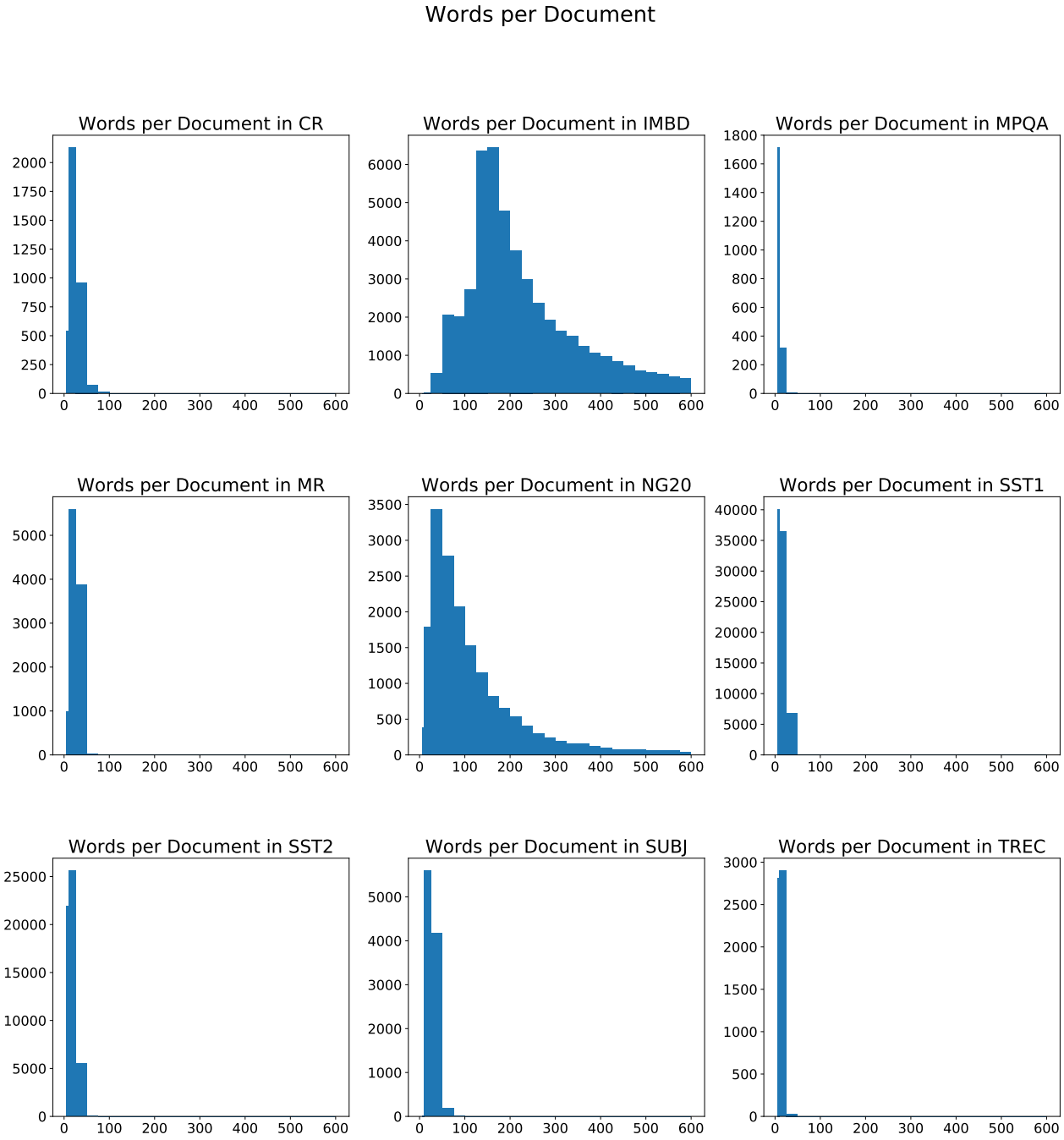



Figure 4.1: The distribution of documents length per dataset. The bins of the histogram are the same while the y axis is different for each plot.

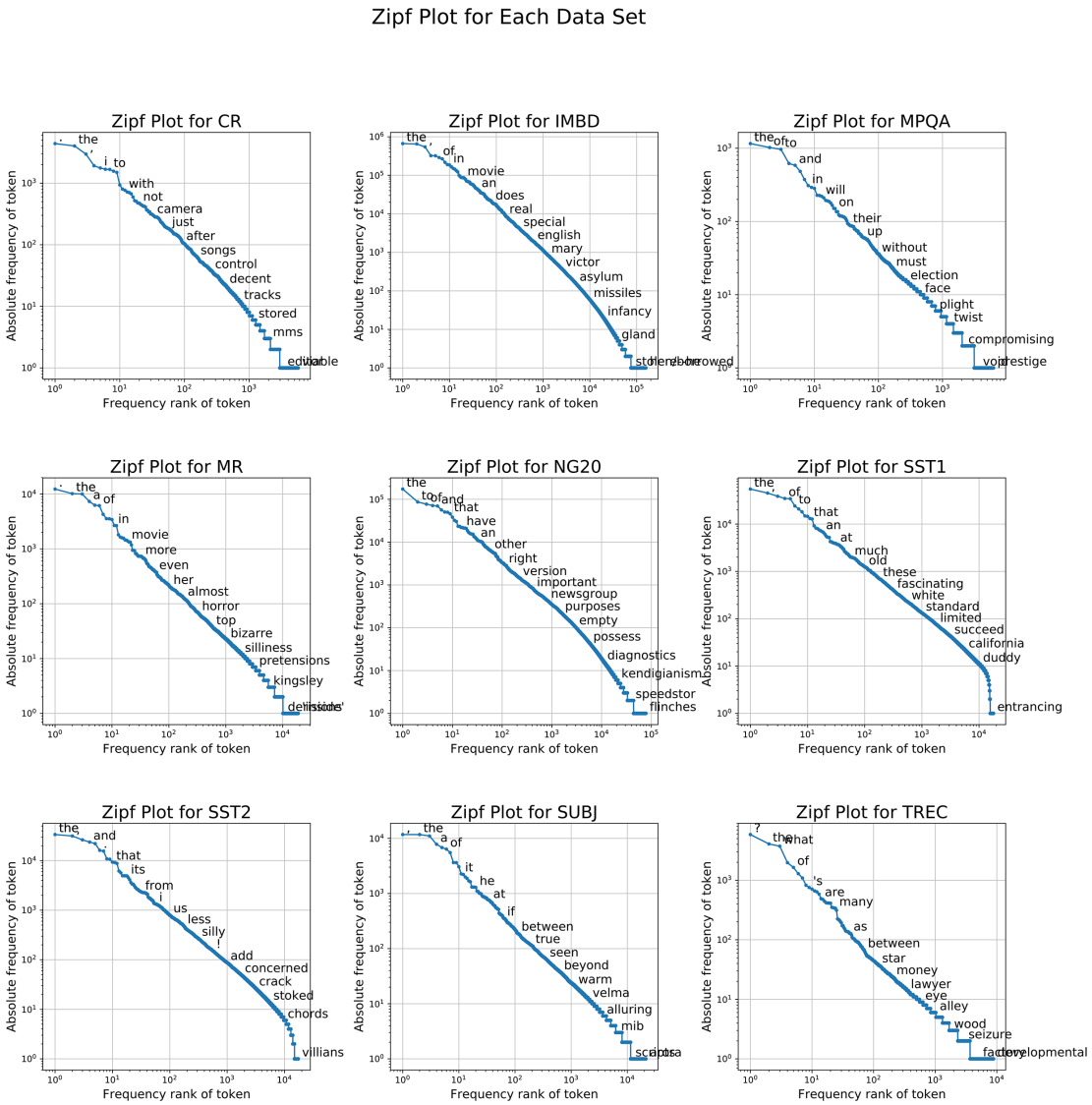



Figure 4.2: This plot shows the words ordered by their frequency as a Zipf plot.

Table 4.2: Datasets

| Dataset | Description |
|---|---|
| CR | Annotated customer reviews of 14 product categories from Amazon (cameras, MP3s, etc.). The task is to predict positive/negative reviews [Hu and Liu, 2004]. |
| IMDB | Dataset used in the paragraph vector paper [Le and Mikolov, 2014]. Consists of 100'000 movie reviews with 25'000 positive, 25'000 negative and 50'000 not labeled documents. The dataset was gathered from the IMBD movie review platform. |
| Movie Review - RM | Movie reviews with one sentence per review. The classification task involves detecting positive/negative reviews (Pang and Lee, 2005). [Pang and Lee, 2005] |
| MPQA | The MPQA short for Multi-Perspective Question Answering Subjectivity Lexicon was created by [Wiebe et al., 2005]. This classification task for this dataset is the opinion polarity detection on the phrase level. |
| News Group - NG20 | The 20 Newsgroups dataset is a collection of approximately 20'000 newsgroup documents, distributed evenly across all the 20 different newsgroups. The dataset was originally collected by [Lang, 1995]. Because of many artifacts and fragments in the text punctuation and numbers were removed. |
| SST1 | Stanford Sentiment Treebank is an extension of MR but with train/dev/test splits provided and finer grained labels (very positive, positive, neutral, negative, very negative) [Socher et al., ]. |
| SST2 | This dataset is a revamped version of the SST1 where the dataset was re-labeled and binned into only positive and negative where neutral reviews were removed [Socher et al., ]. |
| SUBJ | Subjectivity dataset where the task is to classify a snipped as being subjective or objective [Pang and Lee, 2004]. |
| TREC | The TREC dataset contains questions. The task with this dataset involves classifying a question into six question types, whether the question is about a person, location, numeric information, etc. [Li and Roth, 2002]. |

Table 4.3: Examples of text and class for each dataset.

| Dataset | Document | Class |
|---|---|---|
| CR | weaknesses are minor : the feel and layout of the remote control are only so-so ; . it does n 't show the complete file names of mp3s with really long names ; . you must cycle through every zoom setting ( 2x , 3x , 4x , 1/2x , etc . ) before getting back to normal size [ sorry if i 'm just ignorant of a way to get back to 1x quickly ] . | 0 |
| IMDB | bromwell high is a cartoon comedy . it ran at the same time as some other programs about school life , such as " teachers " . my 35 years in the teaching profession lead me to believe that bromwell high 's satire is much closer to reality than is " teachers " . the scramble to survive financially , the insightful students who can see right through their pathetic teachers' pomp , the pettiness of the whole situation , all remind me of the schools i knew and their students . when i saw the episode in which a student repeatedly tried to burn down the school , i immediately recalled . . . . . at . . . . . high . a classic line : inspector : i'm here to sack one of your teachers . student : welcome to bromwell high . i expect that many adults of my age think that bromwell high is far fetched . what a pity that it is n't ! | 1 |
| MR | simplistic , silly and tedious . | 0 |
| MPQA | complaining | 0 |
| NG20 | i am sure some bashers of pens fans are pretty confused about the lack of any kind of posts about the recent pens massacre of the devils actually i am bit puzzled too and a bit relieved however i am going to put an end to relief with a bit of praise for the pens man they are killing those devils worse than i thought jagr just showed you why he is much better than his regular season stats he is also a lot of fun to watch in the playoffs bowman should let jagr have a lot of fun in the next couple of games since the pens are going to beat the pulp out of jersey anyway i was very disappointed not to see the islanders lose the final regular season game pens rule | 10 |
| SST1 | no movement , no yuks , not much of anything . | 1 |
| SST2 | no movement , no yuks , not much of anything . | 0 |
| SUBJ | smart and alert , thirteen conversations about one thing is a small gem . | 0 |
| TREC | how far is it from denver to aspen ? | 5 |

## 4.2 Implementation

The programs were written in Python, using [Anaconda, 2016] as a virtual environment for package management across different servers. The code was written in Jupyter notebooks by [Kluyver et al., 2016]. Essential packages in Python are Keras by [Chollet et al., 2015] for neural network learning, Gensim by [Řehůřek and Sojka, 2010] for Doc2vec, Word2vec and other NLP related tasks together with NLTK by [Loper and Bird, 2002] and Scikit-learn by [Pedregosa et al., 2011] for cross-validation, training-test-splits or grid searches.

The code was deployed on a cluster with 12 nodes, each node containing 12 CPU cores (2x AMD Opteron 6174 2.2 GHz) and 24 logical cores. Jobs were scheduled with Slurm[Jette et al., 2002] a Linux workload manager. In order to run a task on Slurm, two different scripts were needed. First, a bash script containing Slum specific parameter like on which node the job should be executed or how much resources would be used, and the command to execute the corresponding python script. In the Python script, the actual program that runs experiment is located. All tasks were processed using 24 threads with nothing else calculated on the node. Each test was done in one pass through all the datasets with three or ten-fold cross-validation depending on the task for each dataset. The results were saved into a CSV file for each dataset and each method and possible variations like SUM, TF-IDF etc.

For some tasks that did not rely on any parts contained in the Gensim module but relied more heavily on Keras, a GPU cluster was used with eight GeForce GTX TITAN X with 12207 MB GPUs. However, this was only used if the wall time was not measured. In order to run the scripts on the GPU, the same virtual environment was used, and some unique addition in the code forcing the usage of GPUs had to be added.

After all the data was collected and saved into CSV files, Jupyter notebooks that retrieve the data and use them to generate plots were implemented. In this step, most of the diagrams were constructed using the python package Matplotlib by [Hunter, 2007].

## 4.3 Methods

In this section all the methods evaluated are described and the corresponding hyperparameters are listed.

**Bag of Words (BOW)** In this method, BOW representations are built using all the words in the training set to construct a bow representation of all the training and test texts. The text is preprocessed such that every word is lowercased and to reduce the vocabulary size, the minimum occurrence is set to 2. In case a word is not present in the text a zero is written in the vector. The used variations of BOW were count, frequency, binary and TF-IDF wheighted.

**Pre-trained GloVe Vectors** For this method, a large pre-trained GloVe corpus with word vectors of dimension 300 is loaded, trained on 42 billion tokens and 1.9 Million

words in the vocablary [Pennington et al., 2014a]. A similarly strong Word2Vec pre-trained corpus could also have been used. The word vectors for each word in a text have to be grouped in order to create one document vector of dimension 300. The methods used to aggregate the word vectors to a document vector are sum, minimum, maximum and average.

**Learned Word2Vec Vectors** For this approach, word vectors are learned on the training set documents. These word vectors are then further used to construct a simple document vector same as the example above, eg. sum, minimum, maximum and average.

**Learned Doc2Vec Vectors** In this method, the algorithm of paragraph vectors [Le and Mikolov, 2014] are learned and later used for the classification task. Also here the document vectors are only learned for the training set and the test set is inferred from the model with test texts as input.

## 4.4 Model parameters

To maximize the accuracy of the two hyper-parameter heavy methods Doc2Vec and Word2Vec, these hyperparameters were learned with a time-limited grid search. The grid search was applied only on one dataset and then used for all others. This procedure is not advisable in a real-world application, but it ensures that the whole algorithm stays comparable through all datasets otherwise every change of a parameter results in a different acting model.

The grid search was applied using the Scikit-learn package. For the classifier, a simple logistic regression was used knowing that a more complicated neural network classifier could do an even better job. Using a simpler classifier was crucial because the neural network classifier learning could take rather long. The grid search was evaluated in three splits, where the hyperparameters with the highest mean would be candidates to become the new hyperparameters of the method.

**Hyperparameters of Word2Vec** Below a list of all hyperparameters important for the Word2Vec implementation in Gensim.

- Vector size 300
- Window size 16
- negative subsample 5
- minimum occurrence 2
- Iterations 50
- Sampling threshold 1e-5,
- sg = 1: the skip-gram model is applied.
- hs = 1: hierarchical soft max is used.

**Hyperparameters of Doc2Vec** Below a list of all hyperparameters important for the Doc2Vec implementation in Gensim.

- Vector size 300
- Window size 15
- Minimum occurence 2
- Negative subsample 5
- Iterations 50
- Sampling threshold 1e-5,
- Minimum alpha 0.00025,
- initial alpha 0.025
- dm = 0: PV-DBOW mode is used.

**Hyperparameters of Simple Neural Auto-Encoder** Best results were achieved with the input of the IMBD dataset BOW containing a vocabulary of 60'000 words, which was then put through a net of 60'000 in the input layer, 1200 nodes in the first hidden layer, 600 in the second hidden layer and 300 in the last hidden layer of the encoder. Then the decoder attempts to blow up this sequence again with 600 and 1200 intermediate layer until the final layer of the output is reached again. Finally, the input and output are compared, and the weights are adjusted to reduce the cost function, this happens through back-propagation and gradient-descent.

The steps of 60'000, 1'200, 600, 300, 600, 1'200, 60'000 had from all the tested variations the best properties for the IMBD dataset. For smaller datasets with fewer words, this was adjusted. The function should mimic the compression from the IMBD dataset to other smaller datasets with the minimum of 300.

## 4.5 Evaluation of the Vector Models

The evaluation method was chosen to be the same for all different tested methods. It consists of a neural classifier used with an input layer corresponding to the size of the input, a dense hidden layer with 50 neurons and a final classification layer with as many neurons as classes that need to be classified. This specific setup was proposed in [Le and Mikolov, 2014].

**Binary Class Case** In the binary class case, a sigmoid activation function is used together with the binary cross entropy loss function and the Adam optimizer.

**Multi Class Case** In the multi-class case, the activation function is softmax with categorical cross entropy loss and the same Adam optimizer the same as in the two-class case above.

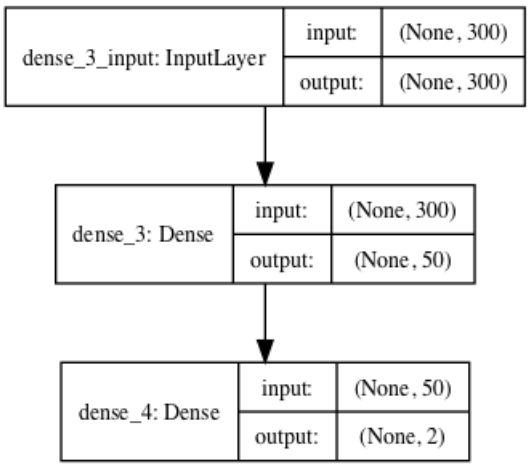



Figure 4.3: Exemplary setup of the network, classifying two classes and having a document vector of size 300 as input.

## 4.6 Methodology

The description of the individual experiments is listed in this section. These experiments were split into three main fields, accuracy, transferability, and speed. In the accuracy part, the accuracy of the different methods was compared. In the transferability part, the transferability mainly of the Doc2Vec method was looked at. Transferability means here that models learned on one corpus could be reapplied to other corpora. Finally, in the last part about speed, the different methods were compared against one another according to their speed, including learning times or document creation.

### 4.6.1 Preprocessing effects

**Accuracy Depends on Preprocessing** Preprocessing the text can be essential and is advisable since one can argue that words occurring only once do not help to build prediction rules to differentiate documents from each other since this rule is not generalizable. Another reason for words rare words could be, especially in customer reviews, that the words are orthographically spelled wrong. Therefore they should either be replaced, corrected or stemmed. The last preprocessing step could be the removal of stop words. In some approaches, this problem can be diminished with the application of TF-IDF weighting of words such that frequent words are weighted less than others. Some approaches already have useful instruments that address these problems, such as the negative sub-sampling of Word2Vec or the special weighting function for frequent co-occuring words in GloVe.

**Hypothesis:** *Pre-processing is important for higher accuracy since punctuation and stop-words do not help on the classification problem.*

Four datasets were selected to check the above-stated question. One dataset with

short average document length TREC, two "medium" datasets CR and SUBJ, one larger dataset here the IMBD was used for this experiment. This base case was only preprocessed such as lower-casing, punctuation surrounding with spaces, verb short forms, negations were split from their verb part, and a word should occur at least twice. On top of that new data sets were created. These datasets were stemmed, stop words removed, punctuation removed and both stop word and punctuation removed and then tested against their base case.

Stopword and punctuation removal were combined because both remove unnecessary parts of the text making it shorter if one might be right the combination of both might be even better. Stemming was not mixed with the other two since this is a different form of normalizing changing not the text but the words in it.

### 4.6.2 Accuracy

**Accuracy Comparison Method and Datasets** To assess the hypothesis below, the accuracy of the evaluation classifier was measured in a 10-fold cross-validation settings. The classifier was trained for 5 epochs on each of the datasets.

> **Hypothesis:**
> *Document embeddings outperform baseline methods in a classification task setup.*

Might Doc2Vec be too complicated and BOW be an excellent approach to use in classification tasks? Possible influencing factors like number of words, document size, number of classes of the datasets should be examined across all the methods. Are there benefits in some datasets that are idiosyncratic for the dataset or might it be a standard feature of these kinds of datasets?

Since the IMBD dataset is evaluated, the best results should be comparable with the ones stated in the paper of [Le and Mikolov, 2014] even though the vector dimension, and the learning test split is different.

> **Hypothesis:**
> *Document embeddings results from the paper can be reproduced.*

**Accuracy Depends on Document Length**

> **Hypothesis:**
> *Document embeddings depend on the length of documents for higher accuracy.*

To answer the question above multiple datasets were created. The datasets contain nine books from the NLTK corpus. The text of these books was normalized, cleaned and then sub-sampled in text passages of length 50, 100, 200 and 400 allowing overlapping or text passages to get enough samples and ignoring sentences borders. These created datasets will enable the prediction of the book and author.

The creation of a new dataset was necessary because datasets with documents of different length vary in their textual features and sources. A flaw of the dataset is the

overlapp in textual passages, which is getting more significant the more extended the text passages to get. This problem is mitigated when the documents are rigorously split in non-overlapping test and train sets (although then the measurement would rely on the split properties). This problem was addressed with the modification of the dataset, once with strict train, test split without cross-validation and once with 10-fold-cross-validation without strict train,test split applied.

The analysis of the length of the document and its implication for the accuracy was only applied to the two most important methods Doc2Vec and the variations of BOW to test whether or not other methods profit from larger documents as a possible comparison.

**Accuracy Depends on the Corpus Size** In all methods, rigorous test train splits were applied to the learning of the Doc2Vec and Word2Vec models. This approach is probably against the nature of these models, especially Word2Vec where good results can be derived from vast amount of texts. Therefore, a test is done on this topic, to check whether or not more documents to train the embedding creation model would result in higher accuracy.

> **Test:** Train Doc2Vec on all documents. Use 90% to train classifier, use the rest for 10% evaluation.
> **Base case:** Train Doc2Vec on 90% of the documents. Use 90% to train classifier, use the rest for 10% evaluation.

**Accuracy of Simple Auto-encoder** The method described in Section 3.5 is applied on all the datasets and compared with the other methods. The encoder is trained on the whole corpus such that each train test split in the cross-validation process could use the same decoder to get the compressed BOW vector. Similar to the accuracy test the results were evaluated after 5 epochs of the classifier in order to be comparable.

### 4.6.3 Transferability

Transferability is a nice feature in ML, because it can speed up other processes, further down the ML pipeline. This concept is called transfer learning. An example of transfer learning is the use of pre-trained word vectors, which can be used as an input for other approaches like a CNN. These CNNs can then build upon the already learned concepts from other models. But is this also applicable to methods like Doc2Vec? Could it be that a pre-trained Doc2Vec model turns out to be useful to infer documents from there, instead of learning them directly on the corpus not knowing which are the perfect hyperparameters for this specific application

In these experiments, the transferability of externally trained Doc2Vec models is evaluated. That word vectors learned on huge corpora are used in myriad different ways is already common data science practice. These pre-trained vectors are often used as a baseline method, or as input for other methods [Chen et al., 2016].

Can Doc2Vec be learned on larger corpora and then be reused for smaller ones? The general benefit would be that the training time could be reduced and replaced with

just the loading time of the right model. Pre-trained word embeddings learned on a vast amount of data also bear the benefit of proven generality on a wide range of texts and words. Another good reason for transfer learning is that the generation of such an embedding corpus, such as the one used for the pre-trained vector centroids [Pennington et al., 2014a] might not be computable for most machines available.

But the inference of the document embeddings is only one part. It might well be that the created embeddings are so general that even the classifier trained on one task can be reused for other similar situations.

In the following, all transferability are described:

**Doc2vec: Transferability of the Model**

> **Hypothesis:**
> *Document embeddings save effort since precomputed embeddings can be used in many applications.*

In this experiment, the idea is tested of the creation of an excellent Doc2Vec model, trained on another dataset than the actual classification task. For this experiment, Doc2Vec models were learned on each dataset. These models were then used to derive document embeddings from all other datasets. These generated document embeddings learned on an external dataset, were then classified and the accuracy was compared with the accuracy that was initially measured when the Doc2Vec model was learned on the same dataset as it would also be used.

The intuition would be that bigger dataset, with richer documents and a with a similar domain, should be able to create embeddings that "work", while datasets in a different field and smaller as the target dataset should result in bad performance.

> **Test:** Learn doc2vec on dataset A infer documents of dataset B, use the inferred embeddings from B.
> **Base case:** Learn doc2vec on B train a classifier. (Result from the Accuracy section)

**Doc2vec: Transferability of the Model and the Classifier**

> **Hypothesis:**
> *Document embeddings save effort since the same classifier can be used in many applications.*

In this case, not only one model as in the case above is reused, but both models are kept. Here the embedding creation model and the binary classifier is reused in all applicable cases. All possible datasets that predict a binary label are here CR, IMBD, MPQA, MR, SST, and SUBJ. But only CR, IMBD, MR and SST are sentiment analyis tasks. The result of MPQA and SUBJ are thought to be low in any case since the task is not to predict a sentiment but subjectivity or objectivity. What could be expected is that the IMBD dataset and its classifier cover enough material such that the application on the MR dataset which has a similar domain should result in a satisfactory result.

**Test:** Learn Doc2Vec on dataset A, learn classifier on the embeddings of A, infer document embeddings learned on A on B use the classifier trained on the embeddings from A.
**Base case:** Learn doc2vec on B, train a classifier. (Result from the Accuracy section)

**BOW Transferability of the Classifier** Similar to the experiment in 4.6.3, this is also attempted for the BOW models and its variations The expected result would be a similar or worse performance and the reasons for better results are also similar to the experiment with Doc2Vec, where models trained on bigger corpa should also be able to cover more cases.

**Test:** Create a BOW vectorization for A, learn the classifier for A, create vectorization for B using the same vocabulary as for A. Use classifier of A for B.
**Base case:** Create an embedding for B learns the classifier for B.

### 4.6.4 Speed

Speed is often a valuable resource especially in the times of fast Internet and very capable mobile phones. Therefore, speed cannot be neglected in this evaluation. The speed of the different methods is looked at, precisely how long it took to set up, learn and evaluate the models for each method and dataset methods. Light is shed on the classifier and its performance along the time axis. Hence, for each dataset and each method, models were created in a three-fold cross-validation process, where the classifier was evaluated after each epoch, reporting time and accuracy, in a total of 50 epochs. Three questions were assessed two further answers the hypothesis.

**Hypothesis:**
*Document embeddings save time since they reduce the complexity of the documents to reduce the work for the classifier.*

**How fast does the classifier converge?** For each dataset and each method, this is checked. Are there differences in the methods and do some methods take longer for the classifier to grasp their potential? Another important question is when is a maximum reached. While in the previous Accuracy section the epochs were set to 5 as a boundary rul,e now the behavior of the classifier can be further investigated.

**What is the final accuracy?** In the previously stated evaluation of the classifier after each epoch for 50 epochs, the average of the three-folds was used to get a mean accuracy of each epoch step. The maximum of these accuracies was taken to create a ranking to give better advice on which of the methods performs best.

**How long does it take?** For each dataset and each method, the duration of loading exterior datasets like pre-trained embeddings, training of models, construction of documents and the learning of the classifier is checked and compared against each other, to finally create a ranking of the methods on all datasets.

# 5

# Results

In this chapter, all the results of the previously stated experiment chapter are listed and discussed. This includes the three main parts accuracy, transferability, and speed.

## 5.1 Preprocessing effects

The results of preprocessing on five different levels with four different datasets are compared among two methods, namely Dov2Vec and the BOW variations. In the first part, the influence of the preprocessing on the accuracy is examined, and in the second part, the effect on the speed is discussed. The transformation of the dataset in its preprocessed versions was left out because of the vast speed of this step even on commercially and publicly available computer hardware.

### 5.1.1 Preprocessing Accuracy

Table 5.3 and 5.2 together with Figure 5.1 and 5.2 show the mean accuracy of the baseline together with four different preprocessing steps, no punctuation, no stop words, no punctuation and no stop words and stemming for the two methods BOW and Doc2Vec.

Table 5.1: The results of Doc2Vec of the four datasets CR, IMBD, SUBJ and TREC preprocessed differently and compared with the baseline the basic preprocessing. In **bold** the highest accuracy per line in *italic* the lowest.

| **d2v** | Only basic preprocessing | No punctuation | No stopwords | No punctuation & No stopwords | stemmed |
|---|---|---|---|---|---|
| CR | 74.735 | 74.193 | 73.393 | *73.251* | **75.040** |
| IMBD | 83.476 | 83.476 | 83.440 | *83.208* | **83.657** |
| SUBJ | 88.550 | **88.835** | 86.770 | *86.635* | 87.515 |
| TREC | 58.148 | **58.752** | 51.961 | *50.605* | 58.553 |

The results show that accuracy-wise, preprocessing does not provide substantial benefits. The results offer that stemming might lead to slight accuracy gains. Small accuracy gains can also be retrieved through the removal of punctuation, but stop word removal can lead to a slight accuracy loss, especially in datasets with short mean document lengths like TREC. This effect is also visible in the two smaller datasets CR and SUBJ, but it is lessened in datasets with longer documents like IMBD.

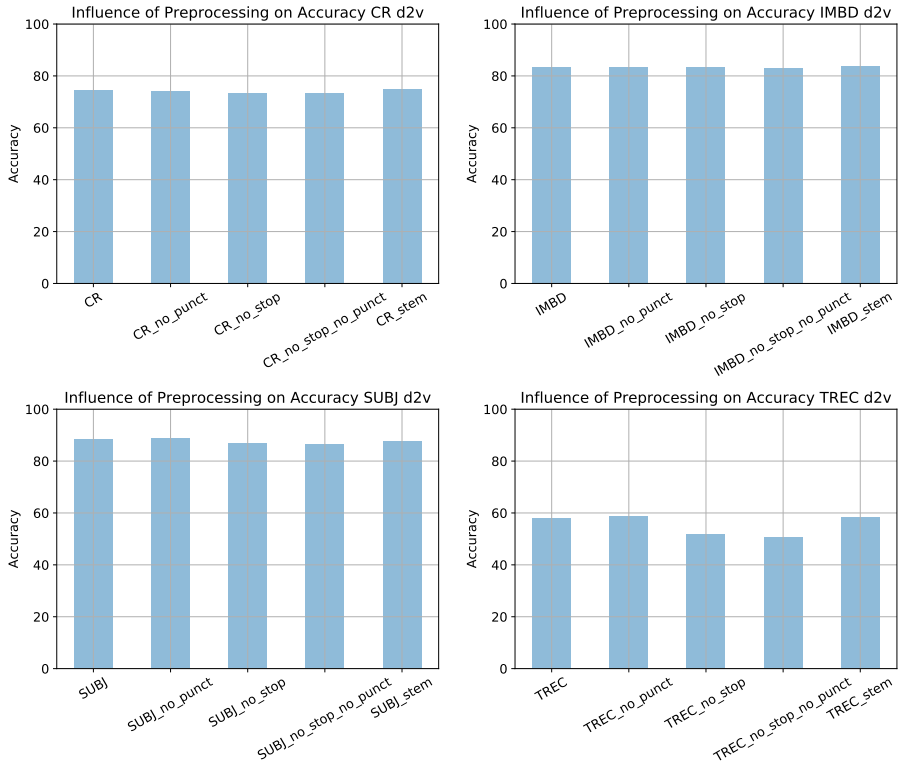



Figure 5.1: The accuracy of the Doc2Vec method on the four different datasets and four different preprocessing methods.

Table 5.2: The results of the many BOW variations of the four datasets CR, IMBD, SUBJ and TREC preprocessed differently and compared with the baseline here the basic preprocessing. In **bold** the highest accuracy per line in *italic* the lowest.

| **bow** | Only basic preprocessing | No punctuation | No stopwords | No punctuation & No stopwords | Stemmed |
|---|---|---|---|---|---|
| CR_binary | 79.531 | 79.757 | 78.352 | *78.299* | **80.458** |
| CR_count | 80.367 | 80.248 | 78.139 | *78.113* | **80.497** |
| CR_tfidf | 79.585 | 79.505 | *77.609* | 77.848 | **80.206** |
| CR_freq | 78.750 | 78.458 | *77.980* | 78.073 | **79.438** |
| IMBD_binary | 88.309 | **88.485** | 88.243 | *88.164* | 88.185 |
| IMBD_count | *88.540* | 88.661 | 88.673 | 88.630 | **88.699** |
| IMBD_tfidf | **88.997** | 88.790 | 88.893 | 88.850 | *88.603* |
| IMBD_freq | 90.744 | **90.796** | 90.776 | 90.750 | *90.334* |
| SUBJ_binary | 90.545 | 90.530 | *88.750* | 88.775 | **90.885** |
| SUBJ_count | 90.625 | 90.680 | 89.060 | *88.945* | **90.730** |
| SUBJ_tfidf | 90.590 | **90.640** | 88.670 | *88.620* | 90.355 |
| SUBJ_freq | 91.285 | 91.165 | *90.255* | 90.270 | **91.350** |
| TREC_binary | **83.904** | 83.735 | *72.093* | 72.378 | 83.803 |
| TREC_count | **83.937** | 83.684 | 72.126 | *72.025* | **83.937** |
| TREC_tfidf | **82.861** | 82.591 | 71.032 | *70.477* | 82.743 |
| TREC_freq | 79.958 | 79.991 | *70.799* | 71.017 | **80.058** |

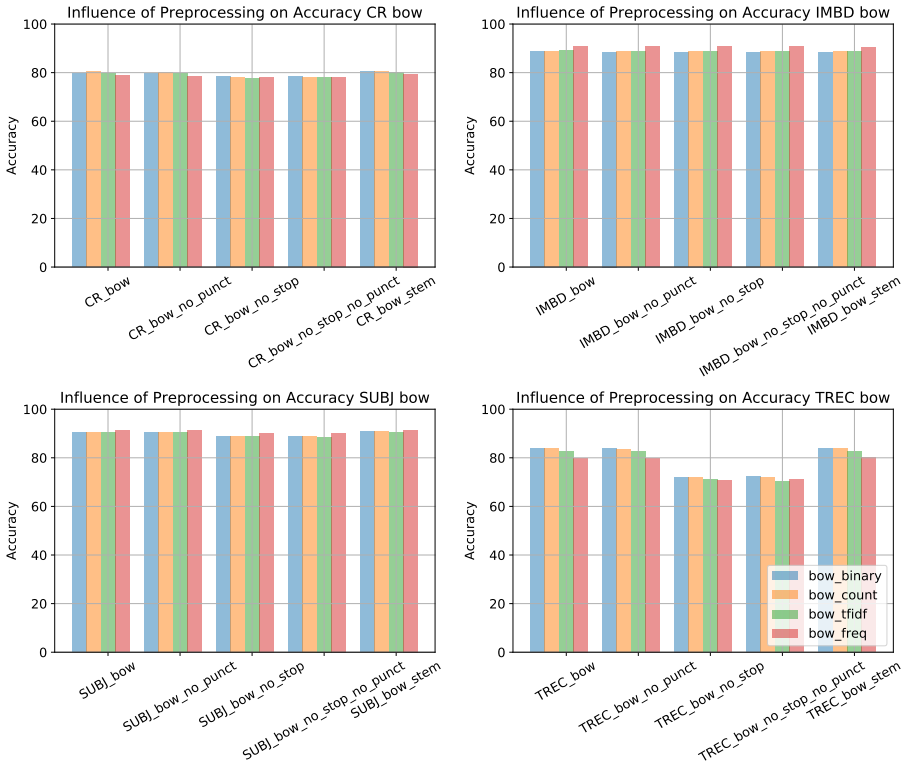



Figure 5.2: The accuracy of four BOW methods on the four different datasets and four different preprocessing methods.

For Doc2Vec the highest accuracy was reached twice with removed punctuation and twice with stemming, while all the lowest values occurred in the dataset without punctuation and stop words.

In the BOW case, the results are similar to the results from the Doc2Vec examination. However, the baseline preprocessing had twice the highest accuracy and also stemming had twice the lowest accuracy while most of the weakest accuracies were reached in the dataset without stop words and the dataset with neither stop words nor punctuation.

### 5.1.2 Preprocessing Speed

In the previous subsection, the results showed that pre-processing only has a minor effect on the accuracy of both the tested methods. Nonetheless, preprocessing has another possible benefit: Removal of stop words and removal of punctuation in general shortens the document length and remove parts of the text that does not vary much from text to text. Hence, processing cuts off the first part of the Zipf distribution. While stemming reduces different morphologies of words to one form, it leads to a reduction of vocabulary, and the vocabulary size shrinks much more with stemming than with either the removal of punctuation or with the removal of stop words. On the other hand, stemming will

not change the number of words occurring in each document. Table **??** shows the effect of the four data pre-processing steps on the two most important statistics of the corpus, mean word count per document and number of unique words per corpus.

Figure 5.3 shows that the removal of punctuation results in a insignificant learning process speed up, while the removal of stop words contributes considerably more in speeding up of the process. The Doc2Vec algorithm is sensitive to document length and the longer the document, the longer the algorithm has to train. It appears that the final vocabulary size has not much influence on the training and evaluation time.

In the case of BOW presented in Figure 5.4 the results are hard to explain as each dataset shows a relatively different pattern. In three of the four cases, the dataset without punctuation has high speed, and similar to Doc2Vec stemming has not much influence on speeding up the process. More interesting is here the difference between no punctuation, no stop words, and no punctuation which was the opposite for Doc2Vec (no punctuation slow, no punctuation and no stop words fast).

### 5.1.3 Discussion

As shown results of the two experiments neither requires heavy preprocessing to achieve higher accuracy and the differences between the pre-processed and the original datasets are neglectable. A significant difference showed here TREC where the performance decreased remarkably in both cases BOW and Doc2Vec. The removal of stop words reduced the average document length of TREC by half.

Interestingly is here that stemming did not help much to increase the accuracy even though the vocabulary size was substantially reduced.

If performance is an issue, preprocessing can help, especially the type of preprocessing that reduces the document size, such as stop word removal. This has to be done with caution though, since it can come at a price as in the example above with TREC showed.

The hypothesis concerning the preprocessing could not be verified.

Table 5.3: The results of Doc2Vec of the four datasets CR, IMBD, SUBJ and TREC preprocessed differently and compared with the baseline the basic preprocessing. In **bold** the highest accuracy per line in *italic* the lowest.

| Name of Dataset | CR | CR_no_punct | CR_no_stop | CR_no_stop_no_punct | CR_stem |
|---|---|---|---|---|---|
| Mean Document Length | 20 | 17 | 11 | 9 | 20 |
| Number of Unique Words | 5713 | 5685 | 5574 | 5546 | 4271 |
| | | | | | |
| Name of Dataset | IMBD | IMBD_no_punct | IMBD_no_stop | IMBD_no_stop_no_punct | IMBD_stem |
| Mean Document Length | 265 | 155 | 233 | 123 | 265 |
| Number of Unique Words | 157088 | 156946 | 157054 | 156912 | 125119 |
| | | | | | |
| Name of Dataset | SUBJ | SUBJ_no_punct | SUBJ_no_stop | SUBJ_no_stop_no_punct | SUBJ_stem |
| Mean Document Length | 24 | 21 | 15 | 12 | 24 |
| Number of Unique Words | 21329 | 21322 | 21195 | 21188 | 15155 |
| | | | | | |
| Name of Dataset | TREC | TREC_no_punct | TREC_no_stop | TREC_no_stop_no_punct | TREC_stem |
| Mean Document Length | 10 | 8 | 5 | 4 | 10 |
| Number of Unique Words | 8829 | 8823 | 8700 | 8694 | 7255 |

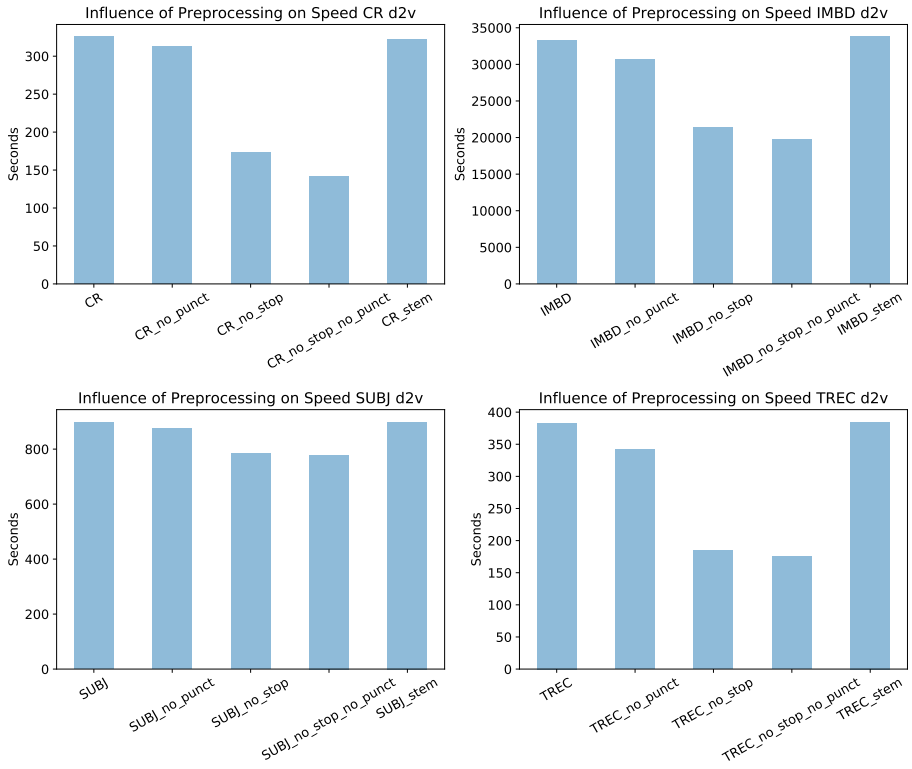



Figure 5.3: This Figure shows the process time per dataset and pre-processing steps for the Doc2Vec method.

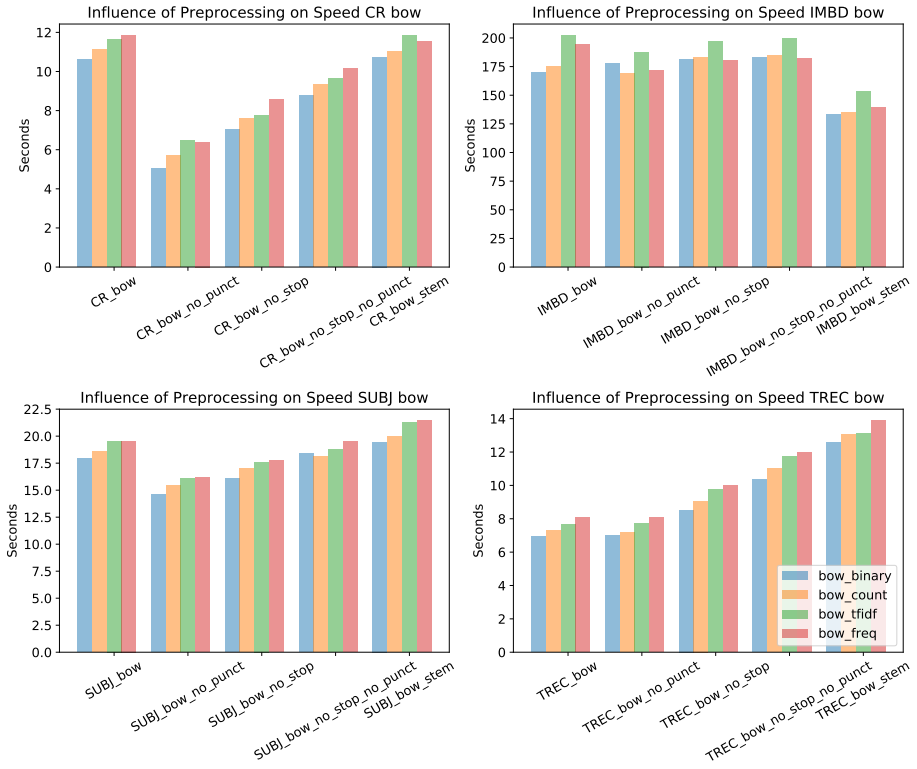



Figure 5.4: This figure shows the process time per dataset and pre-processing steps for the BOW methods.

## 5.2 Accuracy

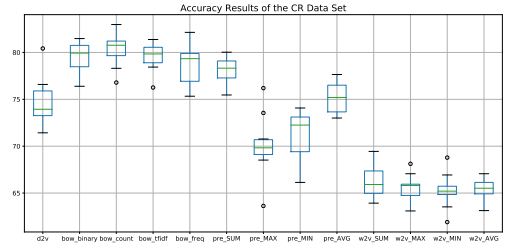
In this section, all the results of the Accuracy part of this thesis are presented. First, all the results of the methods and datasets are shown and explained. Then the two side topics on the length of the datasets and the reproducibility are shown.

### 5.2.1 by Dataset

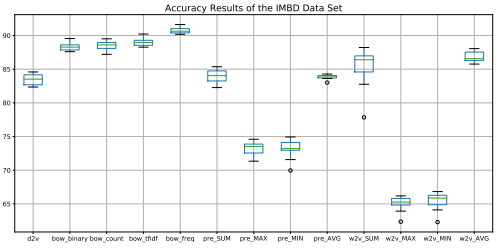
This is the first part of the Accuracy section where the accuracy of all datasets and all methods are compared and reviewed.

In Figure 5.5 d2v is short for a learned Doc2Vec vector on the training set. **bow_binary** is the Bag of Words with binary occurrence in the vector. **bow_count** is the Bag of Words with the count of the words occurring in the document. **bow_tfid** is the Bag of Words approach where their TF-IDF weights the occurrence of the words, **bow_freq** divides the counts with the total number of words such that the whole vector sums up to one. The prefix **pre_** indicates pre-trained word vector in this case from GloVe of size 300 [Pennington et al., 2014a]. **SUM** stands for the summation of all word vectors, **MIN** the minimum of each row if a document is written as multiple column vectors one

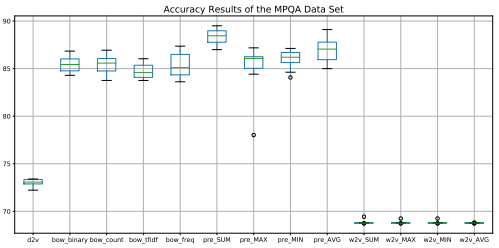
for each word. **MAX** is the same approach as before but taking the maximum. **AVG** takes the average for each row. **w2v_** stands for Word2Vec learned word vectors using the same functions as before to get one final document vector.



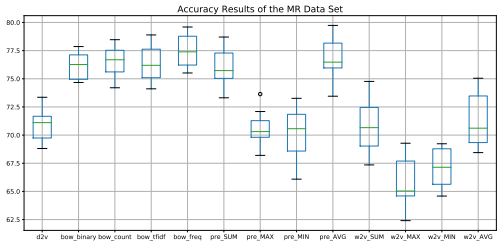

(a) CR



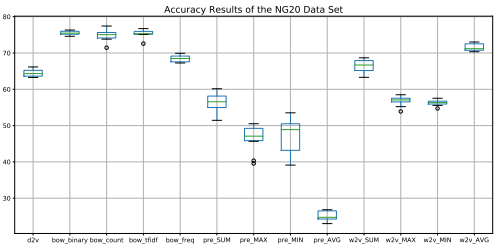

(b) IMDB



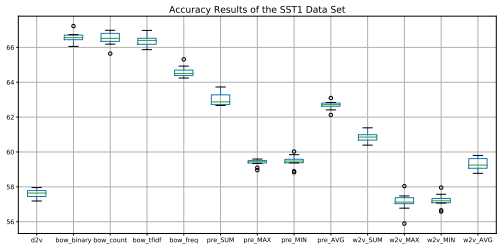

(c) MPQA



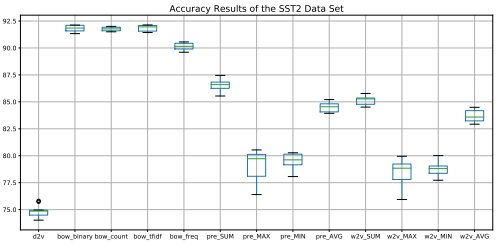

(d) MR



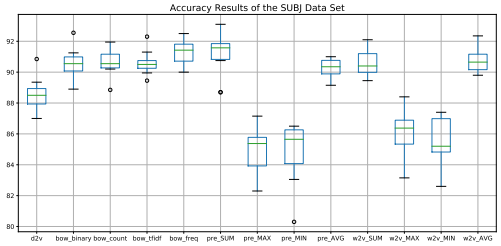

(e) NG20



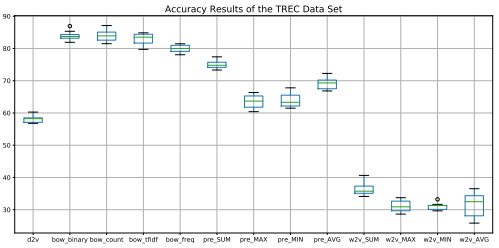

(f) SST1




(g) SST2




(h) SUBJ




(i) TREC

Figure 5.5: Results as box plots of the 10-fold cross-validation. The y-axis is scaled for each plot individually.

**CR** The size of this dataset is a problem for the Word2Vec approach which has difficulties to learn a good representation of the words. This results in bad centroids document vectors and therefore also the weakest accuracy from all approaches. The highest mean value is reached in **bow_count**.

**IMBD** The IMBD dataset has the most extensive documents with a mean count of 265 and with 50000 also a rather large number of them. These two features create the rare occasion that the learned Word2Vec vectors are better than the pre-trained word vectors. The best result is achieved in **bow_freq** and the least accurate methods are the Min and Max aggregation of Word2Vec.

**MPQA** The MPQA dataset has the fewest words per document and with 6234 words just some more than the CR dataset. These two factors make it hard for the two methods, Word2Vec and Doc2Vec, while all variations of a bag of words and most of the variations of the pre-trained vector centroids have good accuracy. Interestingly the vector centroid functions of the pre-trained word vectors are closer together than in other datasets. The few words contained in the documents might explain this, such that the minimum and maximum even in this case can return good results. The best performance was achieved in the SUM of the pre-trained word vectors. The trained word vectors show a terrible performance. This is because of the small number of words per document, similar to CR no good word vectors can be learned.

**MR** The MR dataset has a similar document size compared to CR but has with 10606 documents and 18758 words more than 3-times as much of each. The best result was achieved in **bow_freq** and the worst result is MAX of Word2Vec.

**NG20** The NG20 dataset has the second highest number of words, and the length of the document is with 174 on average the second in the ranking. A specialty of this dataset is the 20 possible classes of news articles which can be categorized. It appears that the average of the pre-trained word vectors centroids has difficulties to create document vectors that are useful for the classifier to successfully predict the 20 classes, while this is not the case for the other pre-trained variations. Rather interesting here is that the trained word vectors with Word2Vec all perform better than the pre-trained ones, as seen before in the IMBD dataset. In most cases this is the other way round. If the corpus and the documents are long enough, it can make sense also to train Word2Vec on the classification data. The BOW group has the best performance with **bow_freq** as the worst and **bow_binary** as the best. Doc2vec lies behind all the BOW variations.

**SST1** The SST dataset comes in two flavors SST1 which has five different cases of sentiment very positive, positive, neutral, negative, and very negative, and SST2 described below. This dataset has the highest number of documents with 160'128, but these documents are rather short with a mean of 7 words per document and also a relatively small number of unique words. The vast amount of documents result in a smaller variation

in the box plot result. The best result was achieved in **bow_binary**, while most of the BOW group has generally higher results. In the pre-trained word vectors, SUM and AVG are far better the MIN and MAX. Doc2Vec is now behind the pre-trained group together with the MIN and MAX of the learned Word2Vec embeddings. The reason why Doc2Vec suffers is the shorter document length in this corpus, such that the method cannot fully capitalize its potential. Important to note is that plenty of examples could not contribute to better results.

**SST2** SST2 is the second flavor of the SST data after all the neutral sentiments were cut away, and the two sentiments on both sides were joined together resulting in only two classes. Figure 5.5.g shows a similar picture where the ranking of the accuracy is about the same as in Figure 5.5.f, but the accuracy is through all methods much higher. The only parameter that changed is the number of classes, removing the difficulty to predict neutral ones. Here the best approach was **bow_tfidf**. Interestingly, the learned word vectors of the Word2Vec algorithm return almost as useful embeddings as the pre-trained ones. Doc2Vec is far behind the rest of the methods as above this might be contributed to the rather short document length of this dataset.

**SUBJ** The SUBJ dataset has a document length comparable with CR and MR datasets. The mean document length is 24. The larger document size is also visible in the performance of Doc2Vec and Word2Vec. **bow_freq** achieves the best accuracy while the whole group generally has high results. AVG and SUM from the pre-trained and the learned word vectors also reached top results.

**TREC** The TREC dataset also has only a few words per document (10), and therefore the performance of Word2Vec and Doc2Vec is rather bad, with better performance for Doc2Vec than the one from Word2Vec. Here the group of BOWs with the highest accuracy is **bow_count** followed by the pre-trained word vectors.

**Overview** The most influential parameter for learned Word2Vec and learned Doc2Vec is the document length. In the tested setup, datasets with more than 20 words per document resulted in better performance of the two methods.

All the variations of BOW resulted in the highest accuracy through most of the datasets. Yet, no a clear trend of which variation of the BOW method works best is visible.

For the pre-trained word vectors and the learned Word2Vec, the sum and the average appear to be the best variation, while the minimum and the maximum only seem to be suitable for shorter documents as shown in MPQA where all the four methods of pre-trained word vectors performed comparably.

The results show that the training of word vectors on a small number of short documents is not advisable. In some rare cases, when the documents are long (NG20, IMBD), the domain-specific trained word vectors can outperform the pre-trained ones.

In none of these cases, Doc2Vec could outperform the BOWs. This bad performance might be related to the fact that the hyperparameters were evaluated once and then used for all datasets, but these hyperparameters should be grid searched before every application for better performance. [Levy et al., 2015] state that algorithm tuning through hyperparameters is the single and most efficient way to get higher accuracy, while other influence factors like amount of data and pre-processing are less critical.

The corresponding hypothesis could not be verified that Doc2Vec could outperform older approaches in all datasets.

Noteworthy is that the error rate reported by [Le and Mikolov, 2014] for Doc2Vec in SST2 12.2% and SST1 51.3% is considerably worse than the ones reached with BOW in combination with their classifier. The best variations of BOW reached in SST2 8.15 % and SST1 22.4%.

Table 5.4: The results of all the methods for each dataset, the best result is in bold.

| | **CR** | **IMBD** | **MPQA** | **MR** | **NG20** | **SST1** | **SST2** | **SUBJ** | **TREC** |
|---|---|---|---|---|---|---|---|---|---|
| bow_binary | 79.531 | 88.309 | 85.471 | 76.083 | **75.494** | **66.578** | 91.821 | 90.545 | 83.904 |
| bow_count | **80.367** | 88.54 | 85.428 | 76.613 | 74.841 | 66.496 | 91.761 | 90.625 | **83.937** |
| bow_tfidf | 79.585 | 88.997 | 84.754 | 76.346 | 75.337 | 66.392 | **91.85** | 90.59 | 82.861 |
| bow_freq | 78.75 | **90.744** | 85.353 | **77.462** | 68.479 | 64.595 | 90.143 | **91.285** | 79.958 |
| pre_SUM | 78.087 | 83.974 | **88.332** | 76.046 | 56.322 | 63.023 | 86.556 | 91.115 | 74.984 |
| pre_MAX | 70.115 | 73.228 | 85.197 | 70.512 | 46.378 | 59.398 | 79.063 | 84.975 | 63.493 |
| pre_MIN | 71.292 | 73.161 | 85.975 | 70.212 | 47.153 | 59.446 | 79.543 | 84.885 | 63.862 |
| pre_AVG | 75.159 | 83.855 | 87.007 | 76.848 | 25.108 | 62.676 | 84.512 | 90.28 | 69.136 |
| learn_w2v_SUM | 66.227 | 85.312 | 68.838 | 70.794 | 66.482 | 60.829 | 85.159 | 90.6 | 36.458 |
| learn_w2v_MAX | 65.473 | 65.046 | 68.820 | 65.833 | 56.812 | 57.116 | 78.410 | 86.075 | 31.116 |
| learn_w2v_MIN | 65.275 | 65.371 | 68.820 | 67.136 | 56.312 | 57.202 | 78.810 | 85.445 | 30.999 |
| learn_w2v_AVG | 65.434 | 86.819 | 68.772 | 71.371 | 71.589 | 59.295 | 83.667 | 90.8 | 31.415 |
| learn_d2v | 74.735 | 83.476 | 73.009 | 70.845 | 64.451 | 57.621 | 74.86 | 88.55 | 58.148 |

### 5.2.2 by Document Length

One of the initial questions was whether or not the document size has an influence on the embedding quality and therefore also on the downstream classification task. For these tasks, nine classic books available from the NLTK package in python were selected, and non-overlapping text passages were taken from the books. These text passages had to be classified according to their author or book.

Interested mostly in the qualities of Doc2Vec, only BOW, and its variations were selected to be compared against each other. As it is visible in Figure 5.6 and 5.7 both methods have a clear upward trend and the more extensive the document, the better the accuracy. Noteworthy is that the performance of Doc2Vec, in general, is excellent in this dataset - most of the time it is even better than the BOW variations. This highly differs from the results from the Accuracy section, where Doc2Vec never had the highest accuracy. The upwards trend makes sense because more text might give more information and evidence to distinguish the book and author.

Figure 5.8 shows a non-linear and possible exponential increase in learning time compared to the document length. Initially, also a data set with 800 words was created, but this could not finish in the time frame of 48 hours[1]. A benefit from the extended initial learning time is the fixed evaluation rate. For the BOW method, the construction of the matrix or vectors is free, but the classification part generally takes much longer.

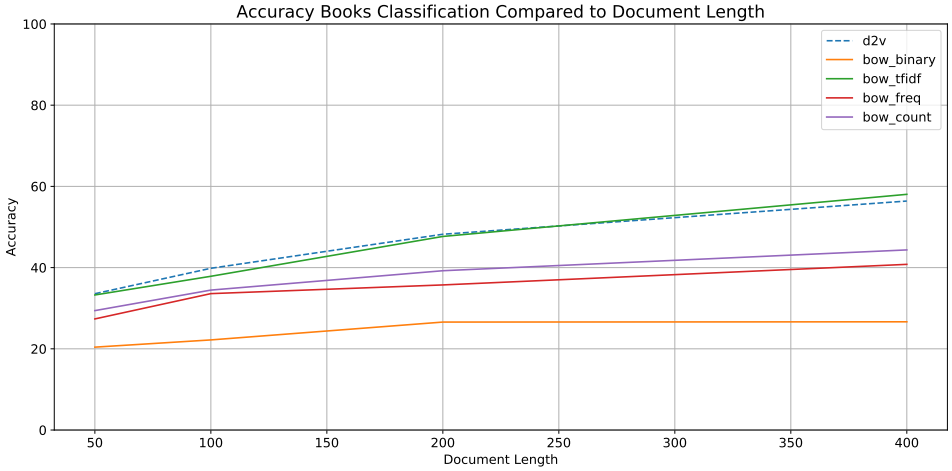



Figure 5.6: The results of two method groups BOW and Doc2Vec in document classification, where text sequences should be matched with their corresponding book.

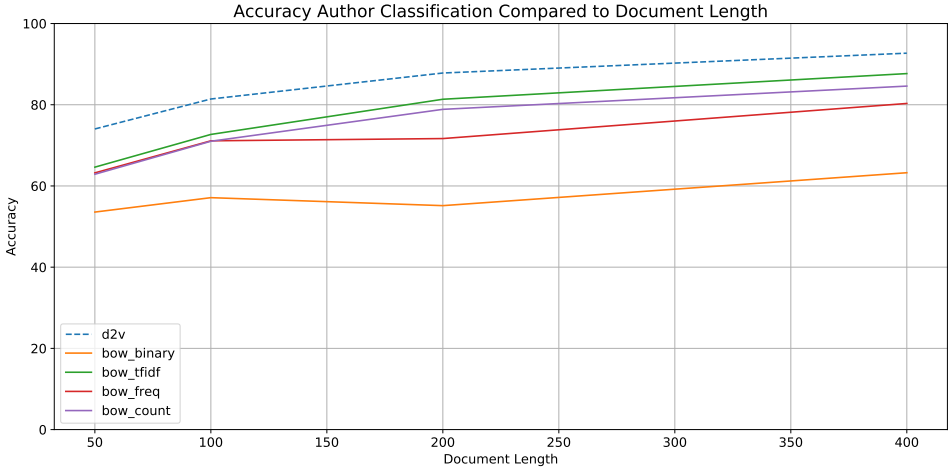



Figure 5.7: The results of two method groups BOW and Doc2Vec in document classification, where text sequences should be matched with their corresponding author.

[1] which is set by the server

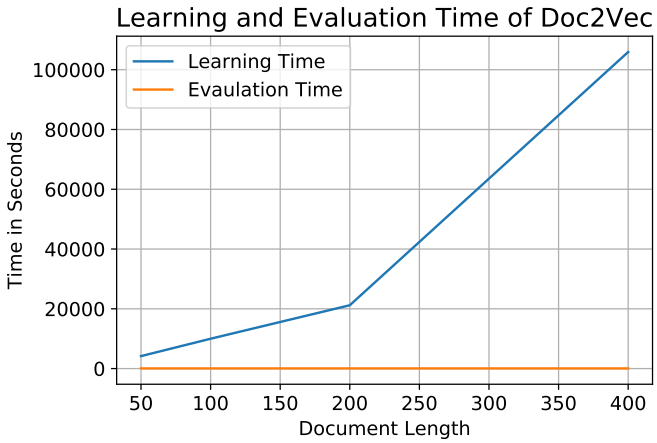



Figure 5.8: This figure shows the learning and evaluation duration of the Doc2Vec method for each of the four dataset with different document sizes.

Figure 5.9 shows the accuracy of the Doc2Vec on the y-axis plotted along the document length on the x-axis for each dataset. The blue line includes the multi-class cases and the orange line does not. A regression line was fit through the data points. The line shows an upward trend even though these measurements are not statistically significant. Here, more data points would be needed, or another way to look at the problem would be to create a regression model with a dummy variable *correctly classified* and *length of the document* to see the influence if longer documents have a higher chance of being better predicted.

In the specific tasks, the hypothesis that longer documents predict better is probably correct. But here, further investigation is needed.

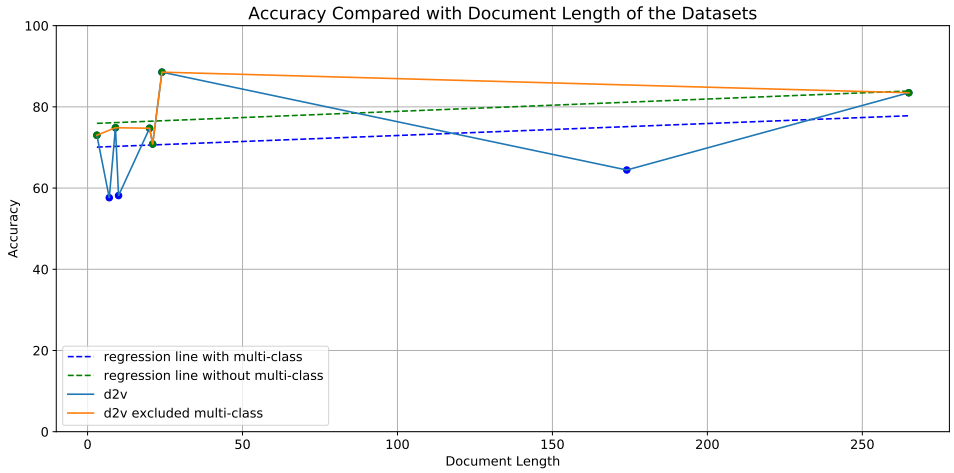



Figure 5.9: Each point in the graph represents one accuracy measurement of the Data sets. The blue regression line includes multi-class datasets, green excludes them.

### 5.2.3 by Corpus Size

In this experiment, the results of the two cases of model creation are compared. The first case uses 10-split cross-validation where Doc2Vec was trained on only the training data, then the evaluation or testing documents had to be inferred using the trained model. These newly created document vectors were then evaluated. The results were compared with the case where all documents were used to create the document vectors, and no documents had to be inferred as they were already trained. The results of this experiment are listed in Table 5.5.

Only 10% more data led to some dramatical changes, especially for the TREC dataset. This accuracy boost might be the case because the TREC dataset only has a small number of documents and also many classes to predict. Interestingly, the 10% more documents to learn the embeddings did not help CR, the dataset with the fewest documents, as much. Nonetheless, there are also some datasets that got a worse result than before. In NG20, SST1, and SST2, all of these already have a large number of documents in the corpus and therefore also probably enough data even without including all available documents.

Table 5.5: The table with the results of the comparison of Doc2Vec evaluation once trained on 90% of the data and once with all the available documents. The change is indicated with a plus or minus.

| | CR | IMBD | MPQA | MR | NG20 | SST1 | SST2 | SUBJ | TREC |
|---|---|---|---|---|---|---|---|---|---|
| train on training data | 74.735 | 83.476 | 71.163 | 70.845 | 64.451 | 55.579 | 74.86 | 88.55 | 58.148 |
| train on all data | 75.398 | 84.578 | 72.03 | 72.205 | 63.962 | 55.479 | 74.745 | 88.665 | 63.052 |
| change | +0.664 | +1.102 | +0.868 | +1.36 | -0.489 | -0.099 | -0.115 | +0.115 | +4.904 |

### 5.2.4 Reproducibility of the Doc2Vec Results from the Paper

In the paper from [Le and Mikolov, 2014], the best accuracy reached in the classification problem of the IMBD dataset was 7.42% error rate. As a start for the thesis, this result was attempted to be reproduced. Unfortunately, it could not be achieved, and the minimum error rate that could be reached was around 10%, which is even higher than the accuracy achieved in the Accuracy section. This higher accuracy could only be attained through the combination of several learned models and the evaluation of the document embedding after each epoch of training the Doc2Vec model. So no good results are missed. The achieved results are probably not based on an average of any cross-validation result, but perhaps the best case of numerous attempts. Another possible explanation why the embeddings returned a higher accuracy in the downstream task is that the corpus of the IMBD document contains 25'000 positive, 25'000 negative and 50'000 unlabeled data and in the better approaches, all data was used to learn better document embeddings. In [Le and Mikolov, 2014] they used 75'000 for the document creation and 25'000 were the evaluation set. While in most of the tested settings in this thesis, the model creation part only looked at data from the training set while ignoring the test set or the additional unlabeled documents completely.

This hypothesis could not be verified.

### 5.2.5 Accuracy by Simple Dense Neural Encoder

In Table 5.6 the results of the classification employing transformed binary BOWs are presented. The approach works better on larger datasets, not including NG20. The created results appear to be a difficult input to further classification for 20 different classes. In most cases, bow2bow has a similar performance as the other methods. But big differences can be seen in MR (10 pp difference to Doc2Vec) and NG20 (22 pp difference to Doc2Vec). This method was mainly optimized for the IMBD dataset and then applied to all others. An optimization possibility is the size of the intermediate steps in the two hidden layers for IMBD being 1200, 600. For the other datasets, a simple scaling method was used that should scale the numbers to the corresponding vocabulary size which might result in a non-optimal result.

The benefit of the creation of such a model is that the auto-encoder can encode the documents in BOW to a lower dimensional vector. In this way, time can be saved in the classification phase.

Table 5.6: The accuracy of the bow2bow auto-encoder. As a comparison also the bow_binary and Doc2vec are presented.

| | CR | IMBD | MPQA | MR | NG20 | SST1 | SST2 | SUBJ | TREC |
|---|---|---|---|---|---|---|---|---|---|
| b2b | 69.608 | 86.586 | 75.891 | 60.950 | 42.065 | 55.832 | 74.023 | 81.710 | 63.996 |
| bow_binary | 79.531 | 88.309 | 85.471 | 76.083 | 75.494 | 66.578 | 91.821 | 90.545 | 83.904 |
| learn_d2v | 74.735 | 83.476 | 73.009 | 70.845 | 64.451 | 57.621 | 74.860 | 88.550 | 58.148 |

## 5.3 Transferability

In this experiment section, the performance of models and classifiers trained on a dataset and then applied to another dataset is tested. Pre-trained word vectors is a similar application of transfer learning which was already shown to be successful.

### 5.3.1 Doc2Vec Transferability of Embedding Model

Figure 5.10 shows the difference between the results of section 5.2.1, and the accuracy that was retrieved from document embeddings generated from a Doc2Vec model learned on another dataset. In the rows the domain dataset which the Doc2Vec model was trained on is indicated, while the columns are the datasets where the documents were inferred from the trained model.

The diagonal of Figure 5.10 is zero because a dataset trained on the corpus and using the trained model to infer the document embeddings from it is already the base case. Interestingly, most of the datasets with equal or more than 20 words also perform very well on the TREC dataset. Fascinating is that there is a rather significant difference between SST1 and SST2 on TREC, one being positive the other one being negative even though the corpus is meant to be somewhat similar.

Table 5.7 shows the row-wise mean of the matrix from Figure 5.10. A higher mean accuracy means that the dataset has a wider application field while a lower number indicates that the embeddings generated from the model trained on this dataset is too specific and cannot cover a wider range of documents. Datasets with longer documents like NG20 and IMBD perform better here, compared to shorter ones. Nevertheless, the midrange datasets MR and SUBJ are not too far behind. This statistic is also slightly misleading since NG20 is a problematic dataset with its 20 classes where all of the externally trained models perform very bad. The highest value was reached with NG20, the trained embedding on NG20 gained 7.24% over the original approach learning TREC on TREC.

Table 5.8 indicates the column-wise mean, this mean shows whether a dataset is easy to classify by externally trained Doc2Vec models. As mentioned before, the NG20 dataset is challenging to predict with externally trained models. Datasets like SST1, MPQA, and TREC seem to be easily predictable by externally learned Doc2Vec models.

Table 5.7: This table shows the average accuracy of the domain dataset where the Doc2Vec model was trained on and applied on all other datasets.

| **dataset** | **CR** | **IMBD** | **MPQA** | **MR** | **NG20** | **SST1** | **SST2** | **SUBJ** | **TREC** |
|---|---|---|---|---|---|---|---|---|---|
| **average** | -8.675 | -3.047 | -10.663 | -4.266 | -2.376 | -8.864 | -6.663 | -4.686 | -9.909 |

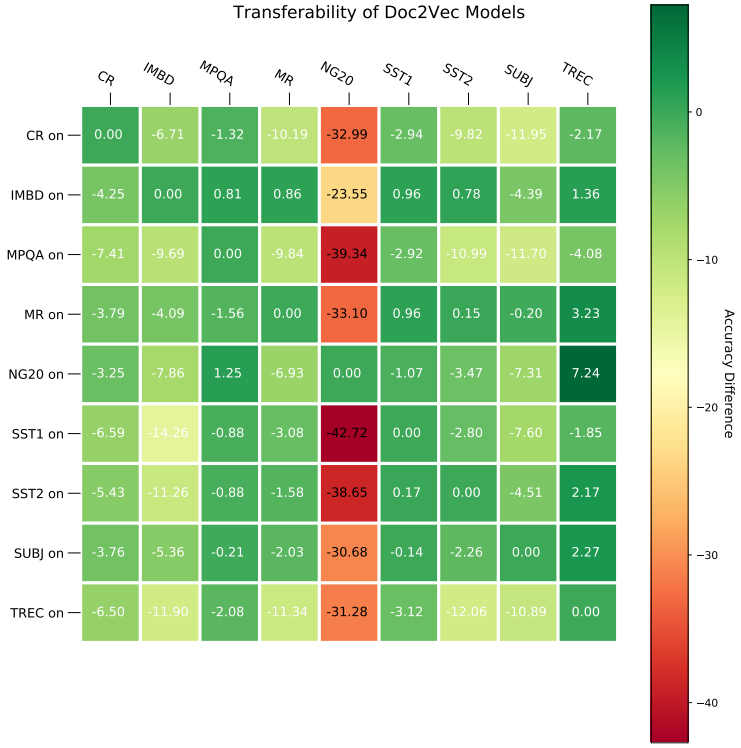



Figure 5.10: In this heat map the accuracy difference between the accuracy result after five epochs and the result of a pre-trained Doc2Vec model from another domain dataset, where all the new documents from the target dataset were inferred. Dark green is a slightly positive value, dark red a very negative one.

Table 5.8: This table shows the average accuracy of the target dataset where an externaly trained Doc2Vec model applied to generate the document embeddings.

| **dataset** | **CR** | **IMBD** | **MPQA** | **MR** | **NG20** | **SST1** | **SST2** | **SUBJ** | **TREC** |
|---|---|---|---|---|---|---|---|---|---|
| **average** | -4.552 | -7.904 | -0.542 | -4.903 | -30.258 | -0.899 | -4.495 | -6.503 | 0.908 |

### 5.3.2 Doc2Vec Transferability of Embedding Model together with Classifier

Now the document embedding model and the classifier are applied together on another dataset. Since the classifier is now used as well all the multi-class datasets are excluded. In contrast to Figure 5.10, Figure 5.11 has much lower values. These results come with no surprise since the classifier should learn dataset specific properties to distinguish the different cases better. The best result was achieved with the application of IMBD on MR. Both these datasets are about movie reviews so it generally is plausible that the model and classifier learned on the bigger dataset can also be used on the smaller one.

The weakest performing dataset to predict and to get predicted is SUBJ but also here this is easily explainable as SUBJ is trained to classify objectivity and subjectivity while all the others are trained to distinguish good from bad reviews. The next weakest dataset is MPQA, but here again, not sentiment but subjectivity is measured. The best result is achieved when SST2 is applied on MR with a lift of 1.2 pp, the next best case even though negative is the application of IMBD on MR with -1.05 pp. These datasets are all based on movie reviews, but the results show that only more elaborated classifiers can be used on simpler datasets to get a better performance but not vice-versa. The average accuracy change per field is -18.468%.

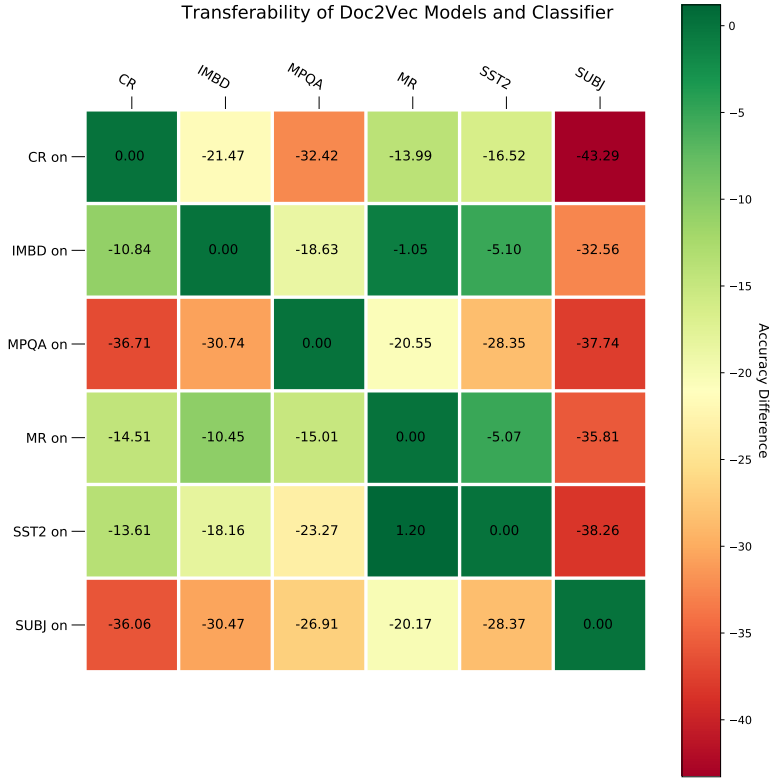



Figure 5.11: The heatmap shows on the left side of the heatmap the Dataset where the Doc2Vec model and the classifier was trained on and on top of the columns the dataset where it was applied to is indicated. Dark green is zero; dark red is a very negative value.

### 5.3.3 BOW Transferability of Document Vectorization Process Together with Classifier

This experiment is similar to the previous experiment, where a document embedding model and a classifier of a dataset was used on another dataset using Doc2Vec embeddings. Instead of using document embeddings the BOW transformation was used.

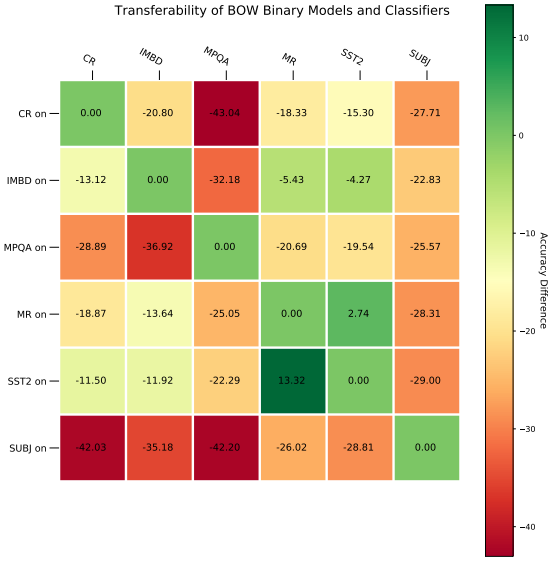



Figure 5.12

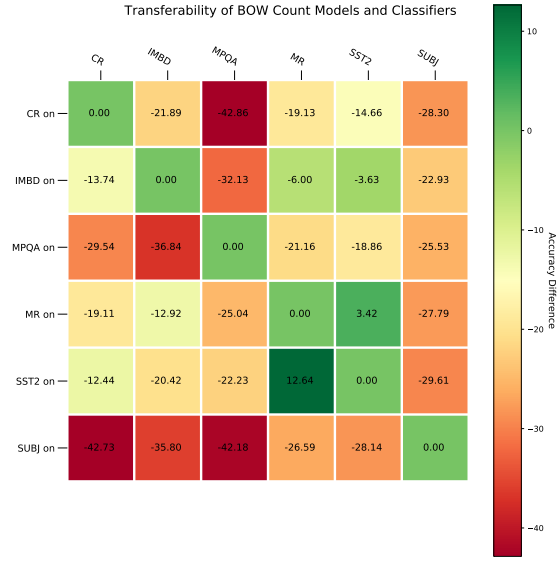



Figure 5.13

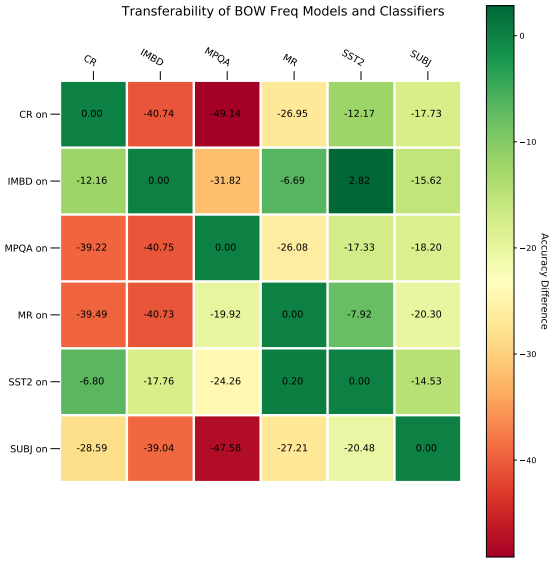



Figure 5.14

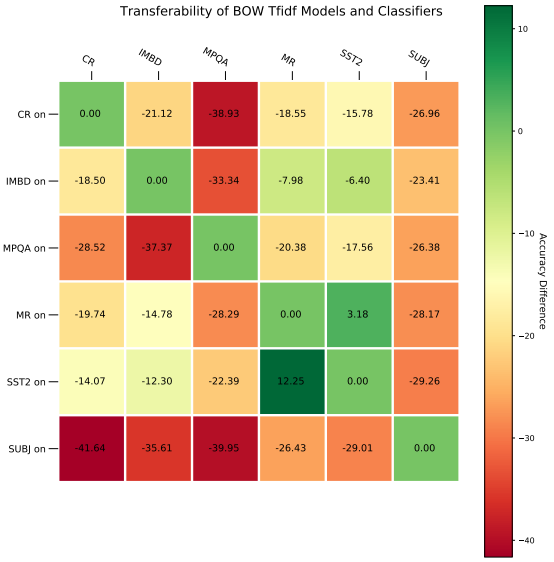



Figure 5.15

The results are comparable with the ones reached in the same experiment with the Doc2Vec embeddings. A difference is the interaction of the SST2 dataset and the MR dataset. In most of the cases, the performance of the application of an externally trained classifier results in worse performance. However, in the case of the binary, count and TF-IDF BOW the utilization of the classifier trained on SST2 applied to MR resulted in a performance boost of 13.32%, which is very remarkable. The four BOW approaches are very close to each other. **bow_binary** has an average of -18.149% per field, **bow_count**

-18.505, **bow_tfidf** -18.538% and **bow_freq** -19.617%. These results are very similar to the -18.468% reached on average per field in the Doc2Vec classifier transferability test.

Figures 5.12-5.15 show the transferability of a classifier trained on a BOW and then applied to another dataset. Note that the scale in Figure 5.14 goes not as high as in the other three cases. It could be shown that the application of a powerful classifier trained on a richer model can result in a substantial accuracy boost, however, on average, this is a nondesired approach. There is not a substantial difference between BOW and Doc2Vec when it comes to using pre-trained classifiers on other datasets.

## 5.4 Speed

The speed of the different approaches is compared in this section. This is also profoundly related to the classifier. In previous sections, the performance was measured keeping the number of epochs fixed. Now the classification task is evaluated after each epoch, giving an insight into what happens in the classifier.

First the Figures 5.16 -5.24 show the accuracy plotted over time. Then the topics speed of convergence, final accuracy and effective time taken are evaluated and reviewed.

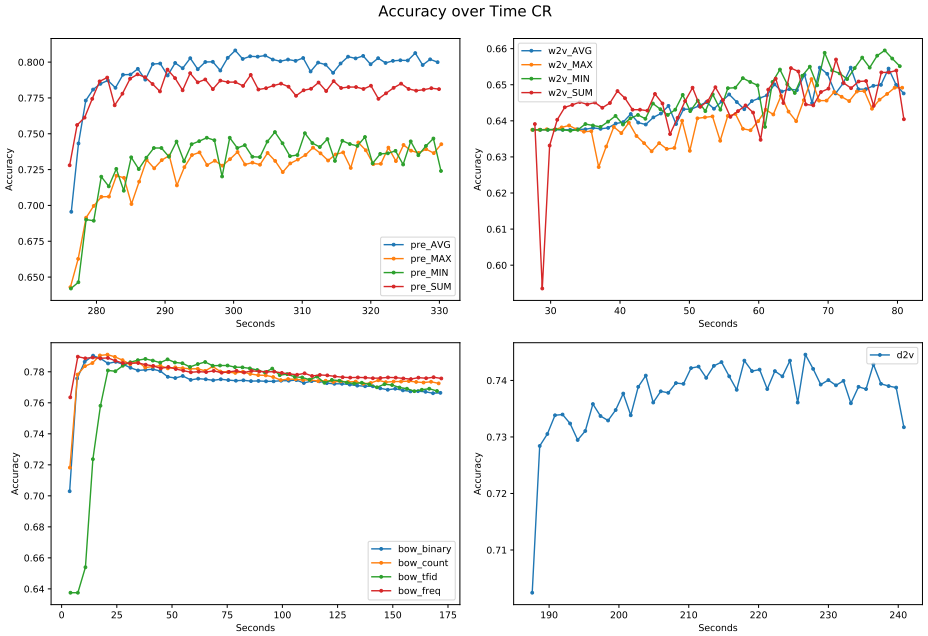



Figure 5.16: Accuracy versus time. The classifier's accuracy was measured after each training epoch. Top Left shows the results of pre-trained word vectors, Top Right the results of learned Word2Vec, Bottom Left the results of the BOW variations and Bottom Right the results of the learned Doc2Vec embedding. The lines were divided into four plots because the time distance is too big to show them in a meaningful way in one plot.

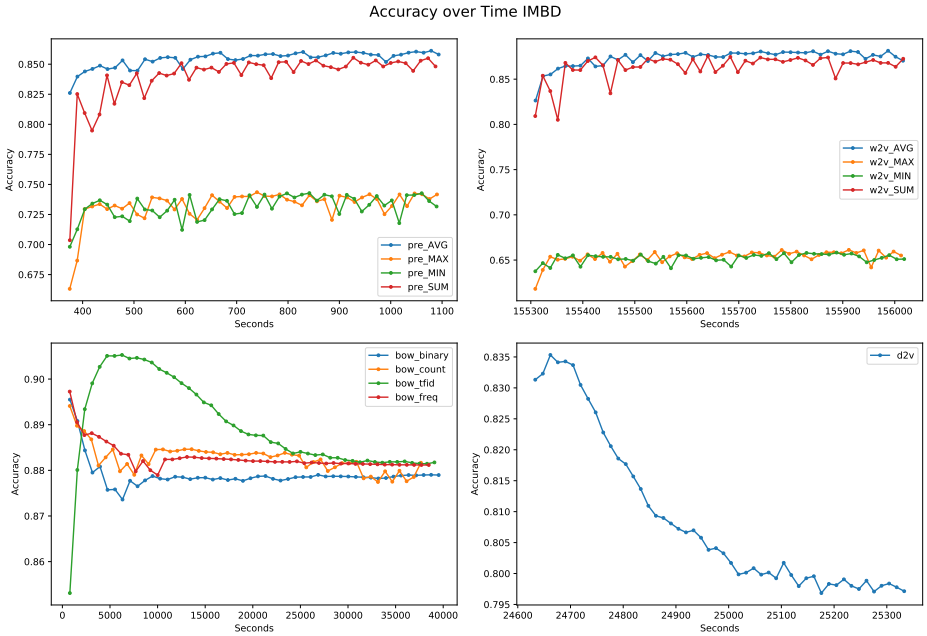



Figure 5.17: See Figure 5.16

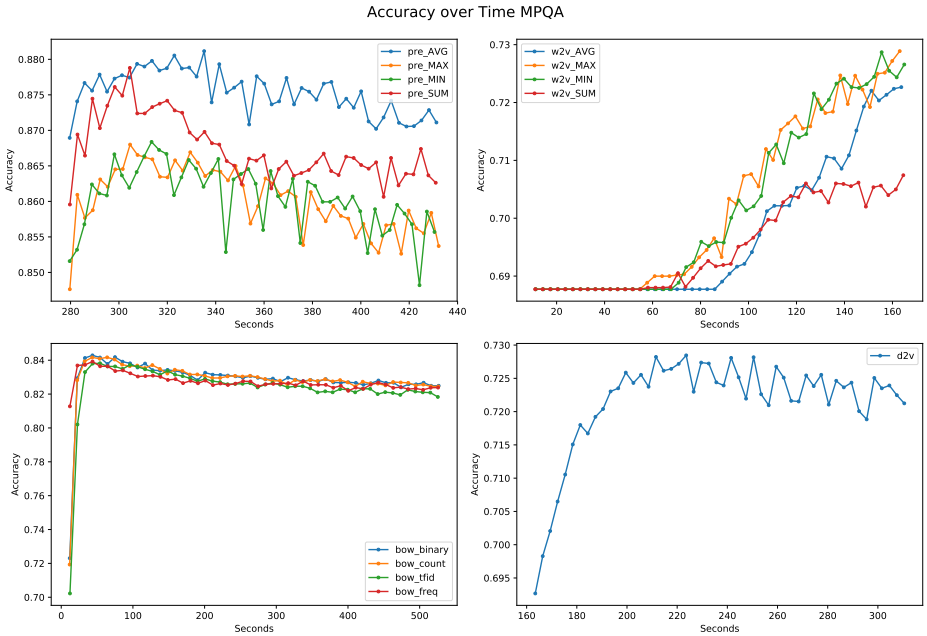



Figure 5.18: See Figure 5.16

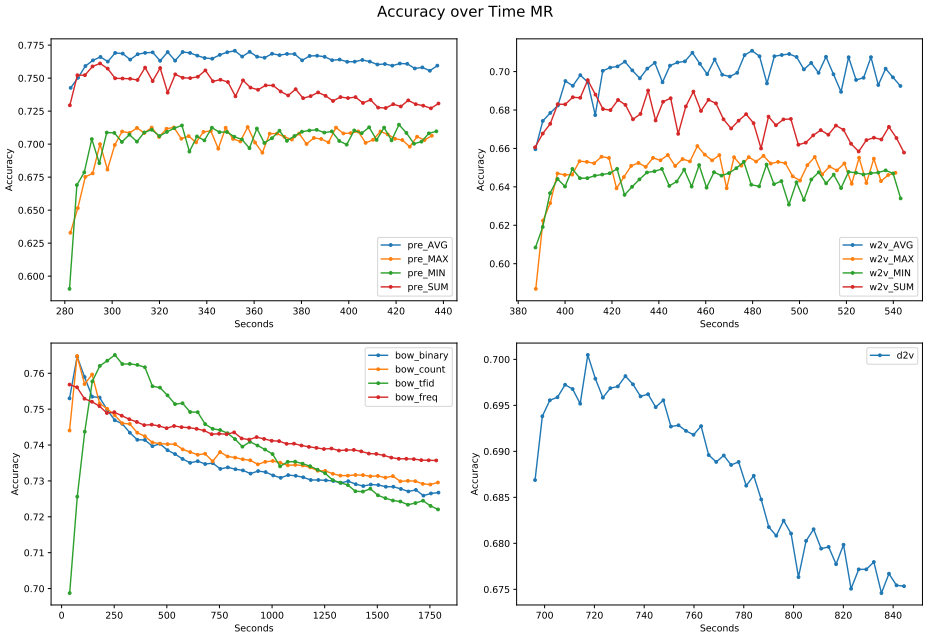



Figure 5.19: See Figure 5.16

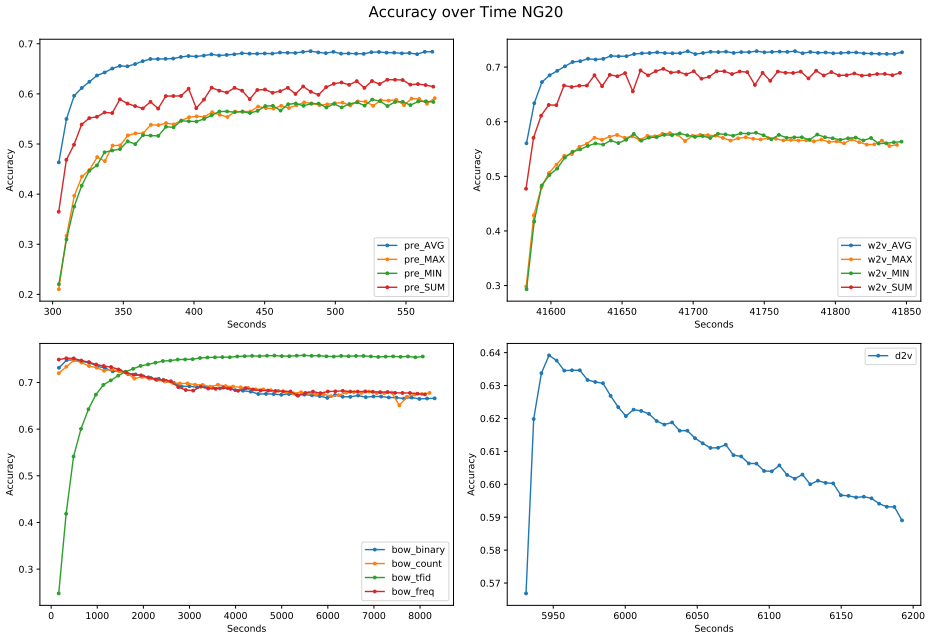



Figure 5.20: See Figure 5.16

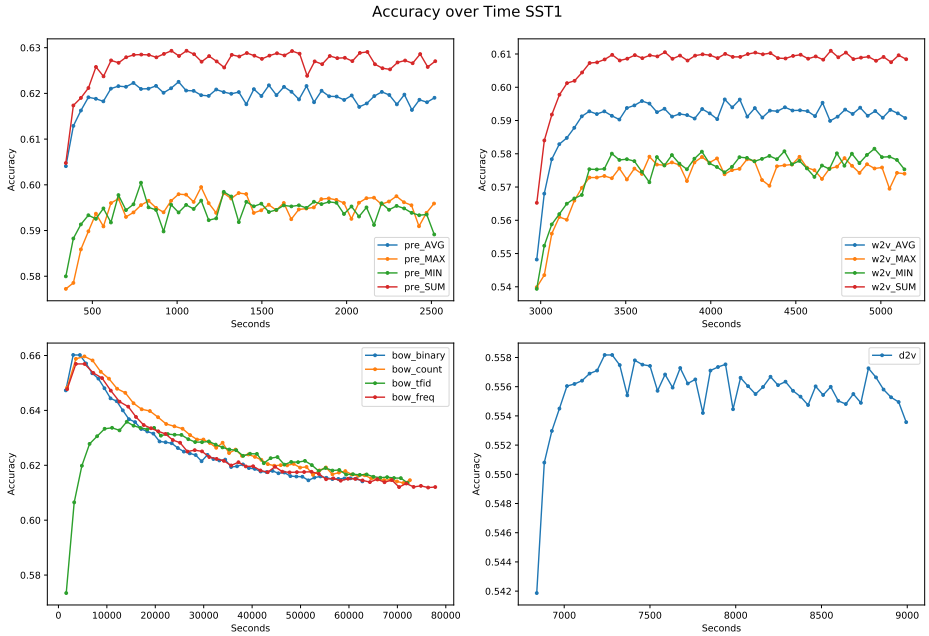



Figure 5.21: See Figure 5.16

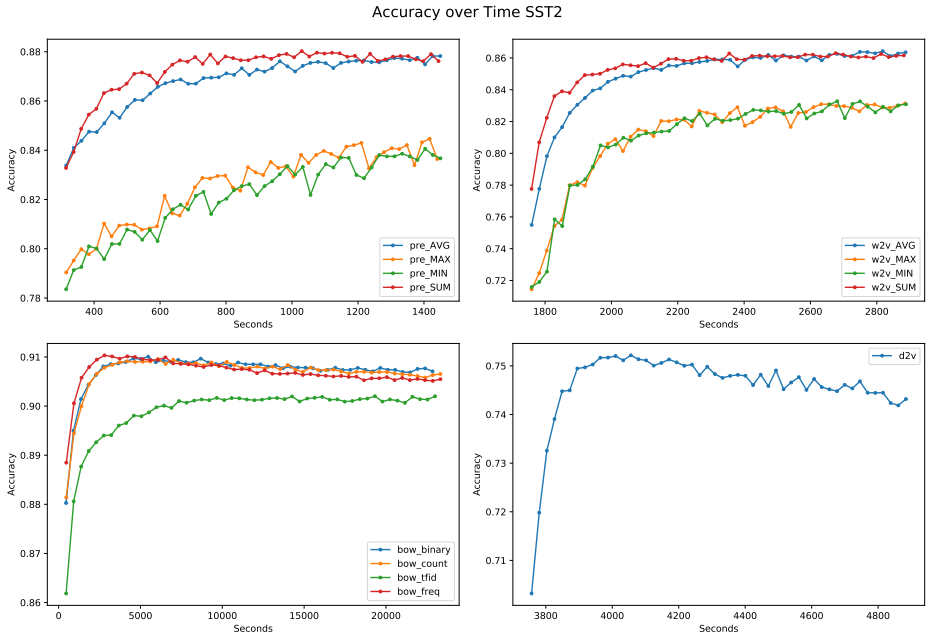



Figure 5.22: See Figure 5.16

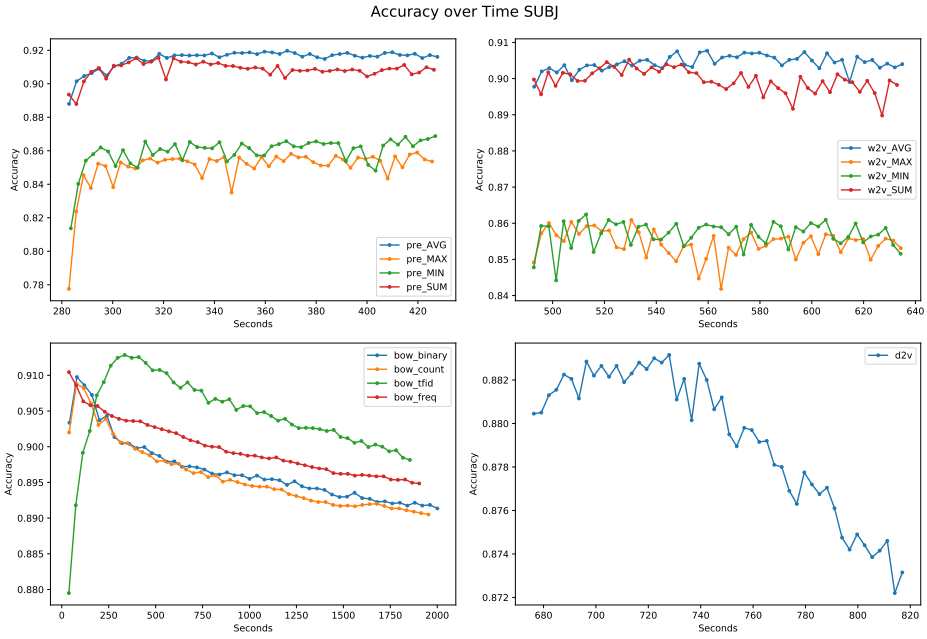



Figure 5.23: See Figure 5.16


Figure 5.24: See Figure 5.16

### 5.4.1 Speed of Convergence

**Pre-trained Vector Centroids** Good results are often reached in about ten epochs, but even then, the accuracy still grows with each epoch. Similar to the pre-trained case and the Accuracy section, the Average and Sum methods are considerably better than Minimum and Maximum.

**Learned Word2Vec** These show often a raising tendency as the vectors learned on the datasets are not as good as the pre-trained ones such that it takes longer to learn a useful function for the classifier. Similar to the pre-trained case and the Accuracy section average and Sum are considerably better than minimum and maximum. This is not the case in MPQA where after 50 epochs minimum and maximum are the highest, this is related to the short document length.

**BOW** The up and down swings of the BOW variations are smaller than in the other methods. The maximum is reached after only a few epochs and is then either slowly decreasing or straight. Bow _tfidf shows a different behavior starting low, then steeply increasing but then evens out like the others.

**Doc2Vec** In seven of the nine cases, the accuracy is slowly decreasing after reaching a maximum early in the first ten epochs.

### 5.4.2 Final accuracy

In Figure 5.9 the final accuracy evaluated after each epoch where the maximum was taken is shown. This ranking is only concerned about accuracy. Doc2Vec is only on the ninth place, after BOW and all of the pre-trained word embeddings. The trained Word2Vec embeddings could not compete against any other method. Unexpectedly, bow_binary achieves top ranking, which did not seem to be first according to the Figures 5.16-5.24, here bow_tfidf looked much more spectacular, starting low rising high.

Table 5.9: This table shows the final accuracy taking the maximum accuracy of the mean in 3-fold cross-validation, in 50 epochs with evaluation after each epoch.

| Methods | CR | IMBD | MPQA | MR | NG20 | SST1 | SST2 | SUBJ | TREC | AVG. | Rank |
|---|---|---|---|---|---|---|---|---|---|---|---|
| bow_binary | 79.041 | 89.550 | 84.287 | 76.463 | 74.854 | 66.020 | 91.004 | 90.975 | 82.662 | 81.651 | 1 |
| bow_count | 79.107 | 89.410 | 84.169 | 76.482 | 74.739 | 65.968 | 90.960 | 90.880 | 82.460 | 81.575 | 2 |
| bow_freq | 78.829 | 90.528 | 83.820 | 76.510 | 75.811 | 63.585 | 90.201 | 91.285 | 82.914 | 81.498 | 3 |
| bow_tfidf | 78.975 | 89.726 | 83.924 | 75.685 | 75.182 | 65.694 | 91.033 | 91.045 | 82.174 | 81.493 | 4 |
| learn_w2v_avg | 65.474 | 88.141 | 72.266 | 71.084 | 72.920 | 59.635 | 86.429 | 90.770 | 37.955 | 71.631 | 10 |
| learn_w2v_max | 65.156 | 66.126 | 72.888 | 66.123 | 57.908 | 57.912 | 83.139 | 86.090 | 41.450 | 66.310 | 13 |
| learn_w2v_min | 65.951 | 65.825 | 72.869 | 65.307 | 57.984 | 58.152 | 83.276 | 86.245 | 42.947 | 66.506 | 12 |
| learn_w2v_sum | 65.700 | 87.557 | 70.743 | 69.555 | 69.687 | 61.097 | 86.298 | 90.525 | 41.316 | 71.386 | 11 |
| pre_trained_avg | 80.816 | 86.118 | 88.115 | 77.073 | 68.528 | 62.253 | 87.824 | 91.970 | 77.016 | 79.968 | 5 |
| pre_trained_max | 74.391 | 74.353 | 86.800 | 71.295 | 59.149 | 59.950 | 84.466 | 85.895 | 68.683 | 73.887 | 8 |
| pre_trained_min | 75.119 | 74.276 | 86.838 | 71.473 | 58.874 | 60.047 | 84.059 | 86.875 | 68.834 | 74.044 | 7 |
| pre_trained_sum | 79.478 | 85.537 | 87.880 | 76.111 | 62.830 | 62.933 | 88.024 | 91.555 | 76.630 | 78.998 | 6 |
| learn_d2v | 74.457 | 83.532 | 72.846 | 70.048 | 63.917 | 55.817 | 75.216 | 88.315 | 60.938 | 71.676 | 9 |

However, if these results are compared with other methods in the field like AdaSent [Zhao et al., 2015] and their results (see Table 5.10) most of the accuracies are far behind. Here it is important to mention that AdaSent does not produce an output of fixed size, which is the premise of all the other approaches, it produces a multi-scale hierarchy.

Table 5.10: Accuracy of AdaSent on some of the evaluated datasets. [Zhao et al., 2015]

| **Method** | **CR** | **MPQA** | **MR** | **SUBJ** | **TREC** |
|---|---|---|---|---|---|
| AdaSent | 86.3 | 93.3 | 83.1 | 95.5 | 92.4 |

### 5.4.3 Wall Time

The wall time describes the time taken until the maximum in 50 epochs is reached. Table 5.11 shows the times in seconds for each method and dataset, and in the last column, the actual ranking is presented. The results are rather impressive from the viewpoint of application of the models, where BOW took very long to calculate every result for 50 epochs. An advantage is that maxima are reached quite fast for these BOW algorithms. Measuring the time until a maximum is reached favors vigorously vector representations that produce early good results and then decrease gradually, but is generally favorable.

Following parts that were included in the wall time calculation and some parts that were not. Pre-processing was not taken into account since it was the same for all methods and datasets. Despite the duration of epochs, which Table 5.12 lists, the different methods had other operations that had to be conducted before the training phase could start. For BOW one of these steps was the vectorization process, which is tied to the tokenizer. This was left out because of the negligible duration of this process. For the Word2Vec the word vectors had to be learned, and then from these learned vectors a document vector was created either taking the average, the sum, the minimum or the maximum, these two processes were included in Figure 5.11. In the pre-trained set the data had to be loaded from the file and also the document vector had to be constructed similarly as in the Word2Vec approach. In the last method, the Doc2Vec model had to be trained to create document embeddings from the text.

Table 5.11: Wall time of the different methods, specifically their time till the maximum in 50 epochs was reached.

| **Methods** | **CR** | **IMBD** | **MPQA** | **MR** | **NG20** | **SST1** | **SST2** | **SUBJ** | **TREC** | **AVG.** | **Rank** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| bow_binary | 14.24 | 791.26 | 43.56 | 73.76 | 501.32 | 3047.00 | 5475.35 | 80.33 | 45.30 | 1119.13 | 6 |
| bow_count | 20.77 | 774.56 | 64.05 | 73.61 | 493.02 | 5363.71 | 6070.18 | 78.38 | 38.92 | 1441.91 | 7 |
| bow_freq | 37.95 | 6260.71 | 54.10 | 252.92 | 5485.82 | 14102.41 | 19322.25 | 334.92 | 129.88 | 5108.99 | 9 |
| bow_tfidf | 7.34 | 770.47 | 43.69 | 37.72 | 328.25 | 3603.61 | 2799.83 | 38.06 | 26.42 | 850.60 | 5 |
| learn_w2v_avg | 73.26 | 155987.17 | 163.70 | 479.75 | 41771.63 | 4082.17 | 2817.85 | 559.62 | 94.30 | 22892.16 | 12 |
| learn_w2v_max | 67.60 | 155911.67 | 163.03 | 456.43 | 41683.64 | 3639.17 | 2885.33 | 530.39 | 94.13 | 22825.71 | 10 |
| learn_w2v_min | 78.17 | 155888.12 | 155.50 | 476.24 | 41744.15 | 4962.10 | 2680.53 | 512.92 | 95.75 | 22954.83 | 13 |
| learn_w2v_sum | 71.14 | 155640.53 | 164.52 | 409.67 | 41679.04 | 4705.42 | 2675.56 | 529.53 | 96.97 | 22885.82 | 11 |
| pre_trained_avg | 300.24 | 1078.74 | 335.22 | 351.85 | 482.50 | 1008.72 | 1426.46 | 368.60 | 360.03 | 634.71 | 4 |
| pre_trained_max | 318.17 | 739.54 | 304.60 | 357.28 | 570.31 | 1141.81 | 1417.31 | 419.72 | 322.55 | 621.25 | 3 |
| pre_trained_min | 306.02 | 841.57 | 313.49 | 421.22 | 526.10 | 786.96 | 1403.70 | 426.84 | 348.07 | 597.11 | 2 |
| pre_trained_sum | 290.36 | 927.22 | 304.51 | 294.90 | 542.23 | 1055.49 | 1029.49 | 318.02 | 362.59 | 569.42 | 1 |
| learn_d2v | 226.70 | 24661.83 | 223.59 | 717.35 | 5947.07 | 7279.71 | 4055.87 | 727.98 | 246.63 | 4898.53 | 8 |

In Table 5.12 the duration of an epoch over all datasets is presented. Here it is clearly visible that BOW is indeed the slowest of these methods in the classification phase. Four factors change the duration of the epochs for the chosen classifier. The first is the number of examples in the dataset. It is unchanged for all methods for the same dataset. The next factor is the input size of the neural classifier while for learned Word2Vec, pre-trained word embedding or Doc2Vec this is always 300 and for the BOW variations this is depending on the dataset and the number of words uniquely occurring in it. The third factor is the number of classes that have to be classified while fewer classes are usually faster than more classes. This is especially visible in the difference between SST1 and SST2 being the same corpus, even though the number of documents is not the same. The last factor is only concerning the BOW variation. As in some classification methods, a continuous number might be a problem and therefore need to be binned. This might not be the reason here, but it appears that the calculation with natural numbers, is more straightforward and therefore less time-consuming than small decimal numbers from the bow_tdidf. The fastest epochs in the BOW group are on average reached in the binary mode, but if the numbers are closely checked the main advantage of bow_binary is visible in the SST1 dataset, this is a dataset with five classes and many documents of short length, where on all other datasets the bow_count is faster.

Table 5.12: Duration of an epoch across all datasets.

| **Methods** | **CR** | **IMBD** | **MPQA** | **MR** | **NG20** | **SST1** | **SST2** | **SUBJ** | **TREC** | **AVG.** | **Rank** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| bow_binary | 3.431 | 790.498 | 10.530 | 35.814 | 166.353 | 1255.445 | 456.906 | 40.017 | 6.395 | 307.266 | 10 |
| bow_count | 3.416 | 753.241 | 10.521 | 35.760 | 164.124 | 1451.824 | 466.432 | 39.063 | 6.396 | 325.642 | 11 |
| bow_freq | 3.401 | 781.600 | 10.507 | 35.771 | 161.185 | 1442.453 | 459.926 | 37.088 | 6.454 | 326.487 | 12 |
| bow_tfidf | 3.440 | 769.675 | 10.513 | 35.600 | 162.110 | 1556.499 | 466.499 | 38.085 | 6.425 | 338.761 | 13 |
| learn_w2v_avg | 1.098 | 14.448 | 3.143 | 3.220 | 5.398 | 44.222 | 23.012 | 2.919 | 1.712 | 11.019 | 5 |
| learn_w2v_max | 1.094 | 14.374 | 3.129 | 3.178 | 5.327 | 44.131 | 22.982 | 2.907 | 1.708 | 10.981 | 2 |
| learn_w2v_min | 1.087 | 14.495 | 3.167 | 3.222 | 5.391 | 44.117 | 23.032 | 2.906 | 1.707 | 11.014 | 3 |
| learn_w2v_sum | 1.099 | 14.456 | 3.159 | 3.251 | 5.369 | 44.328 | 22.906 | 2.877 | 1.698 | 11.016 | 4 |
| pre_trained_avg | 1.102 | 14.683 | 3.125 | 3.212 | 5.402 | 44.390 | 23.172 | 2.979 | 1.720 | 11.087 | 9 |
| pre_trained_max | 1.112 | 14.617 | 3.144 | 3.165 | 5.436 | 44.333 | 22.987 | 2.937 | 1.711 | 11.049 | 6 |
| pre_trained_min | 1.111 | 14.602 | 3.109 | 3.203 | 5.418 | 44.359 | 23.180 | 2.963 | 1.718 | 11.074 | 8 |
| pre_trained_sum | 1.106 | 14.560 | 3.120 | 3.221 | 5.415 | 44.480 | 23.067 | 2.952 | 1.704 | 11.070 | 7 |
| learn_d2v | 1.089 | 14.286 | 3.006 | 3.031 | 5.335 | 43.955 | 22.991 | 2.876 | 1.647 | 10.913 | 1 |

# 6

# Limitations

This thesis showed empirical results of the application of various simpler methods in comparison to Doc2Vec, across different datasets in order to give advice to practitioners in the field of document embeddings as well as the application of these embeddings in sentiment analysis or in general label prediction.

Nonetheless the following main limitations might be spotted in this thesis:

The evaluated datasets were rather short ranging from 3 to 265, while most of the datasets have less than 25 words per document. The shortness of the documents could also have influenced the performance of the tested methods Word2Vec and Doc2Vec.

The transferability of pre-trained document embedding models could be shown but what factors lead to a better performance and what features a candidate should have in order to be applied in a wide range of document still needs to be clarified.

A simple method was shown how to compress a binary BOW into a low dimensional vector to be used for a document classification task. This method was mostly tested on the IMBD dataset and then just applied on smaller ones, therefore a simple function was created that scales the hidden layers of the auto-encoder dynamically. This function should be refined in order to better work also on smaller dataset. The proposed method was only checked with extrinsic evaluation but whether or not the distance between these encoded embeddings also have similar information about their semantic relatedness or not was not checked.

# 7

# Future Work

As shown in Chapter 6, there are limitations to this work.

Some experiments showed that longer documents also resulted in better accuracy. This effect could be addressed with a broader range of documents, and created document with various lengths. Here also a linear regression to check the influence of the continuous variable document length with the dummy variable correctly predicted could be interesting.

As already proposed, an attempt of creating a Doc2Vec embedding model similar to the *GloVe 42B Common Crawl*[Pennington et al., 2014a] would be of help for a wide range of ML tasks and could help people to overcome the parameter tuning step.

But also possible would be a better weighting scheme for each word of the pre-trained word vectors learned on a huge amount of text such that the aggregation of word vectors multiplied by their weighting would result in a good document representation, but it is questionable if such a rigid weighting scheme even exists.

# 8

# Conclusions

In this thesis a range of baseline approaches to derive document vector representations from text were compared in order to asses whether newer approaches like Doc2Vec outperform simpler methods like BOW in typical downstream classification tasks.

In the different benchmarking tests BOW outperformed the used Doc2Vec configuration. But also other approaches like the word vector centroids of pre-trained word vectors usually showed a better performance than Doc2Vec. The neural language models Word2Vec and Doc2Vec suffer especially when the documents are short to build a sufficiently good dense vector representation. This could be show in the case of the datasets SST1 and SST2 where BOW showed a much better performance then Doc2Vec the result reported by [Le and Mikolov, 2014].

It could be shown that the training of Doc2Vec on other corpora could still create a Doc2Vec model powerful enough to use for satisfactory prediction in downstream tasks.

It could be shown that a pre-trained BOW classifier can return much better results if used on the right dataset.

The results showed that learning Word2Vec embeddings on a classification data sets can only be useful if the dataset has long enough documents but in this case, it can give a slight advantage over pre-trained vectors.

If evaluation speed is essential and BOW shows good accuracy a method to compress large BOWs to be used for constant short evaluation was shown.

One could argue that Doc2Vec was not applied right and some of the hyperparameters should have been set differently. That might well be, but this is also a severe disadvantage over a method like BOW where primarily no hyperparameters exist. This makes BOW less versatile, but as shown in the results with the tested datasets it could still well compete against newer methods, like Word2Vec, pre-trained word embedding centroids and also the applied version on Doc2Vec with the used hyperparameters. As mentioned in [Levy et al., 2015], hyperparameter tuning is the most critical optimization technique of complex hyperparameter rich methods such as Doc2Vec.

# References


[Anaconda, 2016] Anaconda (2016). Anaconda software distribution.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*. arXiv: 1409.0473.

[Baldi, 2011] Baldi, P. (2011). Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, pages 37–50. JMLR.org.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

[Blei, 2003] Blei, D. M. (2003). Latent Dirichlet Allocation. page 30.

[Bojanowski et al., 2016] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv:1607.04606 [cs]*. arXiv: 1607.04606.

[Brokos et al., 2016] Brokos, G.-I., Malakasiotis, P., and Androutsopoulos, I. (2016). Using centroids of word embeddings and word mover's distance for biomedical document retrieval in question answering. In *Proceedings of the 15th Workshop on Biomedical Natural Language Processing*, pages 114–118, Berlin, Germany. Association for Computational Linguistics.

[Cer et al., 2018] Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.-H., Strope, B., and Kurzweil, R. (2018). Universal Sentence Encoder. *arXiv:1803.11175 [cs]*. arXiv: 1803.11175.

[Chen, 2017] Chen, M. (2017). Efficient Vector Representation for Documents through Corruption. *arXiv:1707.02377 [cs]*. arXiv: 1707.02377.

[Chen et al., 2016] Chen, S., Chen, G., and Wang, W. (2016). The joint effect of semantic and syntactic word embeddings on sentiment analysis. In *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, pages 366–370.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. *https://keras.io*.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA. ACM.

[Dai et al., 2015] Dai, A. M., Olah, C., and Le, Q. V. (2015). Document Embedding with Paragraph Vectors. *arXiv:1507.07998 [cs]*. arXiv: 1507.07998.

[De Boom et al., 2016] De Boom, C., Van Canneyt, S., Demeester, T., and Dhoedt, B. (2016). Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters*, 80:150–156. arXiv: 1607.00570.

[Finkelstein et al., 2002] Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2002). Placing search in context: the concept revisited. *ACM Trans. Inf. Syst.*, 20(1):116–131.

[Finley et al., 2017] Finley, G., Farmer, S., and Pakhomov, S. (2017). What Analogies Reveal about Word Vectors and their Compositionality. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017)*, pages 1–11, Vancouver, Canada. Association for Computational Linguistics.

[Ganitkevitch et al., 2013] Ganitkevitch, J., Van Durme, B., and Callison-Burch, C. (2013). PPDB: The Paraphrase Database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, Atlanta, Georgia. Association for Computational Linguistics.

[Goth, 2016] Goth, G. (2016). Deep or shallow, nlp is breaking out. *Commun. ACM*, 59(3):13–16.

[Hinton, 1986] Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12. Hillsdale, NJ: Erlbaum.

[Hu and Liu, 2004] Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 168–177, New York, NY, USA. ACM.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95.

[Jette et al., 2002] Jette, M. A., Yoo, A. B., and Grondona, M. (2002). Slurm: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag.

[Ji et al., 2015] Ji, S., Yun, H., Yanardag, P., Matsushima, S., and Vishwanathan, S. V. N. (2015). WordRank: Learning Word Embeddings via Robust Ranking. *arXiv:1506.02761 [cs, stat]*. arXiv: 1506.02761.

[Kiros et al., 2015] Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R., and Fidler, S. (2015). Skip-Thought Vectors. *arXiv:1506.06726 [cs]*. arXiv: 1506.06726.

[Kluyver et al., 2016] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., and development team [Unknown], J. (2016). Jupyter notebooks ? a publishing format for reproducible computational workflows. In Loizides, F. and Scmidt, B., editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press.

[Kusner et al., 2015] Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From Word Embeddings To Document Distances. page 10.

[Landauer et al., 1998] Landauer, T. K., Foltz, P. W., and Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284.

[Lang, 1995] Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339.

[Lau and Baldwin, 2016] Lau, J. H. and Baldwin, T. (2016). An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation. pages 78–86. Association for Computational Linguistics.

[Le and Mikolov, 2014] Le, Q. V. and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. *arXiv:1405.4053 [cs]*. arXiv: 1405.4053.

[Levy et al., 2015] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3:211–225.

[Li and Roth, 2002] Li, X. and Roth, D. (2002). Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Linzen, 2016] Linzen, T. (2016). Issues in evaluating semantic spaces using word analogies. *arXiv:1606.07736 [cs]*. arXiv: 1606.07736.

[Loper and Bird, 2002] Loper, E. and Bird, S. (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. arXiv: 1301.3781.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. *arXiv:1310.4546 [cs, stat]*. arXiv: 1310.4546.

[Miller, 1995] Miller, G. A. (1995). Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

[Pagliardini et al., 2017] Pagliardini, M., Gupta, P., and Jaggi, M. (2017). Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features. *arXiv:1703.02507 [cs]*. arXiv: 1703.02507.

[Pang and Lee, 2004] Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Pang and Lee, 2005] Pang, B. and Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL*, pages 115–124.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014a] Pennington, J., Socher, R., and Manning, C. (2014a). *GloVe 42B 300, Common Crawl. http://nlp.stanford.edu/data/glove.42B.300d.zip*.

[Pennington et al., 2014b] Pennington, J., Socher, R., and Manning, C. (2014b). Glove: Global Vectors for Word Representation. pages 1532–1543. Association for Computational Linguistics.

[Porter, 1997] Porter, M. F. (1997). Readings in information retrieval. chapter An Algorithm for Suffix Stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Řehůřek and Sojka, 2010] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. *http://is.muni.cz/publication/884893/en*.

[Rogers et al., 2017] Rogers, A., Drozd, A., and Li, B. (2017). The (too Many) Problems of Analogical Reasoning with Word Vectors. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017)*, pages 135–148, Vancouver, Canada. Association for Computational Linguistics.

[Socher et al., ] Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank.

[Weston et al., 2014] Weston, J., Chopra, S., and Adams, K. (2014). #TagSpace: Semantic Embeddings from Hashtags. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1822–1827, Doha, Qatar. Association for Computational Linguistics.

[Wiebe et al., 2005] Wiebe, J., Wilson, T., and Cardie, C. (2005). Annotating Expressions of Opinions and Emotions in Language. *Language Resources and Evaluation*, 39(2-3):165–210.

[Wu et al., 2017] Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A., and Weston, J. (2017). StarSpace: Embed All The Things! *arXiv:1709.03856 [cs]*. arXiv: 1709.03856.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *arXiv:1502.03044 [cs]*. arXiv: 1502.03044.

[Zhao et al., 2015] Zhao, H., Lu, Z., and Poupart, P. (2015). Self-Adaptive Hierarchical Sentence Model. *arXiv:1504.05070 [cs]*. arXiv: 1504.05070.

# List of Figures

# List of Tables