ERASMUS UNIVERSITEIT ROTTERDAM
ERASMUS SCHOOL OF ECONOMICS

MASTERS THESIS FOR A DEGREE IN OPERATIONS RESEARCH AND QUANTITATIVE LOGISTICS

Investigating the Performance of Different Feature Representations in Text Classification within Machine Learning

Author
Daniël den Brave, 450190

Supervisor
Dr. Michel van de Velden
Second assessor
Dr. Flavius Frasincar

DEPARTMENT OF ECONOMETRICS

ERASMUS UNIVERSITY ROTTERDAM

September 10, 2018

Abstract

Automatic text classification has numerous applications such as information retrieval out of a corpus, sentiment analysis of text or spam filtering. The use of machine learning in these text classification tasks are increasing in both frequency and performance. Unfortunately, this fast performance increase in machine learning techniques for text classification is often not accompanied with a better theoretical understanding of these machine learning models. This research investigates the performance and links to related theory behind some automatic text classification methods.

Broadly speaking, automatic text classification consists of three steps. The first step is pre-processing, which reduces noise and removes unwanted textual features. The second step is feature representation, which transforms the textual corpus to a numerical data matrix. The last step is the classification of this data matrix using a classifier. This thesis specifically focuses on two feature representation methods: Bag-of-Words and Bag-of-Concepts. The performance as well as the important features of the models are investigated in this thesis.

The Bag-of-Words is commonly used and represents a document by the frequency of its words. On top of the word term frequency representation, we also investigate the effect of applying different weighting schemes (TFIDFs) to the Bag-of-Words as well as using boolean features instead of term frequencies.

Bag-of-Concepts is an alternative to Bag-of-Words and works similarly through representing a document by the frequency of its concepts. These concepts are made by clustering word vectors generated by the `word2vec` algorithm, which assigns each word a vector based on the context it appears in. The clusters, the numbers ranging in size between 10 and 1000, were made using K-means. All of these feature representations are classified with multinomial naive Bayes.

In this thesis three data sets were used, consisting of labeled documents. The first data set investigated, R52, is a subset of the commonly used Reuters-21578 news set. The other data sets have as source questions from the Q&A forum StackExchange: one data set contains questions regarding computer games and the last data set contains questions regarding the English language.

Of the four Bag-of-Words variants that were evaluated in this research, the word term frequencies Bag-of-Words was found to be the best performing one, as measured using F1-score. The boolean Bag-of-Words performs slightly worse, it is however a good alternative. In contrast to literature, the use of TFIDF decreased the performance of the classifier. It is hypothesized that this is due to the lack of parameter optimization as well as feature selection in this thesis. But, albeit investigated, a solid theoretical explanation for this result cannot be given in this research.

In general, the Bag-of-Words representation was found to be performing somewhat better than the Bag-of-Concepts representation. Nevertheless, a significant dimension reduction was achieved using Bag-of-Concepts and the word clusters (concepts) that arise from the Bag-of-Concepts method were considered (humanly) relevant to the classification tasks. Because of this, the Bag-of-Concepts method could be used to identify important keywords for a certain class.

Acknowledgements

This thesis would not have been completed without the help of a few people. I would like to take a small paragraph to thank them all. First of all, obviously Avinash. Thank you for the many different types of support. No Python question was too stupid and your enthousiasm for coding is something that made a large impression on me. Your were eager to help with every aspect of my thesis, and maybe most importantly, showed me around at PwC. Thank you a lot for all your help!

Thank you Michel for supervising my thesis, giving me guidelines and remarks and for taking your time, even when abroad in many different countries, to answer my questions.

While finalizing my thesis, I greatly appreciated the discussion and comments from Flavius. and I would like to thank you for your help. As a second assessor of my thesis, your critical view helped me to dot the i's and cross the t's of this work.

Jeroen, thank you for giving me the opportunity of this internship within the Forensics department of PwC. You gave me a general direction for me thesis, even though eventually I did venture another way. Your sharp commentary helped me writing a successful thesis. I am grateful that you pushed me to use (and learn) Python. Marvin, thank you for the nice discussions about machine learning and NLP. Your critical notes, although we did not always agree, were appreciated a lot!

I would also like to thank all the other colleagues at Forensics PwC, some of whom had no idea what I was actually doing, but still made my internship at PwC very enjoyable.

Looking forward to keeping in touch with all of you.

Ondanks dat jullie inhoudelijk misschien niet net zo betrokken waren bij mijn scriptie als anderen, Pap, Mam, dankjulliewel voor jullie onvoorwaardelijke steun. Ik was mijn studietijd niet zo makkelijk doorgekomen zonder jullie advies, aanmoediging en hulp. Ik ben ontzettend dankbaar met jullie als ouders.

Daniël

Contents

1 Introduction

Under civil procedure and investigation rules litigants must, when requested by a court, produce documents that are reasonably relevant to the issue and facts of the matter. Traditionally, this process, called “discovery” was performed by consultants or attorneys which review boxes of paper by hand. Nowadays, this information has become digital. This gives rise to the concept of electronic discovery (e-discovery).

Lawsuits and regulatory investigations are common in today’s society and due to fast growing numbers of digital communication, costs of e-discovery are spiraling out of control. The costs of document review can amount to as much as 25 percent of all costs (Gruner, 2008; Roitblat et al., 2010) and for the large-volume e-discovery cases it is even reported that these costs can rise to 70 percent of all production costs (Nicholas M. Pace, 2012). Currently, typical cases involve hundred of thousands or even millions of electronic documents needing to be accurately searched for relevant documents. Because of the large corpora that have to be searched through, e-discovery has grown to become a multi-billion dollar industry.

1.1 Text classification in e-discovery

In the context of e-discovery, good text classification methods are highly sought after. Using an algorithm to automatically label a part of the corpus would highly reduce the associated costs. The long term goal in e-discovery is to have a system that accurately and quickly classifies documents without the need of human intervention.

There are two main properties that set document classification in e-discovery apart form other document classification tasks. In e-discovery, each classification task is unique. This is in contrast to for example spam filtering (the definition of spam does not change) or labeling movie reviews as positive or negative. However, a document which is relevant to one e-discovery case, might very well be irrelevant to another. Therefore, for every new case, a fraction of the documents of the corpus should be labeled by a human to be available for training the algorithm. After a small training phase has been completed, the classifier should label all unreviewed documents.

In other text classification tasks, the problem is treated solely as a supervised learning problem. Features of a text can be selected using already provided class labels and the (machine learning) algorithm can be trained on a part of the already labeled data, which was available from old use cases. In a real life e-discovery case, labels are scarce and should be gathered one-by-one, which is a costly process. For this reason, documents that should be labeled are carefully picked, using a semi-supervised algorithm. The field which optimizes the documents to be labeled query for humans is called *active learning*. The original idea for this paper was to research different kind of active learning strategies. The optimization of active learning in the context of e-discovery requires a data set with not only training examples, but also a human difficulty level per document. Since each e-discovery case has new guidelines regarding what documents are considered relevant or irrelevant, humans are prone to make errors in classifying documents, especially if the document can have multiple interpretations. These human errors (and the reduction of these as reviewers get accustomed with the case) should somehow be taken into account while optimizing active learning algorithm for an e-discovery case. Unfortunately, data regarding the interactions between human labeling and text classification algorithms is hard to find. Because of this, active learning is no longer the focus of this thesis, but some assumptions throughout this thesis are made with this research field in mind.

The second thing that sets e-discovery apart from other text classification tasks is the fact that it is often court ordered, which means that very strict guidelines have to be followed. Because of this, a high reliability has to be proven. As a consequence, an algorithm that accurately classifies documents on itself is not enough: it has to be shown that theoretical foundation of the machine learning classifier is solid.

For this reason, the use of machine learning techniques for e-discovery is not yet allowed in court by the majority of judges even though some promising text classification algorithms are available.

1.2 Downsides of machine learning

Since automatic text classification is desirable, research which investigates the theoretical foundation of machine learning algorithms is in demand. The first requirement for investigating a machine learning algorithm used in an e-discovery case is an available data set and outcome of the algorithm. Unfortunately, e-discovery data is generally very privacy sensitive. The data used in e-discovery cases contains personal e-mails of employees and perhaps even information regarding customers or clients. Because of this, data regarding specific e-discovery cases was unavailable, and an alternative approach for this thesis had to be found. This was not a huge problem as the interpretability of machine learning algorithms is an issue that not solely appears in the field of e-discovery.

As machine learning is gaining more applications in all different kinds of aspects of every day life, not only courts are asking for the inner workings of models, but also regulatory bodies are questioning its use in every day tasks. If we are unable to verify how an algorithm works and on what it bases its decisions, it is hard to convince others that it is functioning as expected. The underlying question in this all is: *should we allow computers to make decisions with algorithms we do not fully understand?* For example see Stokmans and van Lonkhuyzen (2018). This principle is especially true if we can only inspect the input and output, and not the model itself. Using paranormal humans instead of computers, P.H. Dick already posed this question (with regards to predicting felonies) in 1956 with his successful book *The Minority Report*.

The issue is accelerated by the fact that academic literature regarding machine learning has the tendency to be very goal oriented. If a higher accuracy rate is achieved, or if the algorithm is a little faster, the result is publishable. This appears even more true for papers and algorithms that are published by large technology companies such as Facebook, Google or Microsoft. See for example Bojanowski et al. (2016); Mitra et al. (2016). Theoretical analyses of text classification algorithms are lagging behind, simply because the results are considered to be more interesting than the underlying mathematical model. In this thesis, an attempt is made to gain more insight into the inner workings of some algorithms used in text classification.

As stated earlier: e-discovery data is difficult to obtain. Accordingly, other data sets are used. The first data set investigated is a subset of the Reuters-21578 data set called $R52$. This data set contains news articles from Reuters, with a corresponding label. Besides the $R52$ data set, we investigate questions from the popular Q&A forum `www.stackexchange.com`, regarding the topics computer games and the English language. Each of the questions is categorized with one or multiple labels. A full description of these data sets is given in Section 3. We classify the documents of these data sets with a few different kind of text classification algorithms. In this thesis, we look into the performance of these different classifiers, as well as the human interpretability of their performance: *can we explain why a certain label was given to a document?*

1.3 Relation to e-discovery

This thesis was written in combination with an internship at PwC, which, among other services, provides e-discovery support. Hence, a connection with e-discovery in this thesis was desired. For reasons stated earlier, investigating and optimizing an actual e-discovery case was not possible. Therefore, in this thesis it is also attempted to suggest possible improvements for the current work-around regarding the limitations of machine learning in e-discovery.

Because of the 'black box' issue of machine learning algorithms, in an actual e-discovery case, a large list of positive and negative keywords is conceived to reduce the size of the to be investigated corpus. This has its obvious drawbacks of being not specific enough and/or being too specific. Commercial software that assists in the keyword generation is available, but extensive descriptions of the relevant algorithms is unavailable. Generally, domain experts use their knowledge to produce an initial list of keywords. Consequently, an algorithm, which uses semantic and dictionary knowledge, devises alternatives that could also be included in the keyword list.

Therefore, in addition to investigating the veracity and explainability of the methods of our interest, we would like to investigate if they can be used right away to help in the keyword generation process that is currently used in e-discovery cases.

1.4 Process of text classification

Text classification is not straightforward, and there are many steps that can be adjusted to optimize the chosen algorithm. Broadly speaking, there are multiple steps that need to be performed to get a working text classification algorithm. The first step is pre-processing, which often consists of removing common occurring words (stop words such as *a, of, the*), stemming of words (which leaves just the stem, *affecting* $\rightarrow$ *affect*), lemmatization (grouping together inflected forms of a word using dictionary knowledge, *better* $\rightarrow$ *good*) and replacing or removing characters, or words which are deemed to carry no extra information such as interpunction or numbers. Depending on the application, steps can be omitted or altered. For example, in machine translations stemming most likely does not increase performance, as the word *affecting* should be translated differently than *affects*. The goal of pre-processing is to reduce noise in the data and to improve the performance of your model.

The next step is representing the (pre-processed) text as vectors, resulting in a regular matrix for your entire corpus (often called vectorization). This is called feature representation. In general, the feature representation is a function (or method) which maps the textual data to a vector of numbers. The vectorization of text is required as classifiers solely work on numerical data.

The last step in text classification is classifying the data matrix, which can be performed by many different kind of classification algorithms.

1.5 Research questions

Two different feature representation methods are Bag-of-Words (BoW) and Bag-of-Concepts (BoC). BoW is a naive way of vectorizing texts, as it simply constructs a list of the vocabulary, and counts the occurrence frequency of each word in all documents. Therefore, it is easy for humans to understand and in literature it is often used as a benchmark method. Besides the plain BoW approach, a transformation called Term Frequency-Document Inverse Frequency (TFIDF), a class of weightings schemes, is often applied on BoW as it has been shown that they improve the performance (Rennie et al., 2003).

Bag-of-Concepts (BoC) is conceptually closely related to BoW, but theoretically more sophisticated. BoC makes use of `word2vec`, a small neural network that embeds words in a relatively low dimensional space. The word vectors resulting from this embedding can be used to measure similarity between words. These word vectors can be clustered, and each cluster can be considered as concept: since the word vectors are nearby each other in the embedding space, they must be (semantically) similar. Per document, the occurrences of these clusters (concepts) are counted and these counts are used as the number representation of documents for BoC.

This leads to the following research question:

How does Bag-of-Words (with different TFIDF weightings) compare to Bag-of-Concepts?

This question is investigated by answering the following subquestions:

1. How do the Bag-of-Words (with different TFIDF weightings) and Bag-of-Concepts feature representations compare in performance?
2. How comprehensible are these methods for humans (or: how well can the results be explained)?
3. Can Bag-of-Concepts be used as a tool in the keywords generation process for an e-discovery case?

The motivation behind the first subquestion is straightforward. Since we are comparing the methods BoW and BoC, their performance should of course be compared. The second subquestion examines the 'black box' aspect of a model. How well can a human observer follow the decisions of the algorithm? It is expected that the BoW model should be easy to interpret, as it is just a collection of words. Texts that are vectorized by BoC, are represented by cluster counts, where each cluster has a conceptual meaning. For this reason, it is expected that these representations are harder to understand, but still meaningful to humans. The last subquestion attempts to improve current e-discovery practices. More precisely, we want to investigate if the concepts that are made by BoC can be useful in the keyword generation process.

1.6 Approach

To investigate these questions, we make use of the three steps in text classification which were described earlier. First, we pre-process three different data sets (Reuters, StackExchange Gaming and StackExchange English), the exact procedure is described in Section 4. Next, we vectorize the textual data using BoW and BoC. After which we apply different weighting schemes to the BoW representation. To make use of the BoC methods, we first have to generate word vectors and cluster them. The theoretical foundation of these methods are explained in Section 2. After making the BoW and BoC feature representations, the data matrices are classified using naive Bayes, and the model is investigated. Naive Bayes is used in this thesis as this method is commonly used as a benchmark. Moreover it is a fast and simple model and is well understood. This all is summarized in a general overview in Figure 1.

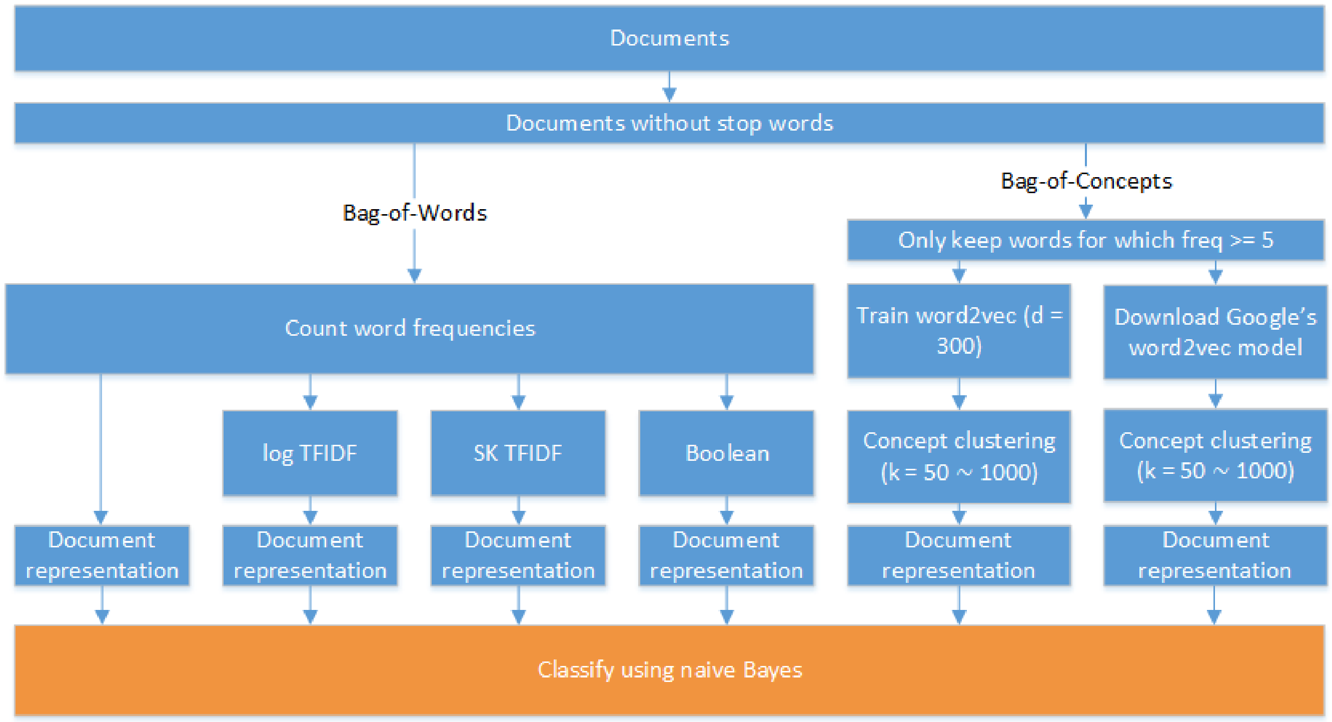


Figure 1: Global overview of the different kind of feature representations that are investigated in this thesis.

2 Theoretical Background

In this section we elaborate on the theory related to the models we use to classify text. We explain Bag-of-Words and possible feature selection methods and the two TFIDF weighting schemes which will be applied. Furthermore, a thorough, yet not complete, explanation of the `word2vec` algorithm will be given as well as explaining the subsequent clustering process that results in the Bag-of-Concepts feature representation.

2.1 Feature selection and Bag-of-Words

Textual data cannot be directly used as input for machine learning classifiers. Therefore, the text first has to be vectorized. The Bag-of-Words (BoW) approach is perhaps the easiest method. In this model every single word is used as a feature and the vectorized document is simply the frequency of all words (term frequencies). The number of different words in a corpus can be huge (easily above 25000) which typically leads to sparse document vectors. Therefore, in almost all literature a number of attributes m are selected, with m ranging in size often between 50 and 5000 (Androutsopoulos et al., 2000; Metsis et al., 2006). The selection of which attributes should be used is not straightforward. Using the m most occurring words in a corpus does most likely not provide the most discriminatory power between the classes.

Different methods to select the optimal features have been proposed. Sarkar (2013) did an empirical study that compares four feature selection metrics with different classifiers, namely Information Gain, Mutual Information, Chi-squared and Symmetrical Uncertainty. Unfortunately, all of these common occurring metrics are supervised methods, meaning they need pre-labeled data points to work with. In the context of this research (e-discovery), labeled data is not readily available and therefore supervised feature selection is not possible. Unsupervised feature selection methods are discussed in numerous papers, but due to the absence of meaningful label information their performance is inferior to supervised feature selection methods. It is suspected by the author of this thesis that for this reason, they are not used as widely in academic literature. As the scope of this research is limited, no feature selection is performed and all words (resulting from the pre-processing step) are used as attributes. This leads to our Bag-of-Words with dimensions V by n, where V is the size of our vocabulary, and n is the number of documents.

2.2 Term-Frequency Inverse Document Frequency

Counting all words gives us the Bag-of-Words feature representation and although this simple method has proven its usefulness, improvements on the BoW have been widely suggested (Rennie et al., 2003; Kosmopoulos et al., 2008; Georgala et al., 2014). A consequence of counting all words is that there are large differences in the frequency counts, as some words are common whilst others are rare. Intuitively, a word that is common in the corpus is likely to appear in all classes and is therefore not a good discriminator. Therefore, these words should be given less weight than the ones that occur rarely. Moreover, frequent occurring words could have some random variation in their class distribution, which could cause fictitious (significant) correlations. On top of that, due to their high frequency they can (even with low weights) strongly influence the class label. A measure that implements a solution to this intuition in an heuristic is *Inverse-Document Frequency* (IDF). When this weighting (which has an inverse relationship with the frequency) is multiplied by (a transformation of) the Term Frequency (TF), we obtain TFIDF.

There are different kind of weighting schemes that belong to the class of TFIDFs. All of these have in common that the IDF portion decreases with increased frequency of the word throughout the corpus, and the TF portion increases with the frequency of the word in a document. In this thesis, we test two different kind of TFIDF's. The first TFIDF is suggested by Kosmopoulos et al. (2008), and has a logarithmic term frequency. Therefore, we denote it by log TFIDF. The second weighting scheme is the TFIDF as implemented by the python package `scikit-learn` which is denoted as SK-TFIDF.

Kosmopoulos et al. (2008) has tested multiple variants of TFIDF and described the log TFIDF version as the best performing one. For this reason, log TFIDF is used in this thesis. The SK-TFIDF variant has been chosen as an alternative for comparison. Since `scikit-learn` is a popular machine learning package for python, it is likely that their TFIDF transformation is widely used. Therefore, it is suited as a benchmark comparison.

2.2.1 log TFIDF

Log TFIDF weighting scheme incorporates three transformations from the straightforward term frequency of words in a document (the pure BoW approach). For details and a more thorough explanation see Rennie et al. (2003). The first transformation compensates for the fact that multinomial naïve Bayes uses a multinomial model to describe the text, which is inaccurate in reality (this is further explained in Section 2.6). Term frequencies have a heavier tail than a multinomial distribution; it has been found that a relatively simple log-transformation can compensate for this. The second transformation accounts for words which occur a lot in both classes. It should be prevented that these words dominate the classification model (for the reasons stated in Section 2.2). To overcome this, the IDF transformation has been proven to be useful. Lastly, document length should be taken into account, as otherwise longer documents will dominate the class preferences. If a word occurs in a document, it is likely to occur again in the same document. As this effect is of greater magnitude in longer documents, normalization should take place to prevent biasing the parameter estimation of the multinomial distribution.

For the mathematical notation of the transformation, some notation is introduced. A vector $\langle x_{1j}, x_{2j}, ..., x_{mj} \rangle$ represents a document j. Here m is the number of attributes. Each scalar x_{ij} is initially the term frequency, that is, the frequency of attribute t_i in document j. Moreover, δ_{ik} is the boolean version of this. That is, 1 if $x_{ij} \geq 1$ and 0 otherwise.

$$\text{(i)}\ \hat{x}_{ij} \Leftarrow \log\big(x_{ij} + 1\big) \text{ (term frequency transformation)}$$

$$\text{(ii)}\ \hat{x}_{ij} \Leftarrow \hat{x}_{ij} \cdot \log\left(\frac{\sum_k 1}{\sum_k \delta_{ik}}\right) \text{ (inverse document transformation)}$$

$$\text{(iii)}\ \hat{x}_{ij} \Leftarrow \frac{\hat{x}_{ij}}{\sqrt{\sum_{i=1}^{m} (x_{ij})^2}} \text{ (document length transformation)}$$

$\hat{x}_{ij}$ is the first TFIDF statistic that is used in this thesis (Kosmopoulos et al., 2008), and is referred to as log TFIDF (due to the log transformation of the term frequencies).

2.2.2 SK-TFIDF

The log TFIDF variant is compared with the default TFIDF transformation as provided by the python `scikit-learn` package. We now describe the transformations done using the default parameters, using the same notation as above.

$$\text{(i)}\ \hat{x}_{ij} \Leftarrow x_{ij} \cdot (1 + log(\frac{1 + \sum_k 1}{1 + \sum_k \delta_{ik}})) \text{ (inverse document transformation))}$$

$$\text{(ii)}\ \hat{x}_{ij} \Leftarrow \frac{\hat{x}_{ij}}{\sqrt{\sum_{i=1}^{m} (x_{ij})^2}} \text{ (document length transformation)}$$

By comparing these equations to the previous ones, one can easily recognize that the TFIDF of `scikit-learn` has no log term frequency transformation and adds a one to the denominator and nominator of the IDF.

2.3 Word2vec

A downside of BoW is a huge feature space, as each word has its own dimension. This can cause computational difficulties. In machine learning applications, reducing the dimensions of your feature space without the loss of too much information, is desirable. One way to reduce the dimensions for BoW is feature selection. This however loses all information of words that are not selected as features. Another method that can reduce the dimensions was introduced by Kim et al. (2017). Instead of counting words, 'concepts' in a document should be counted. This is the foundation of the Bag-of-Concepts method.

In literature regarding information retrival, concepts are often made using domain onthology. These concepts, in combination with TFIDF (or Concept-Frequency IDF) are subsequently used to represent documents (IJntema et al., 2010; Goossen et al., 2011). Kim et al. (2017) propose to make the concepts in a different way. In their paper, the concepts are made by all words of a corpus in a small neural network. Once this neural network is trained on a certain objective, each word has an embedded representation in $\mathbb{R}^n$, with n often in the order of magnitude of a few hundred. Each embedded word representation can be interpreted as a word vector which gives rise to the name of this embedding method: `word2vec`. These word vectors can be clustered using k-means, after which each cluster is considered a concept.

A summary of this process is shown in Figure 2. In the next subsections, we first explain how `word2vec` embeds the words to make word vectors out of a corpus and subsequently how the clustering process works. Since the clusters of BoC consist of word vectors, it is important to understand how these word vectors are constructed. Therefore, we first explain the theory behind `word2vec`, before giving the details of the clustering methods.

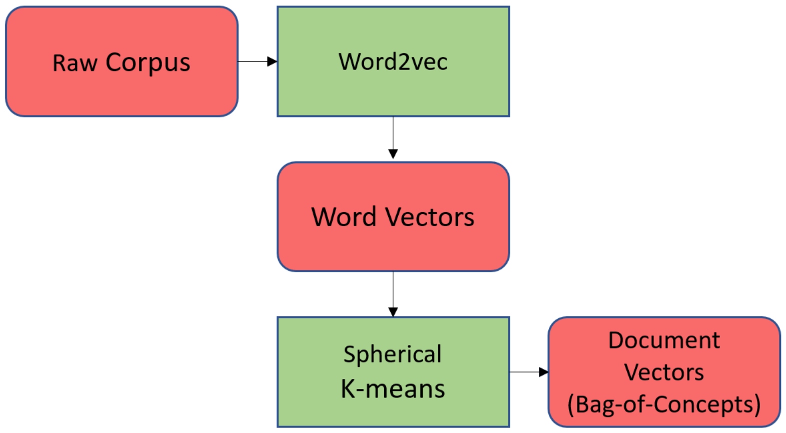


Figure 2: Overview of Bag-of-Concepts.

Mikolov et al. (2013a) introduced the `word2vec` algorithm. Instead of treating a word as an atomic unit, words are described with vectors based on their context. This relies on the distributed hypothesis (Harris, 1954), which states that words in a similar context have a similar meaning. `Word2vec` uses this assumption and predicts the (continuous) context of a word, which is then used as a vector representation of the word. The idea is shown Figure 3.

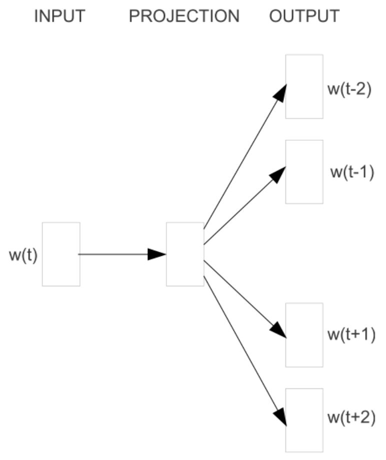


Figure 3: `Word2vec` predicts the context of the word using a Skip-Gram model. A context window size of 2 is used in this figure. Figure from Mikolov et al. (2013a).

Surprisingly, Mikolov et al. (2013a) found that the `word2vec` algorithm can be used for more than just constructing dense document vectors. They show that after training on a large corpus, they can capture semantic relations between two words accurately. They give as example that the word vector that is closest to `word2vec(`*biggest*`)` - `word2vec(`*big*`)` + `word2vec(`*small*`)` is `word2vec(`*smallest*`)`. In other words, they show that two pairs of words with the same semantic relationship can be identified using simple algebraic operations on the word vectors. If all the word vectors of a document are simply averaged out, a document vector can be created. It has been shown that this document vector performs well when used to determine similarity between documents (Kim et al., 2017). A large downside of this document vector is that all human interpretation is lost, which is not the case for the Bag-of-Words model.

2.4 Word vectors

Even though the `word2vec` algorithm employs a neural network, it is not considered to be a deep learning technique. As it only uses a single hidden layer, the model is surprisingly simple. The `word2vec` algorithm has two main models: the Skip-Gram model as well as the continuous bag of words (CBOW) model. The Skip-Gram model attempts to predict the context given a single word, while the CBOW model predicts a single word using the context as input (i.e. reverse the in- and output in Figure 3). Mikolov (2013) elaborates that they are comparable given enough training data. Explicitly he mentions the following advantage and disadvantage of Skip-Gram vs CBOW:
"Skip-Gram: works well with small amount of the training data, represents well even rare words or phrases"
"CBOW: several times faster to train than the Skip-Gram, slightly better accuracy for the frequent words"

Furthermore, as best practice he recommends *"to try few experiments and see what works the best for you, as different applications have different requirements"* (Mikolov, 2013). Since the scope of this thesis is limited, we have chosen to only investigate the Skip-Gram model and the CBOW version is not explained any further.

Similar to many other unsupervised methods that attempt to learn word representations, co-occurrences of words are used as a source of the information of the model. A possible way to devise a word vector given a certain corpus, is to use a count based method. Count based methods, like `GloVe` (Global Vectors, Pennington et al. (2014)), simply count the occurrence of words nearby and store this information in a co-occurrence matrix, which is then decomposed. This window size of the context is a hyperparameter that should be tuned, but a size of 5 is typical. Using this value, the 5 words in front of the word of interest as well as 5 words behind it are seen as co-occurring with the center word (for a total of 10 words). As offset or distance is not taken into account. Therefore, words that solely occur in pairs (think of 'Angeles' which is always preceded by 'Los' in a certain corpus) do not have a probability of one of occurring together, as other words in the vicinity of 'Angeles' are also in the context of 'Los'.

Similar to `GloVe`, `word2vec` uses the co-occurrence of words as basis of the model, but attempts to predict the co-occurrence matrix instead decomposing a certain transformation of it using a weighted least squares. Pennington et al. (2014) claims that this result in the exact same weights as `word2vec` (Řehůřek, 2014). The discussion of `GloVe` versus `word2vec`, albeit interesting, is outside the scope of this thesis. I refer interested readers to Řehůřek (2014).

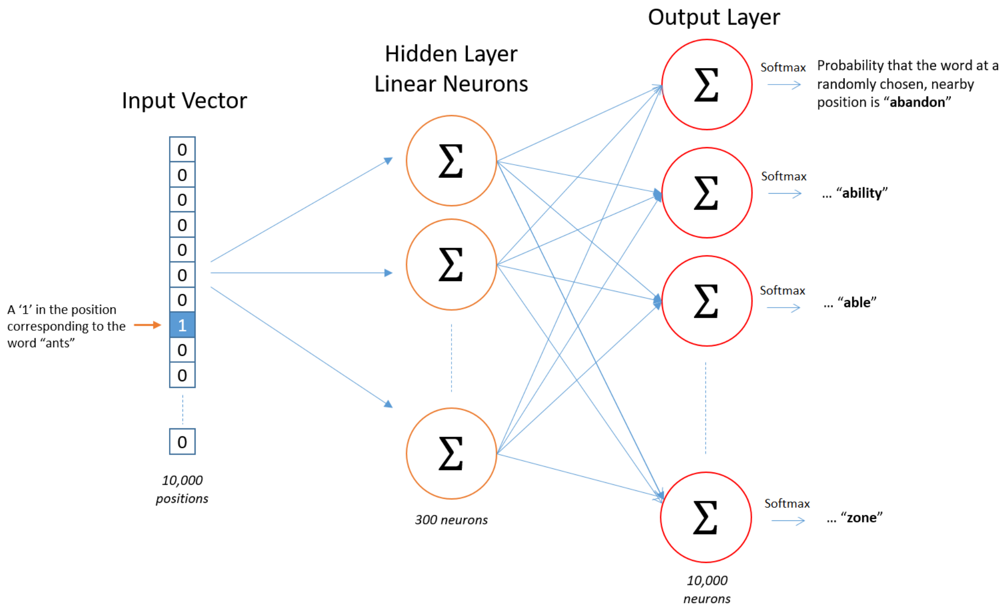


Figure 4: `Word2vec` predicts the context of the word using a Skip-Gram model. The context is predicted using a hidden layer. Figure from McCormick (2016a).

A global overview of the `word2vec` Skip-Gram architecture is shown in Figure 4. The hidden layer are the encoded center word vectors, while the output layer can be seen as the encoded output word vectors. Basically, the model has to learn two vector representations for each word. One as a center word and one as context word. This is because it assumes that the center word and context words come from different vocabularies, as the same word obtains different weighting for its two appearances (one as center word and one as context word). This assumption is not explicitly motivated (Goldberg and Levy, 2014), but is further discussed in Section 2.4.6.

Unlike other neural networks, the training objective is not actually what the model is used for. The neural network is trained solely to learn the weights of the hidden layer and is in the end not used to predict the context words (even though that is the training objective). These hidden layer weights are actually the word vectors that the model is supposed to learn. Therefore, once the model is trained, the hidden layer can be saved and the rest of the model can be discarded.

2.4.1 Training objective of word2vec

The objective of `word2vec` is to predict the surrounding words of a certain input (center) word. It does so by maximizing the probability of all context words, given the current center word. The objective function of the model is stated as following:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{-5 \leq j \leq 5, j \neq 0} p(w_{t+j}|w_t;\theta) \Rightarrow J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-5 \leq j \leq 5, j \neq 0} \log\big(p(w_{t+j}|w_t;\theta)\big) \quad (1)$$

Here T is the total count of words in our corpus, w_t is our center word and w_{t+j} is the jth word after the center word. θ represent all the variables that are optimized. In other words, given a set of parameters θ, the likelihood of all 10 context words w_{t+j} conditional a center word w_c should be as high as possible. Before any training the model is initialized randomly (θ is set to reasonable but random values). The way in which θ is used in the model can be seen in Figure 5, which is an enlarged but more detailed version of Figure 4.

The hidden and output layer are here shown explicitly in matrix form. The elements of those layers (matrices) are the weights of the network. The goal of the model is to learn the values of these two matrices. These are the unknown θ values ($|\theta| = 2Tn$, with T vocabulary size and n number of embedding dimensions). Once these values are known, the embedded word vectors can simply be retrieved from these parameters.

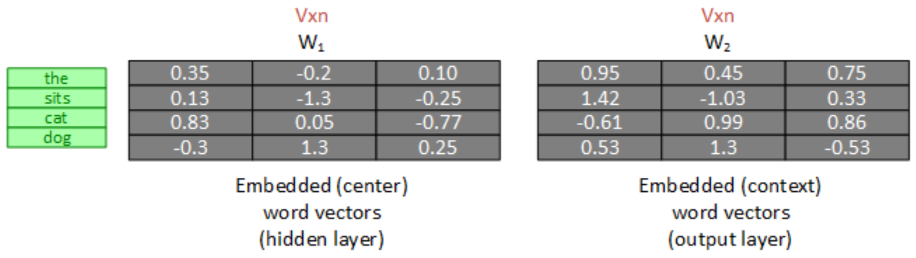


Figure 5: Hidden and output layer explicitly written in matrix form, for a vocabulary of four words. The matrices W_1 respectively W_2 are the hidden respectively output layers ('neurons') as shown in Figure 4.

The input of the model is w_t, this is a one hot vector of the center word. That is, a one in the position that corresponds to the word and zero otherwise. This vector is multiplied with the weights in W_1. Each row of W_1 contains a respective embedded center word vector. As w_t is a one hot vector (a single one, the rest all zeros), this multiplication can be seen as a look-up in matrix W_1 and results in vector v_c, the encoded word vector that corresponds to the input (center) word. The size of this vector, in Figure 4 shown as number of neurons, is a parameter that should be tuned. As the resulting word vector (`word2vec(`*word*`)`) is actually this row in W_1, it should be noted that this dimensionality is also the dimensionality of the resulting word vector. In literature, values between 100 and 1000 are common (Mikolov et al., 2013a; Kim et al., 2017; Socher, 2015). To obtain context word probabilities, this encoded word vector is multiplied with matrix W_2. Matrix W_2 is the embedded output matrix: again each row can be interpreted as a word vector, but this time for the context word, in probability form: $p(context\ word|center\ word)$. After performing the operation $W_2 v_c$, we get an output score for each word in vocabulary V. When the softmax function is applied, the predicted context word probabilities are produced.

2.4.2 Exponential normalization: Softmax

There are two matrices with values that can be optimized to maximize the probability of a context word. To understand the optimization technique, one has to see that each row in W_2 corresponds to a specific word in the vocabulary. In Figure 5, we have four of these rows which we denote as u_o, which can be interpret as the embedded output word vector. In other words, we can say that u_o is the vector associated with the output (or context) word o. Similarly, in matrix W_1 there are four rows, denoted as v_c, that corresponds to the center word c (which, once again, in the end will be the word vector as produced by `word2vec`). The dot product of these two vectors, $u_o' v_c$, gives a loose measure of similarity.

Given this measure of similarity, we can now attempt to express the probability for an output word o given the center word c: $p(o|c)$. As we have an embedding for both output word o as well as center word c, we can use their dot product. Since we are calculating probabilities, the sum of all possible output words given a center word should be equal to one. We could simply normalize the output scores (dot products), but that would be invariant to scale differences. As the absolute difference between the output scores increases, so should the probabilities. Therefore, exponential normalization (softmax) is used, as is common in neural networks. With this in mind, $p(o|c)$ can be expressed as $\frac{exp(u_o' v_c)}{\sum_{w=1}^{n} exp(u_w' v_c)}$. This formula is the mathematical operation of softmax. These rows u_w, which combined are the matrix W_2 and v_c, which combined for all c are the matrix W_1 are the parameter θ in Equation 1. Furthermore, the expression for $p(o|c)$ can be substituted in Equation 1 to get an explicit objective function for each center and context word.

2.4.3 The network of word2vec: an example

An example of how `word2vec` calculates probabilities, using the weights from Figure 5 is shown in Figure 6. Here, our vocabulary consists of four words (*the, cat, sits, dog*). Our goal is to find p(*sits* | *cat*). For this, the one hot vector corresponding with the center word *cat* is multiplied with W_1. This results in the third row of W_1 as the embedded word vector. This would also be the `word2vec` output from `word2vec(`*cat*`)` if training was completed, and is denoted in the paragraph above as v_c. This vector is multiplied with W_2. If we take the second row of W_2, and multiply it with v_c,

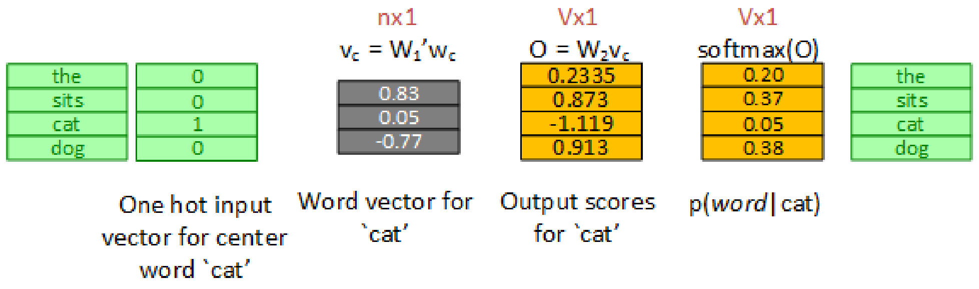


Figure 6: Extracting the probabilities of the `word2vec` architecture.

we get 0.873. As the third row in W_2 corresponds with the word *sits*, this is the 'score' for p(*sits* | *cat*). To get the actual posterior distribution of words, softmax normalization is performed over the vector containing all the scores, denoted as O in Figure 6. This results in a probability of 0.37 for p(*sits* | *cat*), as shown in Softmax(O) in Figure 6.

If we have a corpus consisting of one sentence: *"the cat sits"*, Equation 1 attempts to maximize the probabilities for *the* and *sits* given the center word *cat*. Therefore, the probabilities as shown in Figure 6 for *the* and *sits* would increase during training, while the probability *dog* would decrease. If we had another sentence *"the dog sits"*, the same would happen, but now for the probabilities associated with *dog* instead of *cat*. This results in similar word vectors for *cat* and *dog*, as is expected since these words are closely related and are likely to occur in similar contexts.

2.4.4 An example for calculating the probability

To train the neural network, training instances have to be generated. The context of a word should be predicted giving some trainings examples. For a sentence *"The cat sat on the table"* with a context window of one, six combinations can be formed. As if a word occurs at the start or end of sentence, half of the context is cut off, e.g. the context for the word *table* consists of only the word *the*. An example of a training instance is *"the cat sat"*. The objective is to maximize the probability of the words occurring in each others neighborhood (Equation 1).

Using the same notation as earlier, we calculate this probability using $\frac{exp(u_o' v_c)}{\sum_{w=1}^{n} exp(u_w' v_c)}$. Substituting the words in J′(θ) in Equation 1 we get

$$p(the|cat)p(sits|cat) = \frac{exp(u_{the}' v_{cat})}{\sum_{w=1}^{n} exp(u_w' v_{cat})} \cdot \frac{exp(u_{sat}' v_{cat})}{\sum_{w=1}^{n} exp(u_w' v_{cat})} \tag{2}$$

Since $\max y$ is equivalent to $\max \log y$, maximizing this probability is equal to maximizing J(θ) of Equation 1:

$$u_{the}' v_{cat} + u_{sat}' v_{cat} - 2 * \log\left(\sum_{w=1}^{n} exp(u_w' v_{cat})\right)$$

To obtain a loss fuction in standard format, we multiply this function by minus one and we find the objective function which we want to minimize. Taking the derivative of this with regards to v_{cat} gives us a well defined function for updating W_1, and taking the derivative with regards to u_o (o being a context word such as '*the*') leads to an update rule for W_2 (Rong, 2014).

However, a corpus can easily consists of 10000 unique words and 300 columns for the embedded dimension is not uncommon. Therefore, there are three million weights both in the input (W_1) and output layer (W_2) which needs to be updated. To train this model with a large corpus is not an easy task. Mikolov et al. (2013b) addressed these issues in another paper and came up with two innovations to avoid needing thousand of computing hours to training the network.

His first idea was to sub-sample words that appear frequently. Words that occur many times are also likely to occur in many different types of context. A word like "the" does not tell much about the meaning of another word (in our example, "Table"). Moreover, there are many more training instances containing the word "the" than needed to learn a good vector for the word. Therefore, frequent words are sampled less than you would proportionally expect.

It does this by iterating over all words w_i in the corpus. The word w_i is removed with a probability $p(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$. t is a chosen as a certain threshold, typically around 10^{-5} and $f(w_i)$ is the frequency of w_i (Mikolov et al., 2013b). This probability distribution was chosen as a heuristic, it was found to work well in practice (Mikolov et al., 2013b). This probability of removing words reduces the size of the corpus and therefore also effectively increases the window size of other words, as words which were present in the original sentence are now 'deleted' and the next nearing neighbor is seen as context.

2.4.5 Negative Sampling

Another adjustment to improve the training speed is (Skip-Gram) Negative Sampling, which is a form of Noise-Contrastive Estimation.

As stated earlier, there are 6 million weights which need to be updated (using the example with 10000 unique words by embedded dimension space of 300 from earlier). Three million of those are in the embedded center word vectors (the hidden layer) and 3 million of those are in the embedded context word vectors (the output layer). For every training instance, the predicted context vector (the outcome of p(*words* | *context word*)) should closely resemble *words*. Using the example from earlier, Equation 2 should return a high probability for the words *the* and *sat*, while equaling zero (or a low probability) for all other words. This results in problematic updating, as all three million weights in the output matrix (W_2) should be adjusted.

The hidden layer is more manageable as only the row which corresponds to the one hot input vector (corresponding to the word *cat* in the example) is relevant for the weights of this training instance. This can be concluded directly from the fact that $u_o'v_c$ only depends on v_c but needs to be calculated for every output word o, as is needed in the denominator of the softmax normalization.

To reduce the number of weights that need to be updated every iteration, Mikolov et al. (2013b) introduced negative sampling. Instead of updating the entire matrix W_2, only a small number (as an example they give 20) of rows are updated. The negative rows (negative means words that are not part of the training instance) which are updated are randomly sampled from the empirical (unigram) distribution raised to the power of $\frac{3}{4}$. This distribution is empirically chosen by Mikolov et al. (2013b). Thus at each iteration, the hidden layer corresponding to the center word on top of a few rows in the output layer which correspond to words which were not the training pair were updated. Moreover, the row in the output layer which corresponds to the context word in the training pair is also updated (McCormick, 2016b).

This is only an intuition for Negative Sampling. In reality, Negative Sampling uses an objective function which differs from Equation 1, which specifically maximises the probability that an training instance came from the corpus. There are numerous sources that do a good job of explaining the mathematics behind it, but this is outside the scope of this thesis. Suggestions for the information regarding the mathematical formulations are Goldberg and Levy (2014) or Rong (2014).

2.4.6 Using the center word or the context embedding

So far, the training objective of `word2vec` has been discussed. The actual goal of `word2vec` is to make (dense) word vectors. Since the dimension of the input size is the same as the dimension of the output size, dimension reduction is not achieved. Moreover, the output vector is still likely to be sparse: given a single input word, many more words are contextually unrelated than related. However, the context of a word is also embedded in the embedding matrix (W_1 in Figure 5). As W_2 is independent of the input, the multiplication of W_2 with embedded word vectors results in a probability for all words. Due to the earlier described distributed hypothesis, words that appear in the same context should have similar output probabilities. Therefore, their embedded vector (which when multiplied with W_2 produce the posterior probabilities) should be similar. These embedded vectors are the eventual output vectors of the `word2vec(`*word*`)` algorithm. In other words, `word2vec(`*word*`)` is the row that corresponds to *word* in W_1, or equivalent `word2vec(`*word*`)` = v_{word} after the training phase has been completed.

It should be noted that the vectors relating to the output layer in W_2 could also be used as word vectors. However, it seems that Mikolov et al. (2013a) without further motivation uses W_1 for the embedded word vectors, disregarding the possibility of using the embedded output vectors in W_2. Other contributing research almost always overlooks this assumption as well.

Having seperate embeddings for words as center word (W_1) and context (W_2) assumes that the input and context words come from separate distributions, which on the first glance may look like a surprising assumption. Even though this assumption is implicitly made in the papers from Mikolov, it does allow for minimizing the occurance of a word in its own context: consider the center word *cat* and the context word *cat*. It is possible that the words do not occur frequently in their own context, and therefore the probability p(*cat* | *cat*) should be minimized. If there is only one vector representing *cat*, the dot product on itself should be minimized, which is not possible. Having separate embeddings for words as center word and context word does allow for assigning a low probability Goldberg and Levy (2014).

Recently, a paper by Mitra et al. (2016) appeared which investigated the vectors in W_1 and W_2. His paper looks into the possibility of using W_2 as embedded word vector, or even combining the vector resulting from W_1 with the vector from W_2. They found that the embedded output vector are more reflective of topical similarity and useful for determining document "aboutness". Unfortunately, the scope of this paper was limited and this is not further investigated in this paper.

2.4.7 Tuning the hyperparameters

As shown from the section above, a lot of implicit assumptions are made by Mikolov et al. (2013a) in the development of this `word2vec` algorithm. Even though a general overview of `word2vec` is given by them in their papers, there are numerous things in this model that can be tweaked. Context window size and the number of neurons (dimensions) in the hidden layer spring to mind, but there are many more nuances that should be taken into account when employing this algorithm. For example, the power of `word2vec` is often shown by demonstrating that `word2vec(`*king*`)` - `word2vec(`*man*`)` + `word2vec(`*woman*`)` $\approx$ `word2vec(`*queen*`)`. In reality, it could be that the outcompe of this equation is slightly closer to `word2vec(`*woman*`)` than to `word2vec(`*queen*`)`, but this fact is often left out in demonstrations or write ups about `word2vec`.

Moreover, in the C implementation of `word2vec` by Mikolov et al. (2013b), the window size was not fixed. Instead, it was sampled between 1 and maximum window size. This provides an implicit weighting based on the distance between words, as words that are at the border of the context window are less likely to be used as context for a word.

In their implementation, they also added idiomatic phrases as single words in their vocabulary (for example "Los Angeles" as one word instead of two seperate ones). To detect phrases, they first trained the model using only single word. Next, they identified candidate phrases using a data-driven approach, shown in Equation 3, and tested all bigrams that resulted using a set of analogical reasoning tasks on performance (for details see Mikolov et al. (2013b)). The good phrases were kept.

$$\frac{(count(\text{word}_a \text{ following word}_b) - min\_count) * N}{count(\text{word}_a) * count(\text{word}_b)} > threshold \tag{3}$$

In this formula *min_count* is a discounting coefficient and prevents too many phrases which consist of very infrequent words. *threshold* is chosen as the minimum score needed for a bigram to be accepted as a real bigram according to Mikolov et al. (2013b). Mikolov et al. did not provide suggestions for the parameter values, but these are discussed in Section 4.

Moreover, Mikolov et al. did not provide information about the pre-processing of words in their implementation. For this reason, one has to be careful when using the model as trained by them on a part of Google News corpus. Removing stop words and transforming all words of your to-classify-corpus to lower case seems to be standard procedure in text classification, but these steps were apparently not performed by Mikolov et al. (2013b).

Nevertheless, we use the `word2vec` model trained by Mikolov et al. (2013b) model as comparison to the vectors resulting from training the `word2vec` model on our own data sets. The pre-trained model from Mikolov et al. (2013b) was trained on part of Google News data set (about 100 billion words). The model contains 300-dimensional vectors for 3 million words and phrases[1]. On one hand, because the model is trained on this large corpus, the semantic meaning of words should be well embedded in the word vectors. On the other hand, this general definition of the words might be in disagreement with our specific data set. It is possible that the words in our data set have a different context than when they appear in the Google news corpus. It is interesting to compare these results, which is why the pre-trained `word2vec` model is also investigated in this thesis.

[1] `https://code.google.com/archive/p/word2vec/`

2.5 Bag-of-Concepts

Word vectors as obtained by the `word2vec` algorithm are well suited for word representation in a low dimension space. However, as a classifier the word vector representation is not directly suited. It seems far fetched to assume that the word vectors of a document do well in some sort of classifier that separates documents by a (linear) decision boundary. The words are embedded in such a way that words which are similar in semantic meaning are close together in their dimensional space, but this does not necessarily translate into a prediction power for a document label.

Instead, the word vectors in this thesis are clustered with k-means. This results in clustered word vectors, where each cluster is a concept. These concepts in a document can be counted, akin to Bag-of-Words. Kim et al. (2017) has shown that these clusters perform well in comparing document similarity, but no literature that attempts to build a classifier from these clusters has been found by the author of this thesis.

In this research, Bag-of-Concepts possibilities as a feature representation for documents is investigated. Concepts are made by using the word vectors resulting from the `word2vec` algorithm and subsequent clustering of these vectors based on their cosine distance. This metric, instead of the more commonly used Euclidean one, is used because Schakel and Wilson (2015) have shown that the word vector length seems to be unrelated to the word meaning, but is a function of the occurrence frequency of a word. As a consequence, spherical k-means (which uses cosine distance and is insensitive to vector length) is more appropriate than Euclidean.

These clustered word vectors each represent a word concept. As all word vectors are assigned a cluster, each word which occurs in the corpus has a many-to-one relation with a cluster. The number of spherical k-means clusters is predefined and is equivalent to the number of features (or: concepts) that one wishes to obtain. As with all k-means algorithms, a number of random initialization needs to be performed to compensate for the impact of the location of the initial clusters. The implementation of k-means is discussed in Section 4.

2.6 Classification Method: naïve Bayes

After constructing the Bag-of-Words or Bag-of-Concept feature vectors, a machine learning technique has to be applied to classify the vectors. In this research, we use the much tested naïve Bayes (NB) models. These have been studied for over 50 years and have extensively been tested for, among others, the purpose of text classification. Many variants of NB exists, and they all rely on a simplistic independence assumption between the attributes. Yet, they often have a good performance and are therefore used as baseline classifiers which are fast and easy to implement. If the attributes in a text are actually independent, it can easily be proved that NB is optimal, in the sense of minimizing the missclassification rate (Domingos and Pazzani, 1997).

The goal is to calculate $p(l|\vec{x})$ for all classes (labels) l. Here, $\vec{x}$ represents features of a certain document. Text classification can be done multi-class or binary: multi-class predictions aim to correctly the label of a document out of a set of many classes, binary classification only knows positive or negative labels (does the document belong to the class, yes or no?). As is customary in text classification problems, we test our methods using a binary labeling framework. This means that we compare $p(true|\vec{x})$ with $p(false|\vec{x})$. If $p(true|\vec{x}) \geq p(false|\vec{x})$, the document is labeled as relevant (i.e. belongs to the class), and otherwise the document is labeled as false. It is possible to include missclassification cost, for example if the missclassification costs for a false negatives are lower than those for a false positive, one could label documents as *true* if $p(true|\vec{x}) \geq 0.40$ while $p(true|\vec{x}) < p(false|\vec{x})$ also holds. In this thesis, we did not look into these options any further and assumed the standard *true* label if $p(true|\vec{x}) \geq p(false|\vec{x})$ and else the $false$ label was applied.

We would like to calculate $p(l|\vec{x})$, that is, the probability that a document belongs to class c given an attribute vector $\vec{x}$. From Bayes' theorem it follows that the probability is:

$$p(l|\vec{x}) = \frac{p(\vec{x}|l) \cdot p(l)}{p(\vec{x})}$$

Since $p(l_t)$ (relevant or true) and $p(l_f)$ (irrelevant or false) are simply the proportion of true respectively false documents in the training set, the difficulty lies in calculating $p(\vec{x}|l)$. Because of the (strong) independent assumption it holds that $p(\vec{x}|l) = \prod_{k=1}^{K} p(x_k|l)$.

If $p(l_t|\vec{x})$ is larger than a certain threshold, the text is classified as relevant to the label; otherwise, it is labeled as irrelevant. By varying the threshold, one can increase the recall at the expense of precision or vice versa.

In any real life text classification problem the independence assumption does obviously not hold. A simple counter example would be the co-occurrence of the words *oil* and *barrel*. Nevertheless, NB classifiers are relatively effective in actual use cases, even if this assumption is violated (Domingos and Pazzani, 1997; Rennie et al., 2003). $p(\vec{x})$ is independent of the class label, and NB labels $\vec{x}$ as the class that maximizes $p(l|\vec{x})$ the denominator can safely be ignored in the calculations.

2.6.1 Multinomial NB

The way to estimate $p(x_k|l)$ (and thus $p(\vec{x}|l)$) differs per type of NB. In this thesis, we only discuss multinomial NB with Boolean attributes and (transformed) term frequencies as these variants seem outperform other NB variants in many use cases (Metsis et al., 2006; Rennie et al., 2003; Kosmopoulos et al., 2008).

In the multinomial NB (MNB) model each message is represented as a vector $\vec{x}$. Most often, each attribute in $\vec{x}$ represents a word, punctuation mark or otherwise a characteristic of the text. In the Bag-of-Words approach each attribute is a word, in the Bag-of-Concepts method, each attribute is a concept.

Let F be the set of all characteristics, consisting of m attributes, then $F = \langle t_1, t_2, ..., t_m \rangle$ and let x_i be the frequency of attribute t_i in document d. Using the multinomial NB model, each document d can be seen as the result of picking $|d|$ attributes from F with replacement. The probability for each of the attributes is dependent on the class it belongs to. The result of sampling $|d|$ attributes from F is $\vec{x}$. Therefore, $p(\vec{x}|l)$ follows the multinomial distribution (Metsis et al., 2006; Kosmopoulos et al., 2008):

$$p(\vec{x}|l) = p(doclength = |d|) \cdot |d|! \cdot \prod_{i=1}^{m} \frac{p(t_i|l)^{x_i}}{x_i!}$$

where the common assumption that the document length is independent of the class has been made (Metsis et al., 2006; Schneider, 2004). As it assumed that the document length is independent of the class, $p(doclength = |d|) \cdot |d|!$ can be dropped from the equation. Similarly, $x_i!$ is independent of the class and can also be neglected in the equation. For the same reason, $p(\vec{x})$ can be ignored in the equation for $p(l|\vec{x})$). Because of this, the label that maximizes $p(l) \cdot \prod_{i=1}^{m} p(t_i|l)^{x_i}$ also maximizes $p(l|\vec{x})$.

This leaves us with estimating $p(t_i|l)$, for which a Laplacian prior is used. This prior, instead of the direct proportion, is often used to smooth the probabilities.

$$p(t_i|l) = \frac{1 + N_{t,l}}{|F| + N_l} \tag{4}$$

Here $N_{t,l}$ is the frequency of attribute t in class c, and N_l is the sum of $N_{t,l}$ over all attributes t. The rationale behind this smoothing is the following: even if attribute t never occurs in N_l, there still is a non-zero probability that the document belongs to l as we might just be 'unlucky' in our training sample.

In this thesis, we do not solely use the multinomial NB model with term frequencies as described above. If we replace x_i by its Boolean counterpart, that is 1 if t_i occurs in d and 0 otherwise, it has been found by some that performance increases (Metsis et al., 2006; Kosmopoulos et al., 2008). This may seem counter-intuitive, as the amount of information which is provided to the model is decreased. Schneider (2004) provides an explanation for this: they proved that MNB with term frequency is equivalent to NB model with the attributes each independently following a Poisson distribution, under the assumption that document length $|d|$ is independent of the category (Eyheramendy et al., 2003). An intuition for this can be derived from the standard result in mathematical statistics, which states that independent Poisson variables conditional on their sum follow a multinomial distribution. Therefore, if we assume that the attributes are independently distributed with a Poisson distribution, and document length $|d|$ is fixed (or independent of class), it follows that the attributes follow a multinomial distribution (Eyheramendy et al., 2003).

For this reason, Boolean attributes instead of term frequencies are interesting to investigate. If we use Boolean attributes instead of term frequencies, the estimation of $p(t_i|l)$ in the model is adjusted:

$$p(t_i|l) = \frac{1 + M_{t,l}}{2 + M_l}$$

here $M_{t,l}$ is the frequency of documents in class l that contain attribute t and M_l is the total number of documents.

2.7 $\mathrm{F_1}$-score

To answer our research question, we need a metric which defines performance. For unbalanced classes, as is the case in this thesis, the plain accuracy score is unfit to use as a score metric. If there is a large class imbalance, a high accuracy can be obtained by simply predicting everything as the most occurring class. As an alternative, the $\mathrm{F_1}$-score has been proposed. The $\mathrm{F_1}$-score is defined as the harmonic mean of precision and recall, as shown in the following formula:

$$F1 = 2\frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

where recall is defined as $\frac{TruePositive}{TruePositive+FalseNegative}$, and precision is defined as $\frac{TruePositive}{TruePositive+FalsePositive}$.

Or in interpretation, recall is the fraction of actual *true* documents in are corpus that are successfully labeled as such while precision is the fraction of all documents labeled as *true* that are actually *true*. The $\mathrm{F_1}$-score is used in many different academic literature and is used in this thesis as performance measure.

The use of $\mathrm{F_1}$-score instead of other options as a score metric, such as the precision/recall break-even point or Area Under the ROC-curve, is somewhat arbitrarily made. It is most easy to compare performance using only a single number, and the $\mathrm{F_1}$-score is easy to understand and widely used. Therefore, the $\mathrm{F_1}$-score was chosen throughout this thesis.

3 Data sets

To answer the research questions, we evaluate the BoW and BoC performance on different data sets. In this thesis, three different data sets coming from two type of sources are used. In this section, these three data sets are explained.

3.1 R52

The first data set which was investigated is the R52 data set collection, a subset of the very commonly used Reuters-21578 data set. The Reuters-21578 data set contains 21578 Reuters news articles, which were indexed and categorized by personnel from Reuters. An earliest version of the Reuters data set was released to the public in 1990 and in the years thereafter, many improvements to the data set were made. In 1996 the version which we now know as Reuters-21578 was published. A few years later, the ModApté split was released, which divided the Reuters-21574 set into a training and test set and (among other things) corrected typographical errors in the labels. This pre-specified division made comparison of results between papers easier.

Ana Cardoso-Cachopo (2007) used this ModApté split (originally in XML format) and processed them into a new subset: R52. In this data set, only documents with exactly one label are kept. In the ModApté set, news articles could have none or multiple labels, which could be an (undesired) source of noise in the classification process. Having multiple labels also conflicts with the e-discovery context: a document can only be relevant or irrelevant.

Instead, the R52 data set is used. After filtering out all documents with no or multiple labels, 52 different classes remained (as opposed to the almost 700 in Reuters-21758). Many of these 52 classes are uncommon and since each label separately is treated as a binary classification problem, very unbalanced classes would have to be classified. For this reason, only the four most occurring classes of R52 were used in this thesis. These classes and their proportions are shown in Table 1.

Even though all four classes are in a single data set, each row corresponds to a binary classification task. In other words, in the train data set with 6532 rows, the first classification task is for the class *'earn'* for which 2840 positive labels and 3692 negative instances exists. In the second task, for the label *'acq'*, 1596 positive instances and 5026 negative instances are present in the training set. The documents labeled positive in the class *'earn'* are per definition labeled as negative for the class *'acq'*. The reverse is not true: not all documents that are negative for the class *'earn'* are positive for the class *'acq'*.

Table 1: Distribution of labels of the Reuters data set.

Label name	# train documents	Proportion of train documents	# test documents	Proportion of test documents
earn	2840	0.43	1083	0.42
acq	1596	0.24	696	0.27
crude	253	0.04	121	0.05
trade	251	0.04	75	0.03
Other	1592	0.24	593	0.23
Total	6532	1	2568	1

The documents in the class *'earn'* are related to earnings and earnings forecasts, *'acq'* is for news articles about mergers or acquisitions. *'Crude'* is for documents regarding crude oil and documents labeled with *'trade'* have a relation to (financial) trades.

Furthermore, Ana Cardoso-Cachopo (2007) has already pre-processed the documents. The pre-processing is discussed in Section 4 as the other two data sets undergo the same pre-processing steps.

An example of a (relatively short) document in the R52 data set labeled as *'crude'* is *texaco canada raises crude oil postings canadian cts bbl light sweet now dlrs bbl.*

3.2 StackExchange

As one would like to appraise the results, a different kind of data set is needed. Besides the Reuters data set, another source of data is used. This source of data is the popular Q&A forum www.stackexchange.com. StackExchange has many different kind of boards on which users can ask questions and receive answers regarding different specific topics. Its most known board is Stack Overflow, which is dedicated towards questions for programmers. All questions, answers, comments, users and tags from all boards can readily be downloaded in XML-format from a website[2].

In this thesis, we have chosen two relatively popular boards which in textual composition are comparable with R52. These boards are for questions regarding the English Language, and Gaming.

Each question is labeled with one or more labels by the user submitting the question and the labels can be adjusted by moderators of the forum if necessary. Labels for the Gaming board can be names of games, for example *Minecraft* or *League-of-Legends*. Similar to the R52 data set, only questions with exactly one label are kept. Moreover, we only look at questions and their related labels and do not take answers (called posts or comments) into account.

Identical to the R52 set, we only take the four most common labels in the Gaming and English board, and do not attempt to classify the other posts. The labels for the Gaming data set are straightforward: they are all names of the game that the forum question is about. The four most common tags in the English language are: *'single-word-requests'* in which a single word with a certain meaning is requested in a tip-of-my-tongue way, *'meaning'* which is for questions about the meaning of a word, *'grammar'* for questions regarding grammar in the English language and *'word-choice'* which is assigned to questions regarding the choice of a certain word. A summary of the label distribution of the Gaming and English data set can be seen in Tables 2 and 3.

Below, two (again relatively short) examples from the Gaming and English data set are given.

- Gaming, labeled *minecraft*: *where are my minecraft saves located i want to copy modify delete a minecraft world where can i find my saves folder*
- English, labeled *grammar*: *why is it always on foot not on feet while we normally use both our feet to walk why is it grammatically acceptable to say on foot not on feet* [3]

The difference between the examples of the Reuters and the two StackExchange data sets immediately catches the eye. While the English and Gaming data set are easily readable by a human (although a bit clumsy due to the lack of interpunction), this is not necessarily the case for the Reuters set. The original news message was already hard to interpret, namely: "TEXACO CANADA RAISES CRUDE OIL POSTINGS 24 CANADIAN CTS/BBL, LIGHT SWEET NOW 25.60 DLRS/BBL". As a result of the high information density (which is partly caused by abbreviations) as well as the use of specific jargon the Reuters data is harder to read.

Table 2: Distribution of labels of the Gaming data set.

Label name	# train documents	Proportion of train documents	# test documents	Proportion of test documents
minecraft	2884	0.20	1463	0.20
the-elder-scrolls-5-skyrim	2034	0.14	1101	0.15
league-of-legends	1319	0.09	655	0.09
diablo-3	1288	0.09	662	0.09
Other	7186	0.49	3363	0.46
Total	14711	1	7244	1

3.3 Balanced data sets

As can be seen from the tables, the proportion of test documents in the sets ranges from 0.42 to 0.03. Moreover, the proportion of occurrence of the different labels is very similar in the English and Gaming data set, while the Reuters data set has a more variance in the occuring proportions.

As the classification methods used might suffer from the very unbalanced samples, each of the 3 data sets were also undersampled, such that the most occurring class in each data set (*'earn'*, *'minecraft'* and *'single-word-requests'*) was

[2] https://archive.org/details/stackexchange

[3] As one might be interested in the answer: https://english.stackexchange.com/questions/168038/why-is-it-always-on-foot-not-on-feet, unfortunately no (to the author) satisfactory answer is given.

Table 3: Distribution of labels of the English data set.

Label name	# train documents	Proportion of train documents	# test documents	Proportion of test documents
single-word-requests	3852	0.22	1991	0.22
meaning	2233	0.13	1125	0.13
grammar	2045	0.12	1001	0.11
word-choice	1334	0.08	677	0.08
Other	7849	0.45	4056	0.46
Total	17313	1	8850	1

balanced. The resulting three data sets are summarized in Table 4. Only the balanced label is tested in these sets. As the test set is unaltered, the proportions did not change from the proportions described in Tables 1 to 2.

In total, we have twelve binary classification tasks to test on with unbalanced data sets (Tables 1 to 3), and three binary classification tasks with balanced data sets (Table 4).

Table 4: Distributions of labels in the balanced data sets.

Data set	Label name	# train documents	Proportion of train documents
Reuters	earn	2840	0.50
	Total	5680	1
Gaming	minecraft	2884	0.50
	Total	5769	1
English	single-word-request	3852	0.50
	Total	7704	1

4 Methodology

Using the different data sets as described in the previous section, we evaluate the performance of the BoW and BoC feature representations. In this section, we explain the way Bow and BoC are implemented, and elaborate on the pre-processing of our data sets.

4.1 Pre-processing

As stated in the introduction, pre-processing is the first step in the pipeline of text classification and prepares the documents such that unwanted features (noise) in the documents are reduced. The data set R52 was already pre-processed by Ana Cardoso-Cachopo (2007) and are therefore not further altered. The steps performed by Ana Cardoso-Cachopo (2007) are:

- Substitute TAB, NEWLINE and RETURN characters by a space.
- Keep only letters (that is, turn punctuation, numbers, etc. into spaces).
- Turn all letters to lowercase.
- Substitute multiple spaces by a single space.
- The title/subject of each document is added at the beginning of the document's text.

On the StackExchange data set, identical transformations are applied. Even though the author of this article recognizes that there are better ways to pre-process the data (such as removing single quotation marks instead of replacing them with spaces as "doesn't" should be represented as *'doesnt'* and not as *'doesn'* and *'t'*). For simplicity, reproducability and comparability it was chosen not to further alter or (attempt to) improve the pre-processing step. Moreover, it could be argued that there should be good theoretical or empirical reasons before a pre-processing step is justified. As time is a limiting factor in this research, the pre-processing procedure as performed by Ana Cardoso-Cachopo (2007) is used. The only addition to this list is the removal of stop words, as the vast majority of academic literature regarding text classification seems to use a stop word removal procedure. The removal of stop words should improve the model, as it is likely that stop words do not have any explanatory power regarding the different classes in our data set. The author of this thesis acknowledges that this is only a heuristic argument, nevertheless stop word removal is included in the pre-processing.

Stop word removal is done using a parameter in `CountVectorizor` function from the python package `scikit-learn` (Pedregosa et al., 2011). Enabling the stop word removal parameter in `CountVectorizer` removes all words which are featured in a 318 long list as described by the Glasgow Information Retrieval group[4]. Moreover, all single character words were removed, as the R52 data set did not contain any single character words either; thus a total of 344 unique words were removed.

In addition to the steps listed above, (escaped) HTML tags were present in the StackExchange data set. All of this mark up was removed, that is, everything between "<" and ">" or the escaped versions "&lt;" and "&gt;".

4.2 Bag-of-Words

To answer our first research question, *'How do the Bag-of-Words (with different TFIDF weightings) and Bag-of-Concepts feature representations compare in performance?'*, we need to evaluate the performance of BoW on our data sets. To do so, we first make a BoW feature representation of our data sets (split in a training and test version), and classify the test set using NB. In the remainder of this section, we explain how the BoW and BoC feature representations are made.

A vocabulary of each data set was made by taking all words in both the test and training set. Next, bigrams are made using the `Phrases` function of the `gensim` python package (Řehůřek and Sojka, 2010). This function defines bigrams as two subsequent words according to a formula described by Mikolov et al. (2013a). The formula is shown in Section 2.4.7, Equation 3.

[4]`https://github.com/igorbrigadir/stopwords/blob/master/en/glasgow_stop_words.txt`, note that the word *'computer'* is on this list while the authors of `scikit-learn` did remove it in their implementation.

The default values for this equation in the `gensim` implementation are 10 for *min_count* and 5 for *threshold*. However, after a visual inspection of the bigrams, the author of this researched deemed that many of the generated bigrams are incorrect. Examples of some incorrectly generated bigrams are *'experts_say'* or *'i_am'*. Therefore, in the implementation for this thesis, *min_count* is set to 50, while *threshold* is set to 20. As these parameters were chosen based on a short visual inspection of the resulting bigrams, it is likely that these values are not optimal.

There are many ways to improve on the bigram generation. For example, one could use a set of pre-defined bigrams which are generated on a single large corpus with optimized parameters. However, as the scope of this thesis is limited, there is no room for this optimization. For this research, bigrams were generated separately for each data set, using the same parameters described above. This obviously has its consequences as the two StackExchange data sets have a much larger vocabulary (due a lower information density as well as slang being used, which is not present in Reuters news articles) which is represented in the formula as N. The vocabulary size is shown in Table 5.

About 150 ($\pm$ 50) bigrams were retrieved for each data set. If we instead used the default parameters of the python package `scikit-learn`, 3000 ($\pm$ 1500) bigrams would have been generated.

After generating the bigrams, the vocabulary of the training set (after pre-processing) was used to determine the features of the BoW. This can also be seen in the second column of Table 5. Only using the vocabulary of the training set results in some words in the test set not having a representation (or column) in the BoW, but these words were simply ignored.

Subsequently, term frequency (TF) BoW matrices were generated using the `CountVectorizor` from `scikit-learn` (Pedregosa et al., 2011). This TF BoW matrix is denoted as *Integer* in this thesis. After the integer BoW matrices were made, the following three different transformations on Bag-of-Words were applied and tested for performance (as measured by F1-score) using naive Bayes.

1. Conversion to Boolean matrices

2. TF-IDF transformation using the `scikit-learn` package (denoted as SK-TFIDF)

3. TF-IDF transformation as suggested by Kosmopoulos et al. (2008) (denoted as log TFIDF)

Naive Bayes was performed using the `MultinomialNB` function from `scikit-learn`. In this function, a parameter value for α, which denotes the amount of smoothing, has to be supplied. This value was set to one, which boils down to the usage of Laplacian smoothing (Equation 4), following literature (Schneider, 2004; Kosmopoulos et al., 2008; Metsis et al., 2006). Furthermore, the class priors ($p(true)$ and $p(false)$) were fit on the training data.

4.3 Bag-of-Concepts

Besides the BoW representation with different weights, we also want to investigate the BoC representation. Before the BoC representations can be made, all words in the vocabulary have to have a `word2vec` embedding. A visual reminder of the steps can be found in Figure 1.

BoC is made by clustering the word vectors from two different `word2vec` embeddings. The first set of word vectors result from training the `word2vec` network on the to be classified data sets; that is, the data sets as explained in Section 3.

The other set of vectors come from a pre-trained model as delivered by Mikolov et al. (2013a). This model was trained on a Google News data set which consists of roughly 100 billion words and has a vocabulary of 3 million words and phrases, with a vector length of 300 features. To fairly compare them, we also make use of 300 features in the model we train on our own data sets.

We work with `word2vec` through the implementation in the `gensim` python package. We train our model on the collection of all sentences in our test and train set (which were pre-processed in the same was as described in Section 4.2). A window size of 5 is used (as is customary) and no min count for words is used. In line with the Google model, negative sampling (with 5 noise words) instead of hierarchical softmax is used.

From the Google pre-trained model, the word vectors of three million words in their vocabulary can readily be downloaded. It should be noted that Google's vocabulary contains words with capital letters, non-alphabetical characters and phrases. Even though the pre-trained model has a vocabulary of three million, many of the words in our corpus do not occur in Google's model. Because of this, we have to deal with out-of-vocabulary words. This vocabulary discrepancy is handled in a very naïve way. If a word from our corpus occurs in Google's vocabulary, we keep it. If the word does not occur in all lowercase in Google's model, but it does occur with the first character capitalized

('Ohio' instead of 'ohio'), the capitalized word embedding is kept. If it does not occur in all lower case or with the first letter capitalized, the word is ignored.

Once we have all our word embeddings (via the self-trained model or from Google's model), we look up the word vector of each word in our vocabulary, and save them in a matrix. This matrix is N by 300 when trained on our own data sets (with N being the second column in Table 5), and k by 300 when using Google's model (k being the third column in Table 5). These two matrices can be clustered to obtain the concepts necessary for the Bag-of-Concept representation.

4.3.1 Clustering the word vectors

At first, the word vectors as generated by the procedure described above were clustered using k-means. However, after inspecting the clusters, many nonsensical words (random string of letters with no interpretation such as *'mar'*, *'bne'*, *'mias'*, *'efa'* or *'efe'*) seemed to dominate every cluster. As these nonsensical words all occurred sporadically (less than three times), it was chosen to adjust our documents such that only words which occur at least five times in the entire corpus are incorporated in the clusters. This reduced the size of the vocabulary/word vector matrices by a significant amount. The size of our vocabulary when `word2vec` was trained our own data sets reduced to about ten thousand words (fourth column in Table 5), while the intersection of our vocabulary with that from Google's model was even smaller (fifth column in Table 5).

Table 5: Vocabulary size of our data sets. In brackets is the feature representation which uses the specified vocabulary.

	Full	No stop words (BoW)	No stop words, in Google's vocab	No stop words, freq >= 5 (BoC self-trained)	No stop words, freq >= 5, in Google's vocab (BoC Google)
Gaming	28857	28529	14639	10765	7720
English	46085	45748	23938	14721	11311
Reuters	26284	25975	11905	8739	5606

The clusters were made using `scikit-learn`'s k-means function. As this function does not have an option to use a cosine distance metric, the vectors were normalized and subsequently Euclidean k-means was performed. As for normalized vectors Euclidean distance is equivalent to a linear transformation of cosine distance (the proof is shown in Appendix D), k-means with normalized vectors with the L^2 norm results in the same clusters as cosine similarity without normalization.

K-means is in practice a fast algorithm, but it is likely to converge to local minima. In an attempt to find the global minimum (or a good local minimum), a number of random starts have to be performed. The best local optimum, that is, the one with the lowest within-cluster sum of squares, is considered as the best cluster partition.

The amount of clusters which were investigated are [10, 25, 50, 100, 150, 200, 250, 300, 400, 500, 750, 1000]. To initialize starting points, the k-means++ algorithm was used, as is default in the `scikit-learn` k-means implementation used in this thesis, this algorithm is explained in Appendix A. To determine an acceptable number of random starts, maximum iterations and tolerance which declares convergence, some different parameter values were tested for a cluster size of 50. From this test, 150 random starts with a maximum number of iterations of 150 and a tolerance of 0.5 was estimated as acceptable, while keeping the running times in check.

5 Results

After pre-processing and obtaining the BoW and BoC feature representation of our data sets, we can answer our first subquestion: *How do the Bag-of-Words (with different TFIDF weightings) and Bag-of-Concepts feature representations compare in performance?* To answer this question, we apply MNB to our data matrices and calculate the F_1-scores of our different classification tasks using the BoW representations with different weightings and the BoC representations, which subsequently can be compared. In this section, we first show the F_1-scores from the BoW feature representation and subsequently show the same results for the BoC feature representation. A comparison with the from theory expected results is made, as well as a comparison between the results from BoW and BoC.

5.1 F_1-scores of Bag-of-Words representation

In this section we evaluate the performance o the BoW representation on our three data sets, both with unbalanced as well as balanced labels. We first discuss the unbalanced data sets, and subsequently compare them with the balanced versions.

5.1.1 Unbalanced data sets

In Tables 6 to 8, we can see the F_1-score of the binary classification problem, using MNB with different transformations of the BoW representations. The number displayed in bold is the BoW variant that corresponds to the highest (before rounding) F_1-score.

By comparing the tables below to Tables 1, 2 and 3, one can see that as the proportion of a certain label drops, the difference in F_1-score between SK-TFIDF and the other BoWs increases. The zeros in Table 7 and 8 are caused by no positive predicted samples which leads to an undefined precision, which is shown as zero. The accuracy for SK-TFIDF is still quite high and in some cases even the highest (albeit with a small margin), as can be seen in Appendix B. Nevertheless, one should be careful comparing the accuracy scores as the classes are all unbalanced, leading to a high accuracy by simply predicting all documents as the majority class (see the tables in Section 3).

Table 6: F_1-scores of different types of Bag-of-Words in the Gaming data set.

	Integer	Boolean	SK-TFIDF	log TFIDF
minecraft	**0.89**	0.89	0.56	0.88
the-elder-scrolls-5-skyrim	0.89	**0.89**	0.41	0.87
league-of-legends	0.90	**0.90**	0.36	0.88
diablo-3	**0.8**	0.78	0.04	0.70

Table 7: F_1-scores of different types of Bag-of-Words in the English data set.

	Integer	Boolean	SK-TFIDF	log TFIDF
single-word-requests	**0.67**	0.60	0.01	0.39
meaning	**0.28**	0.14	0	0.08
grammar	**0.32**	0.16	0	0.06
word-choice	**0.04**	0.01	0	0.01

Table 8: F_1-scores of different types of Bag-of-Words in the Reuters data set.

	Integer	Boolean	SK-TFIDF	log TFIDF
earn	**0.98**	0.98	0.97	0.97
acq	**0.97**	0.97	0.79	0.95
crude	**0.86**	0.85	0.12	0.74
trade	0.58	**0.63**	0	0.45

From these tables, it can be concluded that integer BoW without any transformations is the most reliable method of feature representation for these data sets. This is unexpected, as literature reports that Boolean features are superior to the integer variants, and using a TFIDF transformation is usually reported to increase performance compared to not applying a TFIDF transformation. Furthermore, the SK-TFIDF (which is often used) performs worse than the log TFIDF.

These results are surprising and thus interesting and a thorough investigation as to why this exactly happens should be performed. As TFIDF is widely regarded as an heuristic with a lack of substantial theoretical foundation, it is unclear why these results are achieved. To explain the theory behind TFIDFs, some relation towards information theory or even a probabilistic description have been attempted, but defining the appropriate event space (and sticking with it) becomes troublesome. The interested reader is advised to read Robertson (2004) for more information, which names many different kind of theoretical problems TFIDF transformations have. As a result of this heuristic framework, there are no clear guidelines on when TFIDF works. Nevertheless, we attempt to name some anomalies in our data which could explain the difference compared to other literature. The items are listed in such a way that the first reason is most likely a significant factor, while the last item is unlikely to be a cause.

1. No feature selection was performed, which also implies that all words in our corpus are used as a feature. In literature only a number of attributes, which often have a frequency threshold, are used as Bag-of-Words features. Since no feature selection was performed, rare words in our vocabulary are subsequently given a high weight with TFIDF, due to the inverse relation with frequency. It is possible that the inclusion words with a low discriminatory power (in other literature filtered out in feature selection) and rare words (filtered out with a frequency threshold), decrease the performance of our classifier. This is further shown and discussed in Section 5.5.

2. Many words with the highest (log) probability for the *true* class are also frequently occurring. This is contrary to the assumption of IDF (which states that frequent occurring words should be down sampled). E.g. the word with the highest probability of being the label *'minecraft'* is the word *'minecraft'* itself. This is further elaborated on in Section 5.5.

3. TFIDF is unsuitable in combination with naive Bayes. As we see large differences between the accuracy of SK-TFIDF and the log TFIDF, this is unlikely to be a huge factor. Moreover, literature has shown good results with TFIDF and NB (Rennie et al., 2003). Nevertheless, when comparing the results with a random forest, the differences between the four methods are smaller (see Appendix C). This could hint to the fact that the problem lies within the combination of NB and TFIDF, instead of solely with TFIDF. It is preliminary to draw any hard conclusions from these results, as only a small set of parameters have been tested. This impact of parameter choice is further discussed in Section 5.6.

4. The three data sets contain very unbalanced classes, which is an issue for TFIDF (Joachims, 1997; Frank and Bouckaert, 2006). This can also be seen by comparing the F_1-score to the accuracy of the classifier. Even though Tables 6 to 8 imply that the classifier performs poorly, the accuracy is still quite high, due to the unbalanced data sets. As the classification goal is often to obtain an as high as possible accuracy, a low F_1-score is not necessarily detrimental.

5. Document length could be an issue. TFIDF is known to work better with longer documents. Although the documents are not particularly short. Here each document contains on average (standard deviation) 39 (63) words for Gaming, 35 (27) words for English and 65 (66) words for Reuters.

Even though this list gives some intuitive explanations for the differences between TFIDF and non-TFIDF, it does not fully clarify why the Boolean BoWs do not outperform the integer ones and why `scikit-learn`'s BoW performs worse than the log TFIDF. It should be noted that the performance gap between the Boolean BoW and the integer BoW is often small, but Boolean does not perform better than the integer BoW in our experiments, which is in contrast with, for example, Metsis et al. (2006). Unfortunately, the scope of this research is limited and a thorough investigation of the underlying issues is not feasible.

5.1.2 Balanced data sets

Table 9: F_1-scores of different types of Bag-of-Words in the Balanced data sets.

	Integer	Boolean	SK-TFIDF	log TFIDF
minecraft	0.32	**0.35**	0.29	0.29
single-word-requests	**0.55**	0.54	0.51	0.49
earn	**0.98**	0.98	0.98	0.97

The balanced data sets do not perform better than the unbalanced sets. Even though the recall of all three labels is above 95 percent, a huge drop in precision causes a lower F_1-score. This is not unexpected, as the probabilities that a word occurs in the positive class as well as the class priors are overestimated. The F_1-score of the label *'earn'* is not much different when comparing the balanced data set to the unbalanced data set, but that is likely caused by the fact that in the unbalanced data set, the class *'earn'* already had a proportion of 0.43, which is not that unbalanced. Therefore, the results in Table 9 for *'earn'* are similar to those in Table 8. As the results for the balanced data set using BoW are not very promising, we do not further investigate them using Bag-of-Concepts.

5.2 F_1-scores of Bag-of-Concepts representation

The same classification tests on the unbalanced data sets are performed using the BoC representations. Clusters were made as described by the procedure in Section 4.3.1 using different cluster sizes. The inertia of the clusters (the Euclidian distance of each point after normalization to its clusters centroid, summed for all points) can be seen in Figure 7. These inertia's are not surprising: as expected the inertia decreases as the number of clusters increases. The inertia for the word vectors which were made by training on the data set have a much lower inertia than those resulting from Google's `word2vec` model, even though the Google model has a smaller vocabulary. This might imply that the clusters from training the `word2vec` model on the to be classified data sets are better, as the lower within-cluster sum-of-squares shows that the points are clustered closer together.

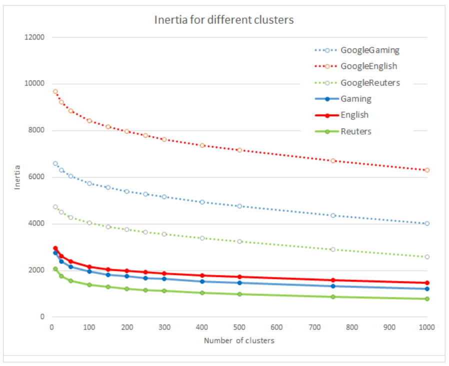


Figure 7: Within-cluster sum-of-squares of cosine k-means clusters with different number of clusters, for Google's pre-trained `word2vec` model and trained on the data set itself.

The resulting F_1-scores of the classification tasks are shown in Figures 8 to 10. Since the number of clusters is a variable that also should be shown, we use figures instead of tables to show the BoC F_1-scores. In these figures, the (straight) dashed line is the performance of the integer BoW representation.

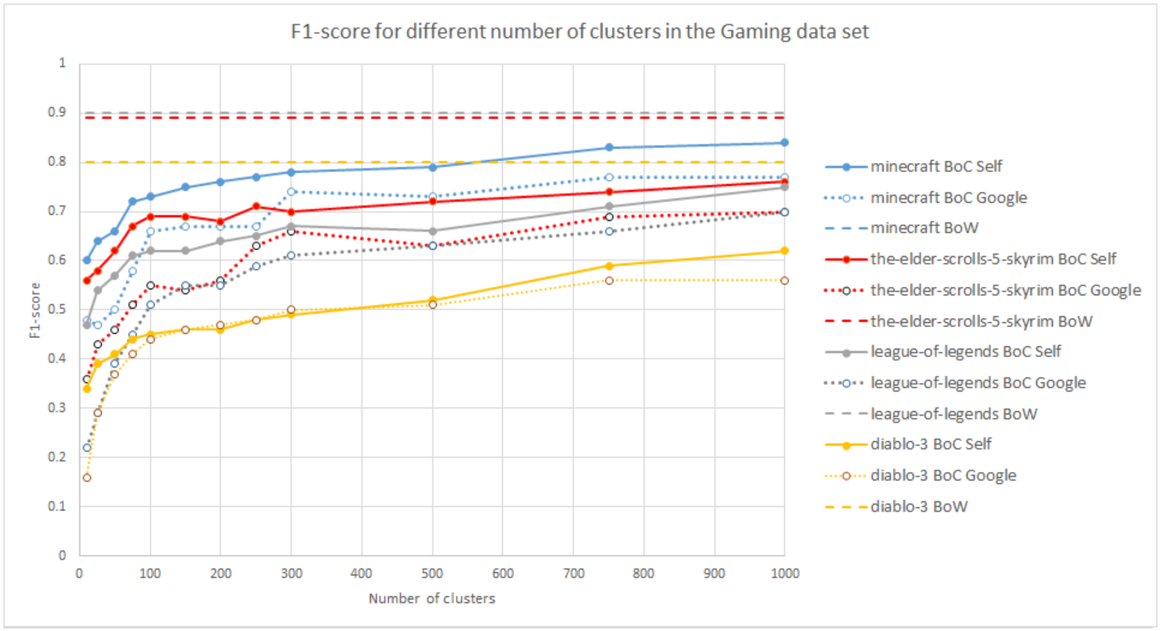


Figure 8: F_1-score of Bag-of-Concepts for different cluster sizes on the Gaming data set. Clusters made with Google's pre-trained `word2vec` model and trained on the data set itself. The BoW line is the F_1-score from the integer BoW representation.

Compared with BoW, the results for BoC are a bit worse, with the exception being the English data set (Figure 9). For the English data set, BoC performs about equal or better than BoW. In general, the more clusters (concepts) the better, but the marginal increase is low after about 100 clusters. This is in line with Figure 7, where the decrease in cluster inertia seems to drop off after about 100 clusters. The clusters made with Google's `word2vec` word vectors perform worse than the clusters made with word vectors resulting from training on the data set itself, with the only exception being the *'crude'* label in the Reuters data set. The two different BoC versions do follow the same trend: a sharp increase in performance as the number of clusters increases from 10 to 100, after which the F_1-scores stabilizes. Overall Google's version performs worse, although there are some exceptions in some data sets with some number of clusters.

It is expected that as the number of clusters increases to the number of words in the vocabulary, the BoC feature representation should approach the integer BoW F_1-score. In the end, if the number of clusters is equal to the number of words in the vocabulary, the inertia is minimized by assigning each word their own cluster which results in the integer BoW representation. The trend of BoC towards the BoW F_1-score is indeed visible, although for some labels there are considerable gaps in the BoW F_1-score and the BoC F_1-score at 1000 clusters. The gap could also be caused by a difference in vocabularies: the vocabulary used for the BoCs is different than the vocabulary used for the BoWs (Table 5). This means that an exact convergence towards the BoW result is unlikely. Nevertheless, the performance of the BoC with 1000 clusters is in most cases not too far off the performance of the (integer) BoW representation. Even though there is less information available in the BoC representation, namely about one third in vocabulary size (Table 5) which are clustered in even less features, the F_1-scores in the English data set are better than those of BoW. This implies that BoC, especially for the English data set, is an efficient method to reduce the size of the dimension of the feature representation. The performance increase could also be caused by the fact that the infrequent words which are removed from the vocabulary are actually detrimental for the performance of the BoW classifier. To make a good comparison, the BoW feature selection should also be tested with the same vocabulary as BoC, but unfortunately this was not possible in this research.

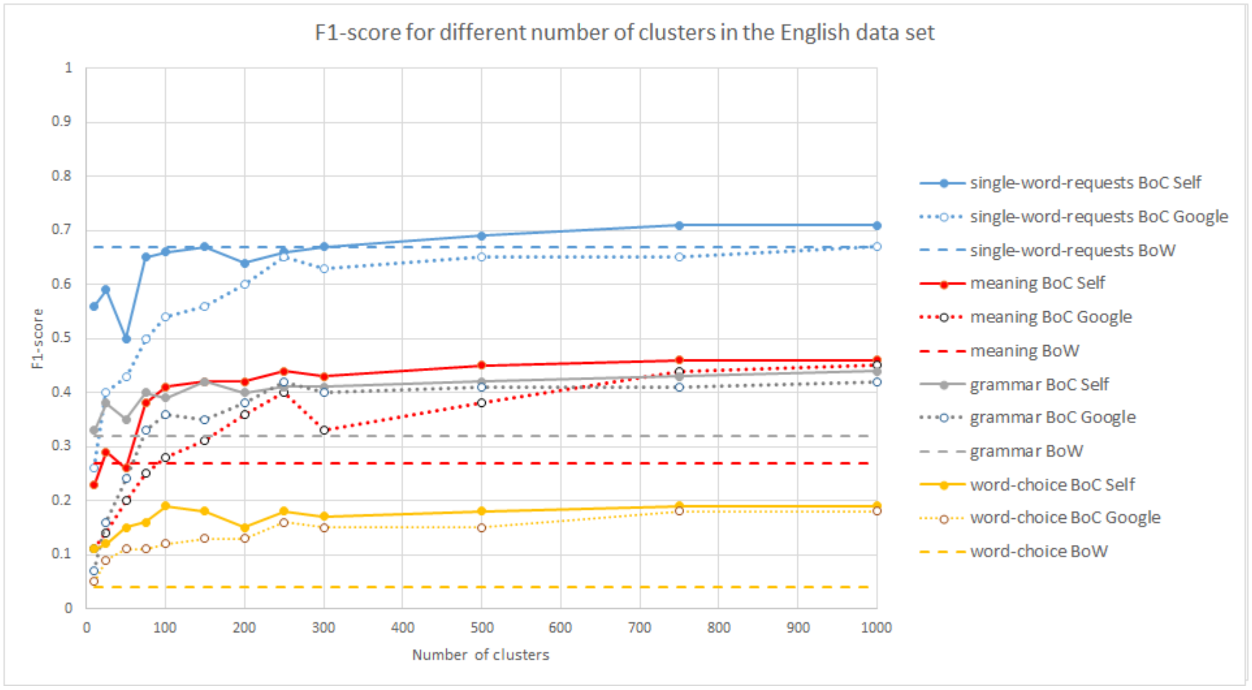


Figure 9: F_1-score of Bag-of-Concepts for different cluster sizes on the English data set. Clusters made with Google's pre-trained `word2vec` model and trained on the data set itself. The BoW line is the F_1-score from the integer BoW representation.

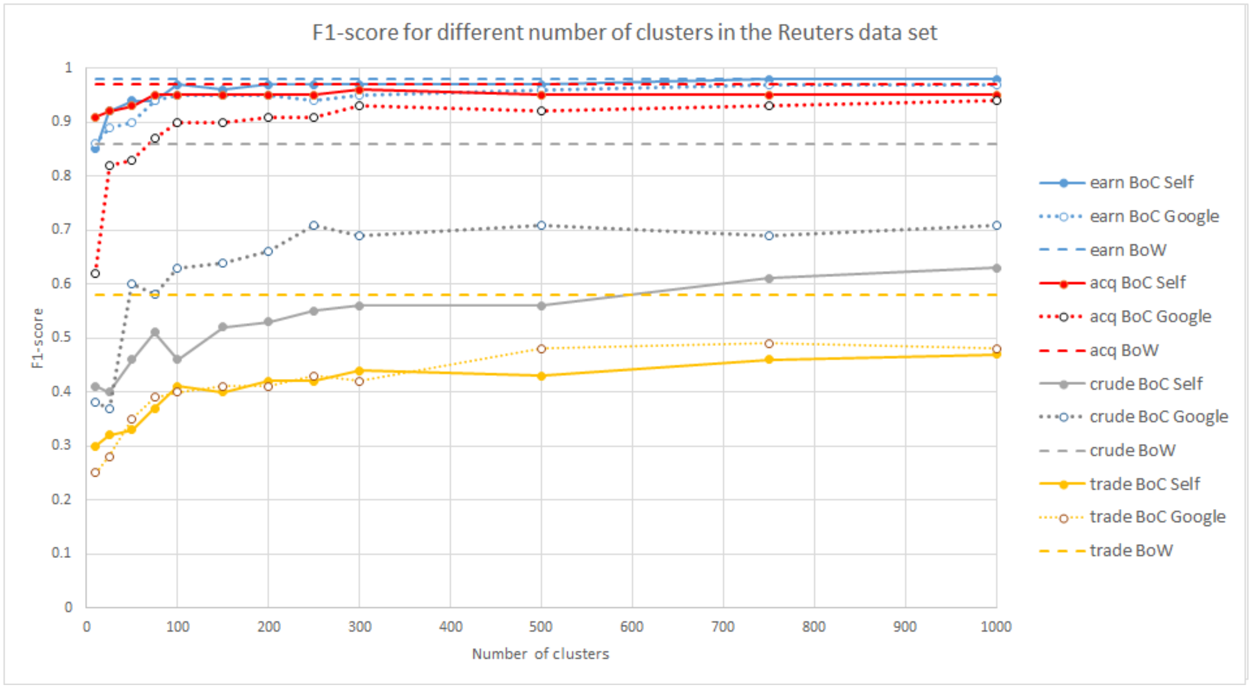


Figure 10: F_1-score of Bag-of-Concepts for different cluster sizes on the Reuters data set. Clusters made with Google's pre-trained `word2vec` model and trained on the data set itself. The BoW line is the F_1-score from the integer BoW representation.

Unfortunately, pure mathematical substantiation for these results is difficult, if not impossible to derive. In an attempt elucidate on the results we look into the probabilities for some important concepts and words that the naive Bayes classifier has estimated in the next section. It is possible that features that are important for the NB model with the integer BoW representation can easily be recognized by a human as also relevant to the class. If this is different for the models from the TFIDF BoW representations, this may shed some light on the performance differences. On a similar note, the words which make up important clusters in the BoC representation are inspected.

5.3 Correlation between feature frequency and feature probability

To explain the performance of the different representations, as well as to answer our second subquestion, *How comprehensible are these methods for humans (or: how well can the results be explained)?*, we investigate the model further. The NB model works in a simplistic way, for each feature (words in BoW and concept in BoC), the naive Bayes classifier estimates $p(feature|class)$. A higher probability correlates directly towards the conditional class. The class can be positive (document should be labeled as the class) or negative (document should not be labeled as the class).

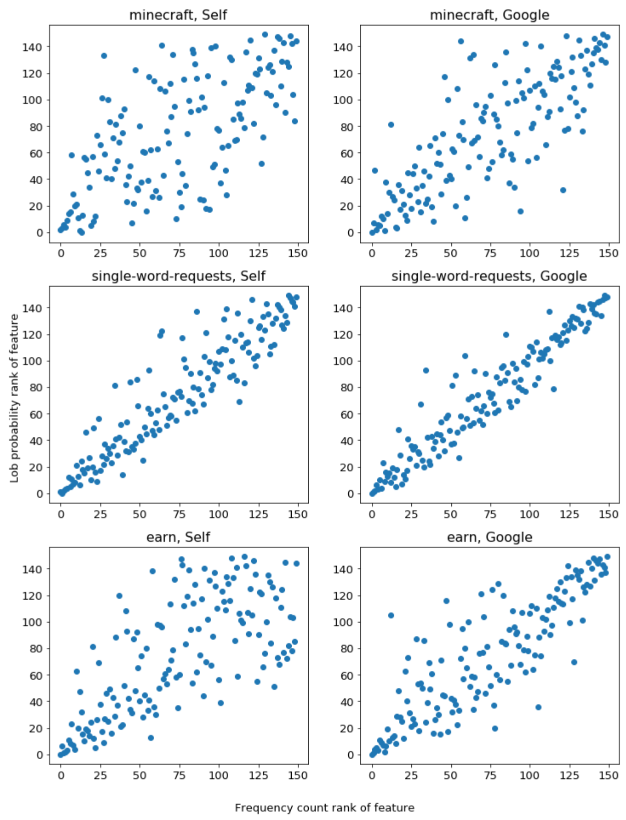
The most important word or concept for a certain label is the one for which $p(feature|positive\ class)$ is the highest. We investigate these probabilities for the BoC representations. After a short inspection, it was noted that cluster for which $p(cluster|class)$ is the highest, is identical for different labels within a data set. That is, the cluster that is the most important for the label *'league-of-legends'* also has the highest probability for the label *'diablo-3'*. It was also noted that the most important cluster was the most occurring cluster. This hints towards a relationship between the importance of a cluster and frequency count of a cluster.

To investigate this relationship, the importance of a cluster in a model (the rank in feature (log) probability for the positive class) versus the occurrence (the rank in the feature frequency count) is plotted. In Figure 11, this is plotted with the ranks in descending order). From this figure, it can be concluded that there is a correlation between the frequency rank of a cluster and the log probability rank of a cluster. On the horizontal axis, the cluster frequency rank is plotted (i.e. if we order all clusters by frequency count in the train set, which rank does the cluster have) and on the vertical axis the feature (log) probability rank of the positive category is plotted. Similar figures can be made for a different number of clusters, but as they all show similar trends, those figures are not shown in this thesis.

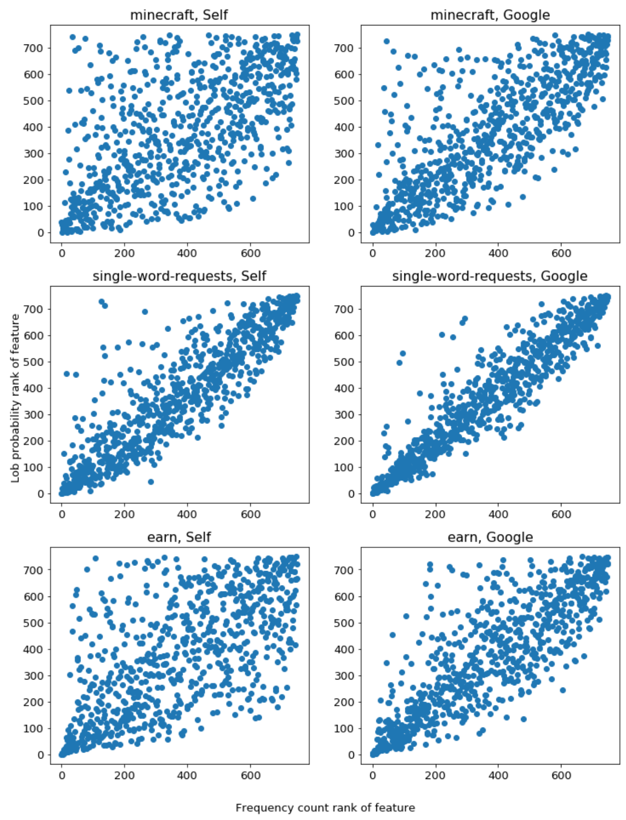
This relation can somewhat be expected from the way $p(feature|class)$ is calculated (items which occur often also occur often in the positive class, see Equation 4). For a NB classifier to be accurate, there needs to be predictive power in (some of) the words present in the training set. This implies that a word should occur more often in the set with a positive label than in the set with a negative label. If the (ranked) probability of a feature being present in a given class can be explained by the (ranked) frequency of said feature in the entire data set, its occurrence is independent of the class. Therefore, the feature contains no predictive power. If this holds true for a lot of features, the classifier should perform worse compared to a feature representation with a more scattered *Probability rank-Frequency rank* plot.

In other words, points exactly on the line y=x, should have little discriminatory power between the true and false (or positive and negative) label of a class. Points that are above the y=x line have a lower probability ranking than their frequency ranking and are therefore less associated with the class. Vice versa, points below the line should correspond positively with the associated class. Therefore, the clusters that are most interesting to investigate, are clusters that are far away from the line y=x. This trend is visible for all the investigated number of concepts. To not clutter this thesis with similar figures, the results for the other number of clusters are left out.

To look into what words make up a concept, a selection of classification tasks has to be chosen. As each cluster consists of numerous words, and we have twelve different binary classification tasks, each with eleven different number of clusters, it is not feasible to inspect them all. We chose to inspect the different clusters within the Gaming data set, for 150 clusters. The Gaming data set is chosen as it is suspected by the author of this thesis that the words which relate to a certain label can most easily be discriminated in the Gaming data set. For example, the word *'minecraft'* obviously belongs to the label *'minecraft'*. In the English and Reuters data set, these direct associations are not as apparent. The choice for 150 clusters is somewhat arbitrary, but using more than 150 clusters does not improve the performance of the classifier much. It was also found that using 150 clusters results in a nice amount of words per cluster to show in this thesis.



(a) Using 150 clusters.

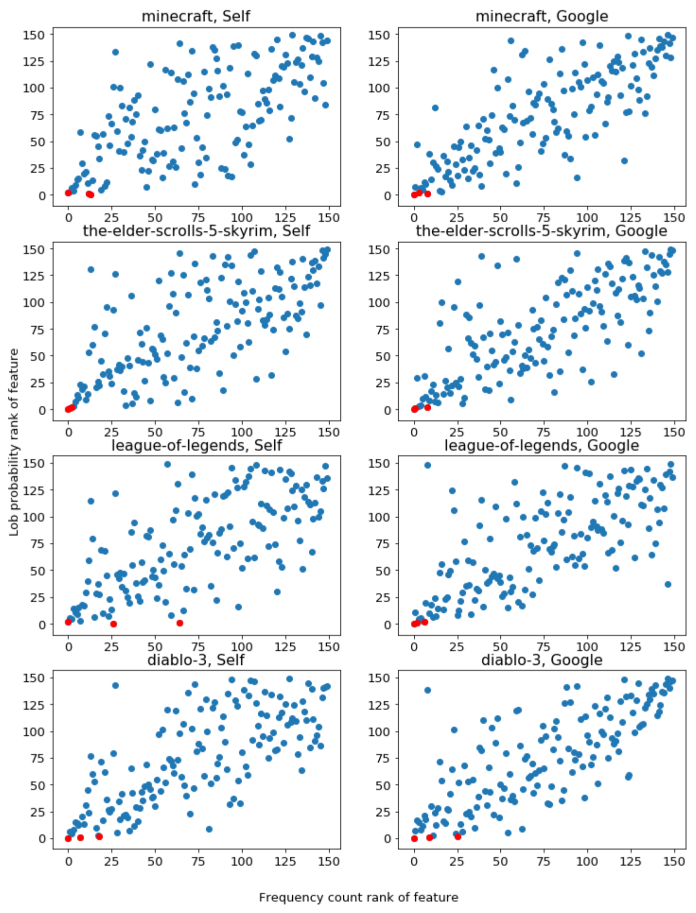


(b) Using 750 clusters.

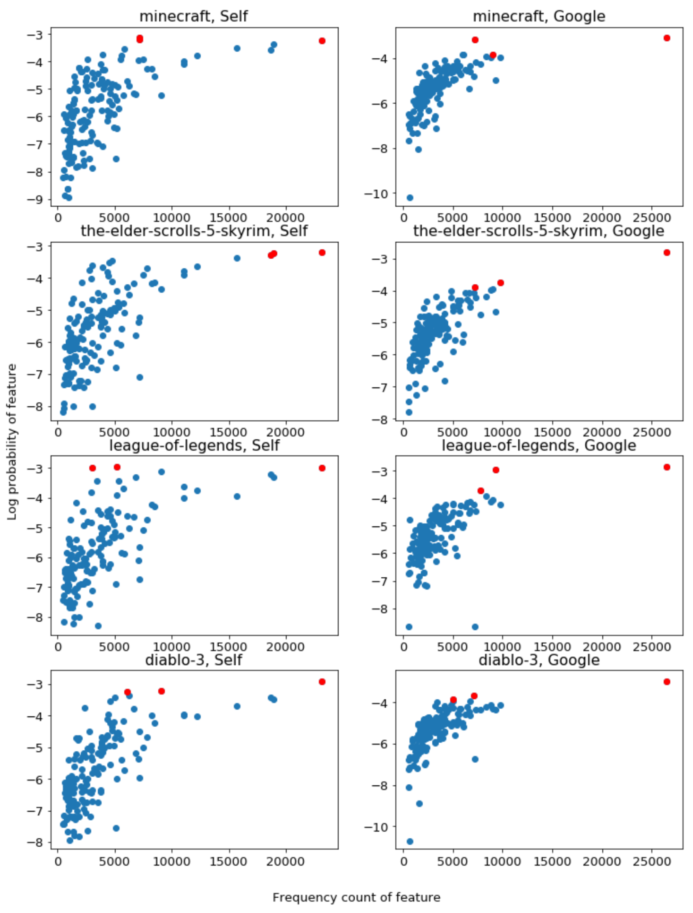
Figure 11: (Log) feature probability rank of clusters (from MNB) versus cluster frequency rank for three labels in three data sets, using two different number of clusters. Both the clusters made with Google's pre-trained model (first and third column) as well as the clusters made using word vectors trained on the data set only are shown (second and last column).

5.4 Investigating the Gaming data set BoC representations

Figures 12a and 12b give an overview between the relation of the log feature probabilities and occurrences. In Figure 12a a plot similar to Figure 11 is shown. This figure is discretized: both the feature log probability as well as the feature frequency are ordered and the rank is shown. This in some sense distorts the plot, as the continuous absolute differences between frequency and log probability of the model are no longer visible. In Figure 12b the actual continuous values for the log probability as well as the frequency counts are shown. The y-axis shows the log feature probability, for which a lower value means less important concepts, while the x-axis shows the frequency count of concepts. There is an obvious relation between the two plots in Figure 12. The bottom left corner in Figure 12a shows the same concepts as the top right corner in Figure 12b. Features with a high rank in log probability and a high rank in frequency (Figure 12a) obviuosly have a high log probability value and a high frequency count (Figure 12b).



(a) (Log) feature probability rank of the clusters versus the cluster frequency rank.



(b) Log feature probability of the clusters versus the cluster frequency.

Figure 12: Log feature probability (rank) of clusters (from MNB) versus cluster frequency rank for the labels in the Gaming data set with 150 clusters. Both the clusters made with Google's pre-trained model as well as the clusters made using word vectors trained on the data set only are shown. The clusters with the highest conditional probability are plotted in red.

The values of the probabilities as shown in Figure 12b are not discussed in this section, as the actual probabilities do not provide extra insight over the probability rank. From the figure, we can conclude that a log probability of -3 is high, and a log probability of -8 is low but this does not give extra information over a high or low probability rank. In this research it was chosen to discuss the probability and frequency rank, because a deviation from the line y=x can easily be seen. The linear correlation between y and x in Figure 12a manifests itself as a log trend in Figure 12b, from which deviations are harder to notice.

In this section, we investigate clusters with the highest conditional probability. These are shown in red in Figure 12. These points were selected, instead of the points that are orthogonally the furthest below the y=x line, because after a short investigation it was easier for the author to classify the words as relevant or irrelevant to the class. After inspecting the clusters which lie orthogonally furthest below the y=x line, clusters with the words such as *'launchwrapper'*, *'trolling'*, *'underwater'* and *'runtime'* appeared. These words could very well be related to only one of the four games investigated, but to the author of this thesis it is not immediately clear what the relationships are. If we simply investigate the clusters with the highest conditional probability, these relationships are more apparent. Therefore, latter are investigated in this thesis.

In the rest of this thesis we use short hand notation for the clusters. We denote the clusters by their probability rank and frequency rank and display them as the (x,y)-coordinates from Figure 12a. For example, the cluster with probability rank 3 and frequency rank 1 is denoted as (1,3). The coordinates for the BoC which makes use of Google's word2vec are from a different vector space than the coordinates from the BoC which makes use of our self trained word2vec model, as the vocabularies for the two data sets are different. Within a word2vec model (Google or self trained), the frequency ranks between labels are identical. If a concept is the most occurring for the classification task with the label *'minecraft'*, it is also the most occurring for the classification task with the label *'diablo-3'*, as the text within the Gaming data set does not change for the different classification tasks: only the labels change. The feature probability ranks however will differ, as these depend on the labels (and the frequency count does not). As those two properties are deemed most important in determining whether a cluster is truly important or not, we only denote these two properties of the cluster, as well as the relevant label and from which word2vec model (Google or self-trained) the BoCs are made.

From Figure 12 alone it can be concluded that there is a common concept present for all labels that has a high conditional probability. Since it is associated with the positive class of all labels, it is unlikely to be an actual important concept for one of the specific labels. This is both the case for the clusters resulting from training on the own data set, as well as for clusters resulting from Google's model. The concept (1,3) for *'minecraft'* which occurs roughly 23000 times in the self-trained version has the words:

```
['active', 'affect', 'assume', 'certain', 'different', 'does', 'example', 'gives', 'including',
   'know', 'lets', 'matter', 'mean', 'multiple', 'non', 'order', 'regular', 'require', 'say',
    'single', 'true', 'type', 'types', 'use', 'used', 'using', 'wondering', 'words', 'work']
```

in Google's version the most common cluster contains the words (1,1):

```
['actually', 'alright', 'anybody', 'anymore', 'anyways', 'awhile', 'bit', 'bunch', 'certainly',
   'crazy', 'damn', 'definitely', 'doing', 'everybody', 'exactly', 'feel', 'feeling', 'felt',
'figured', 'forget', 'forgot', 'fortunately', 'going', 'gonna', 'got', 'guess', 'guessing', 'guy',
 'guys', 'heck', 'hell', 'hey', 'honest', 'honestly', 'hopefully', 'imagine', 'imagining', 'just',
 'kidding', 'kind', 'kinda', 'knew', 'know', 'knowing', 'lately', 'like', 'little', 'look', 'lot',
   'luckily', 'maybe', 'mean', 'obviously', 'okay', 'probably', 'realize', 'realized', 'really',
'remember', 'right', 'secondly', 'somebody', 'sort', 'stuff', 'suppose', 'sure', 'surely', 'tell',
      'thing', 'things', 'think', 'thinking', 'thought', 'whatnot', 'wonder', 'yeah', 'yes']
```

The suggestion is confirmed as indeed none of these words give the impression that they are correlated to a specific label. The subsequent two most important labels for the class *'minecraft'* are more interesting. First we show the two most important clusters from the self-trained word2vec model.

The two clusters with the highest conditional probability for the *'minecraft'* label with our self-trained word2vec model are clusters (14,1) and (13,2). As explained in the previous section, we expect that the point that deviates the most from the line y=x in the *Probability rank-Frequency rank* plot to be the most correlated with the positive class. In this case, this is the cluster (14,1). It is hard to judge for a human whether or not this guideline holds. Nevertheless, the word *'minecraft'* appears in the cluster (14,1), which indeed shows that is a very relevant cluster. The words in cluster (14,1) are:

```
['adventure', 'cheats', 'creative', 'edition', 'mcpe', 'minecraft', 'mode', 'pe', 'peaceful',
                'platform', 'pocket', 'smp', 'survival', 'terraria', 'vanilla']
```

All but one of these words have as common concept the game Minecraft. *'adventure'*, *'creative'*, *'mcpe'*, *'pe'*, *'peaceful'*, *'pocket'*, *'smp'*, *'survival'* and *'vanilla'* are modes or variants in which Minecraft can be played. The only exception in this list is the word *'terraria'*, which is a different game but does have similarities with Minecraft. It could be that Terraria has identical game modes, which causes it to appear in contexts similar to the word Minecraft and therefore ends up in this cluster.

The second most important cluster for this label and model, (13,2), contains the words:

```
['animal', 'animals', 'bats', 'blaze', 'blazes', 'breed', 'cage', 'chicken', 'chickens', 'cow',
   'cows', 'creatures', 'creeper', 'creepers', 'despawn', 'despawning', 'dogs', 'enderdragon',
  'enderman', 'endermen', 'escaping', 'farm', 'ghasts', 'golem', 'golems', 'grinder', 'heads',
 'horses', 'hostile', 'mob', 'mobs', 'monster', 'monsters', 'pig', 'pigmen', 'pigs', 'prevent',
  'riding', 'sheep', 'skeleton', 'skeletons', 'slime', 'slimes', 'spawn', 'spawned', 'spawner',
'spawners', 'spawning', 'spawns', 'spider', 'spiders', 'squid', 'summon', 'tame', 'tamed', 'trap',
            'traps', 'villagers', 'wither', 'wolves', 'zombie', 'zombies', 'zoo']
```

these words are all related to animals or monsters that appear in Minecraft. It can therefore be considered a relevant feature by humans. If we inspect the clusters resulting from Google's model, we find that the three most important clusters are (1,1), (9,2) and (4,3). The cluster (1,1) has already been shown above, as it is the most common cluster in the data set. From the coordinates of the other two clusters alone, we expect the words in cluster (9,2) to be more easily associated with the game Minecraft than the words in cluster (4,3), as this cluster lies close to the line y=x. This can also be concluded from the top right figure in Figure 12b. The leftmost red dot (which corresponds to cluster (9,2)) does not follow the logarithmic trend as much as the cluster which corresponds to (4,3).

The words in cluster (9,2) are:

```
['Archon', 'Bethesda', 'Bioware', 'Cerberus', 'creators', 'demo', 'demos', 'dev', 'developer',
'developers', 'gen', 'Genji', 'guild', 'guilds', 'indie', 'Killshot', 'Magicka', 'Minecraft',
'mmo', 'mod', 'modding', 'Mojang', 'Morrowind', 'Moxxi', 'multiplayer', 'Neverwinter', 'playable',
'prerelease', 'screenshots', 'singleplayer', 'Skyrim', 'spec', 'specs', 'spoilers', 'Starport',
'Steamworks', 'walkthrough', 'Wowhead']
```

while the words in cluster (4,3) are:

```
['absorb', 'add', 'attract', 'augment', 'bring', 'build', 'combine', 'complement', 'construct',
'contribute', 'create', 'deliver', 'deploy', 'distribute', 'employ', 'enable', 'enlarge', 'ensure',
'enter', 'equip', 'establish', 'expand', 'extend', 'forge', 'fortify', 'generate', 'help',
'implement', 'improve', 'initiate', 'introduce', 'lend', 'maintain', 'make', 'manage', 'maximize',
'merge', 'occupy', 'produce', 'promote', 'provide', 'raise', 'reinforce', 'retain', 'secure',
'serve', 'shorten', 'simplify', 'supplement', 'sustain', 'synergize', 'transform', 'translate',
'use', 'utilize']
```

The words in cluster (4,3) indeed do not seem to be associated heavily with Minecraft. Cluster (9,2) does contain the word *'Minecraft'* and *'Majong'*, which is the developer of the game. However it also contains many other game developers and other games such as *'Skyrim'* and *'Starport'*. From these clusters, we can conclude that the `word2vec` algorithm when trained on the data sets that are to be classified form more useful clusters than when using Google's `word2vec` model. This intuitively makes sense, as Google's model embeds the context of a word over a large news corpus. In this corpus, game names and game developers appear in similar documents. In our own data set, game names appear in the context of questions regarding that specific game and therefore are more closely associated with the specific label.

To show the difference between the clusters resulting from Google's `word2vec` model as well as our own, we show two related clusters from the *'league-of-legends'* label. For our own `word2vec` model, the cluster of interest is (65,2). From these numbers alone, it is already apparent that this cluster has a correlation with the label. This cluster is relatively rare, but also very high conditional probability. This cluster contains the words:

```
['ad', 'adcs', 'akali', 'amumu', 'annie', 'ap', 'ashe', 'azir', 'blitzcrank', 'burst', 'caitlyn',
'cc', 'champion', 'cho', 'combo', 'corki', 'darius', 'ezreal', 'fizz', 'gath', 'gnar', 'gragas',
'graves', 'healer', 'healers', 'hydras', 'irelia', 'janna', 'jarvan', 'jax', 'jayce', 'kalista',
'kayle', 'kennen', 'kog', 'lee', 'leona', 'lucian', 'lulu', 'malzahar', 'maokai', 'marksman',
'maw', 'mobility', 'mordekaiser', 'morgana', 'mundo', 'nasus', 'nidalee', 'nocturne', 'nuke',
'nunu', 'olaf', 'rammus', 'ranged', 'renekton', 'riven', 'roach', 'rumble', 'ryze', 'shen', 'sin',
'smite', 'sona', 'soraka', 'swain', 'tank', 'tanky', 'taric', 'teammates', 'teemo', 'thor',
'thors', 'thresh', 'treeline', 'tristana', 'tryndamere', 'udyr', 'vayne', 'volibear', 'xin', 'zed']
```

These words are all (related to) 'champions' (characters) that can be played as in the game League of Legends. Since these names are unlikely to occur in any other context in our data set, this concept is highly correlated with our label. However, when we look for the cluster is associated with champions in Google's model which has the coordinates (3,2) we get the words:

```
['appearances', 'Aztecs', 'baseball', 'bracket', 'champ', 'champion', 'champions', 'championship',
'champs', 'club', 'compete', 'competing', 'competition', 'contest', 'crown', 'eliminations',
'favorites', 'game', 'games', 'javelin', 'league', 'leagues', 'matches', 'matchup', 'matchups',
'medals', 'midget', 'race', 'races', 'relay', 'relays', 'season', 'seasons', 'sprint', 'squad',
'squads', 'team', 'teams', 'title', 'titles', 'tournament', 'tournaments', 'trophies', 'trophy']
```

These words refer to a champion in the general meaning of the word. Convienently, this cluster also contains the word *'league'*, which is a good predictor for the class *'league-of-legends'*. Likely because of the words *'champion'*, *'champion'* and *'league'* this cluster has a high frequency count. The rest of the words in this class are not not directly associated with the game. This demonstrates both the power of training the `word2vec` model on the data set of interest, as well as the predictive power of the (log) feature probability (rank) and the feature frequency (rank). A strong deviation from the y=x line in the *Probability (rank)-Frequency (rank)*-plot implies a relevant concept which is interesting to further investigate.

The clusters with the highest conditional probability towards the class *'the-elder-scrolls-5-skyrim'* are all on the line y=x in Figure 12a or somewhat equivalent follow a logarithmic trend in Figure 12b. After inspection, they are indeed general common words and are therefore not further discussed here. The clusters for the BoC resulting from training `word2vec` on the data set (2,2) and (3,3) are shown in Appendix E. The cluster (2,2) from Google's model is also shown in Appendix E. The cluster (1,1) from both the self-trained model and Google's model has already been discussed above. As the frequency of clusters is used as x-coordinate and it is independent of the class the cluster from Google's model (9,2) for the class *'the-elder-scrolls-5-skyrim'* is identical to cluster (9,3) for the class *'minecraft'*, which is also shown above. The clusters regarding the class *'diablo-3'* are not further discussed as the goal of this thesis is to investigate the performance of BoW and BoC and not to demonstrate the concepts that are present in different games.

Even though some interesting words in concepts have been shown, the investigated concepts do not explain the F_1-scores of the NB classifier. The F_1-scores of the labels using our own `word2vec` model are the highest for the label *'minecraft'*, after which *'the-elder-scrolls-5-skyrim'* follows, then *'league-of-legends'* and the lowest F_1-score is for the label *'diablo-3'*. For the *'the-elder-scrolls-5-skyrim'* label, we have concluded that none of the three concepts with the highest conditional probability can directly be associated with the game by a human. For the *'league-of-legends'* label, there is at least one concept of the top 3 highest conditional probabilities that can be associated with the game by a human. Nevertheless, the classifier performs worse for *'the-elder-scrolls-5-skyrim'* than for *'league-of-legends'*. Obviously, there are many more concepts than three, 150 in this case, which should be investigated. This is intractable for this thesis research and it is unknown if this would result into more insight.

5.5 Investigating the Gaming data set BoW representations

Similar to what we did in Section 5.4, we investigate the five most important words in each BoW classifier for the labels *'minecraft'*, *'the-elder-scrolls-5-skyrim'* and *'league-of-legends'*. Once again, we show the words themselves, as well as their frequency rank after TFIDF/Boolean transformation (if applicable), as can be seen in Tables 10 to 12. Most different BoWs versions have a similar top five of most important words, but the order of importance of these words shuffled.

Table 10: Five most important words for the different BoW classifiers for the label *'minecraft'*. Between brackets is the frequency rank of the word.

	Prob. rank 1	Prob. rank 2	Prob. rank 3	Prob. rank 4	Prob. rank 5
Integer	minecraft (3)	server (31)	java (54)	way (7)	world (25)
Boolean	minecraft (8)	just (2)	way (6)	like (4)	know (7)
SK-TFIDF	minecraft (1)	server (29)	blocks (52)	spawn (36)	world (22)
log TFIDF	minecraft (1)	server (34)	blocks (61)	spawn (39)	block (67)

Table 11: Five most important words for the different BoW classifiers for the label *'the-elder-scrolls-5-skyrim'*. Between brackets is the frequency rank of the word.

	Prob. rank 1	Prob. rank 2	Prob. rank 3	Prob. rank 4	Prob. rank 5
Integer	skyrim (40)	quest (28)	ve (6)	just (4)	level (9)
Boolean	skyrim (35)	just (2)	quest (37)	ve (5)	way (6)
SK-TFIDF	skyrim (33)	quest (26)	level (5)	armor (35)	ve (7)
log TFIDF	skyrim (29)	quest (28)	ve (6)	dragon (136)	level (9)

Table 12: Five most important words for the different BoW classifiers for the label *'league-of-legends'*. Between brackets is the frequency rank of the word.

	Prob. rank 1	Prob. rank 2	Prob. rank 3	Prob. rank 4	Prob. rank 5
Integer	does (2)	damage (10)	game (1)	like (5)	champion (204)
Boolean	does (1)	game (3)	like (4)	know (7)	league (181)
SK-TFIDF	league (123)	champion (204)	legends (235)	damage (10)	lane (245)
log TFIDF	league (128)	legends (210)	champion (214)	champions (267)	lane (280)

The most apparent difference between the integer BoW and the other three types is visible for the label *'league-of-legends'*, Table 12. The only word that seems specific for League of Legends is *'champion'*, which is only the fifth most important word. The other four words in the top five are debatable. For SK-TFIDF and log TFIDF, the words *'league'*, *'legends'* and *'champions'* also appear in the top five: these words are clearly associated with the game. This table contradicts the actual performance (F_1-score) of the classsifier (third row of Table 8). We can conclude that information regarding the five most important words in the BoW classifier is not enough to explain the results. As there are over 25000 words in the vocabulary of each data set, this is not surprising.

In these tables, some words that occur with a high frequency rank that are helpful for the classification appear, such as *'minecraft'* in Table 10, but also words that do not seem to have any discriminatory power such as the words *'just'*, *'ve'* and *'like'* are listed. The word *'ve'* is an artifact resulting from the removal of apostrophes: *should've, would've* in the original data set becomes *should ve, would ve* after pre-processing, as discussed in Section 4.1. This indicates that better pre-processing steps or supervised feature selection is likely to improve results of the classification tasks, as these frequent occurring words should not be as influential in the model.

Table 11 show that the most important word for the label *'the-elder-scrolls-5-skyrim'*, which is the word *'skyrim'*, decreases in frequency rank (occurs relatively more) from the integer BoW to the other BoW variants. A similar relative frequency increase is seen for the word *'minecraft'* in Table 12 for the TFIDF transformations. This is in contrast with the performance of the classifiers, the integer classifier performs better than the SK-TFIDF classifier and about the same for the log TFIDF classifier.

In Section 5.1, it was stated as possible explanation for the bad TFIDF performance that important words are also common, which goes against the assumptions of TFIDF. From Table 10 and 11, we can conclude that this does not have an actual cause, as the relative occurrence of the important and common words *'minecraft'* and *'skyrim'* is increased due to the TFIDF transformation. This strengthens our conclusion that information regarding the five most important words in the BoW classifier is not enough to explain the results.

5.6 Comparison with literature

The results of this thesis are mixed as the performance of TFIDF is not in line with reported results in other studies. The BoC performance is also not better than the BoW model, but the BoC feature representation has shown some interesting concepts. In this section, we elaborate on performance difference found in this thesis and the results in other research.

Kim et al. (2017) was one of the first to use BoC for document representation and reported promising results. The explanation for the BoC and BoW performance gap compared to the paper of Kim et al. (2017) is simple. The experiment they used to test the performance of their methods was not a classification task. In their paper, they measured document similarity (using cosine distance) between three documents. Two of the three documents were of the same class, while the third document had a different class. If the distance between the two documents of the same class is smaller than the distance between the documents of different classes, the test would be marked as success. This is clearly completely different than the text classification task using NB that was performed in this thesis. As BoC is a novel method, the author of this thesis could not find any other literature that makes use of the BoC representation. Therefore the performance of BoC that is found in this thesis cannot directly be compared to literature.

BoW, on the other hand is a lot more common. Transformations that belong to the class of TFIDFs are used in numerous papers, although no paper directly comparing SK-TFIDF with log TFIDF is found. In literature it is found that TFIDF transformations as well as Boolean feature representations lead to better results than using the integer BoW feature representation Kosmopoulos et al. (2008). On the data sets used in this thesis, this improvement is not found. Numerous explanations for the discrepancy between the TFIDF results of this research and literature have been given in Section 5.1, but this thesis was not able to give a waterproof explanation.

This does not mean that all of our results are in conflict with previous work. In our results, integer BoW did perform better than Boolean BoW, but only with a small margin. Metsis et al. (2006) investigated different types of NB for spam filtering and found in their research that Boolean attributes are in general better, but he also states *"the difference, however, is in most cases very small and not always statistically significant"*, which is in accordance with the results of this research, albeit that we found that the integer BoW does perform better.

To gain a better understanding of the poor results found for BoW TFIDF in combination with MNB, we shortly discuss two different papers that have a similar investigation. The first paper is Rennie et al. (2003), which uses a BoW feature representation in combination with MNB. They attempted different TFIDF transformations and found that *"the length normalization transform to be the most useful, followed by the log transform. The document frequency transform seemed to be of less importance."* They did not provide any underlying theory and they used a different data set and score metric (precision-recall break even point). It is therefore hard to compare the results of this thesis with Rennie et al. (2003).

The other paper we briefly discuss is Puurula (2012) who shows that optimizing the TFIDF transformation improves the performance of NB with multiple percentage points. But just like Rennie et al. (2003), they provide no insight in the underlying mechanism. In fact, the main contribution of their paper is the proposal of a search algorithm that is a better alternative to a grid searches to optimize the parameters of the TFIDF transformation and MNB for the best performance. This lack of substantiation is a recurring phenomena in papers regarding text classification using machine learning.

These two papers only list the performance difference between their methods and other methods, and do no attempt to give (much) insight in the origin of this difference. Other examples of papers with similar subjects where the same lack of substantation happens are Frank and Bouckaert (2006); Metsis et al. (2006); Wang and Manning (2012): all of them only report the results of different (and sometimes new) methods.

In addition, the publication of results without theoretical explanation leads to conflicting advise. For example, McCallum and Nigam (1998) report that using a Laplacian smoothing increases the performance of the MNB classifier. Schneider (2003) contradicts this and states that this smoothing decreases the performance. A paper from the same author which was published a year later, Schneider (2004) also investigates naive Bayes and surprisingly states that *"the parameters $p(w_t|c_j)$ are estimated from labeled training documents using maximum likelihood estimation with a Laplacean prior"* without any further reference to his findings from a year earlier: using this smoothing might decrease the performance. The lack of motivation of selected parameters in many machine learning papers may leave the reader confused as to why these implementations were chosen.

This leads to the suspicion that many papers do not attempt to motivate their model design choices. Different combinations of pre-processing, feature selection, Bag-of-Words transformations and naive Bayes parameters are evaluated or copied from previous work. As the performance is likely sensitive to these choices, as well as susceptible to changes in the data set, good performances can often be found which are subsequently published. These statements are of course speculative.

These goal oriented papers make performance valuations and comparisons hard. Kim et al. (2017) even employ a specific method of reporting to increase the apparent performance: they optimize their newly invented algorithm for each classification task separately and subsequently compare it to the average performance of benchmark methods. This of course gives an unrealistic picture of the performance of their own machine learning algorithm (which is, comically, Bag-of-Concepts).

Since the performance of a classifier seems hard to explain or predict, the author of this thesis recognizes that a grid search over different options in the text classification algorithm gives faster and likely better results than looking for a theoretical optimum. It is suspected that for this reason, some authors chose this approach to get striking results.

Because of all these reasons, it is hard to draw a definite conclusion regarding the performance of BoW and BoC in comparison with other literature. It is unsure whether the results in this thesis *really* differ from results in literature, or that simply less optimization steps have been executed. Perhaps after optimizing different options in the text classification process or using different data sets, results similar to that of other literature would have been found.

6 Conclusion

Using the results from Sections 5.1 and 5.2, we answer our first subquestion: *How do the Bag-of-Words (with different TFIDF weightings) and Bag-of-Concepts feature representations compare in performance?* We have seen that the integer BoW representation is the most reliable BoW representation in all of our data sets, with the Boolean representation being a good second choice. TFIDF transformations decrease the performance of our classifier with the pre-processing steps and data sets used for this research. The BoC representation in general performs a couple of percent points worse than the BoW representation, as measured by F1-score. However, a large dimension reduction of the feature representation is possible using BoC.

Using Sections 5.3, 5.4 and 5.5, we answer our second subquestion: *How comprehensible are these methods for humans (or: how well can the results be explained)?* In Section 5.3 we have shown that there is a correlation in the frequency of a word and the importance of it in a NB model. In Section 5.4 we have investigated the words which make up important concepts for the labels in the Gaming data set. The investigated concepts, as well as trends in *Probability (rank)-Frequency (rank)*-plots do show some interesting insights, but they do not explain the F1-score of the classifiers. Even though the NB models are theoretically simple, it is a stretch to call the BoC-feature representations comprehensible by humans. It is possible to get insight in the conditional log probabilities of individual features, but as the number of features increases, it becomes intractable to inspect all of them.

In a similar vein, the five most important words in the NB model using the BoW feature representations do not explain the difference in F1-scores. As there are ten-thousands of words in each vocabulary, of which a portion should be investigated to get an understanding of the model, it is hard to make the model fully comprehensible for humans in an easy manner. This does not necessarily mean that the model is a black box and understanding is impossible. The excess of relevant information that should be taken into account in the model causes that a concise representation of this model which is understandable by humans is impossible.

From Section 5.4, we can also answer our last subquestion *Can Bag-of-Concepts be used as a tool in the keywords generation process for an e-discovery case?* After training a NB model with our BoC feature representation, we can find clusters which are relevant for our class. Examples given are the animals or monsters in the game Minecraft or champions in League of Legends. If the goal of an e-discovery case is to find all documents that are relevant for League of Legends and we can only search using positive or negative keywords, including the names of all League of Legends champions as given by this cluster of words might prove to be very useful. The clusters resulting from the BoC making use of by us trained the `word2vec` model do seem to contain more relevant words than clusters that use Google's `word2vec` model.

If important words of a certain class have to be found, it can be useful to investigate the words with the highest probabilities in the MNB model using the BoW representation. It is even more interesting to investigate the clusters from BoC with the highest conditional probabilities in the MNB model. When training `word2vec` on our data sets, these concepts show relevant keywords that associate with the labels of our interest. This method can very well be incorporated in current e-discovery practices. Using Google's pre-trained `word2vec` model seems to be less suited for this task.

With these answers, we can answer our main research question: *'How does Bag-of-Words (with different TFIDF weightings) compare to Bag-of-Concepts?'*

The performance, when measured using F1-score, of BoWs is in general better than the performance of BoCs, with some exceptions. TFIDF transformations on BoW do not help to improve the performance, nor does using Boolean attributes. This is in conflict with literature, which reports that a Boolean feature representations as well as TFIDF transformations improve the performance of the BoW feature representation. On the data sets used in this thesis, the R52 data set and the questions on two StackExchange boards this improvement is not found. It is hypothesized that the discrepancy between the results of this thesis and literature is caused by a lack of feature selection and the absence of parameter optimization within our models. Nevertheless, this thesis was not able to give a waterproof explanation.

To investigate the understandability of the investigated models, the most important features of BoC and BoW have been inspected. While both BoW and BoC feature representation are, in combination with MNB, theoretically relatively simple, direct and clear explanation for the performance (difference of) the models could not be given.

Additionally, although not the goal of this research, in Section 2 an explanation of the `word2vec` model has been given. While descriptions of `word2vec` are already scattered present in online blogs and (less frequent) in academic literature, the comprehensive description of the architecture of `word2vec` is a welcome addition to this research.

7 Future Research

Many things have been investigated in this thesis, but perhaps the scope of this research was too broad to answer some of our subquestions to full satisfaction. In this section, we shortly propose three possible suggestions to investigate the problems encountered in this research.

7.1 Sensitivity analysis of Bag-of-Words

Bag-of-Words has been used numerous times in many academic literature. Nevertheless, literature which investigates the effects of using TFIFD transformations on BoW is scarce. In combination with `word2vec` and Bag-of-Concepts, the scope of this thesis was too broad to deeply dive in the (theoretical) effect of TFIDF weighting. Fully investigating multiple forms of TFIDF, perhaps on a small data set such that the results are possible understandable for humans, would be a helpful addition to literature.

On a similar note, parameter tuning of BoW can be further investigated. The focus can be on the smoothing constant, but also on the generation of bigrams. The latter received little interest in this thesis, but is on itself a worthy research subject.

7.2 Further investigating Bag-of-Concepts in the use of e-discovery

The concepts resulting from Bag-of-Concepts consists of many interesting keywords. In this thesis, the keywords have qualitatively been assessed. It would be even more interesting to evaluate their performance in an actual e-discovery setting. An experiment which tackles this quantitatively could make use of naive classification. Certain clusters could be marked as important, and all documents containing at least one of these words are marked as the class *true*, while all other documents are marked as *false*. Using this classification method, the importance and relevance of words that make up clusters, or even individual words can quantitatively be assessed in a simple method. This experimental setup resembles the keyword searches currently used in e-discovery cases. In 2008 a contest hosted by Text REtrieval Conference (TREC; Oard and Baron (2008)) asked for good keywords for a legal case, and using BoC assisted keyword generation would perhaps be a good contender.

As BoC is a novel method the author expects that other improvements to this method are possible. Kim et al. (2017) reports using a Concept-Frequency Inverse Document Frequency (CFIDF), which is a transformation similar to TFIDF. Kim also reports that CFIDF increases the performance of BoCs. It is valuable to reproduce these results. Parameters in the BoC could also be subject to further investigation, for example, the embedded dimension size of the word vectors.

7.3 Predicting most important words in Bag-of-Words and Bag-of-Concepts

In this research, we have seen that the *Probability (rank)-Frequency (rank)*-plot of features of a MNB model carries helpful information. We have seen that the some of features with the highest probability (rank) are relevant for our class, but there are probably features that are even more strongly associated with our class than the ones with the highest probability rank. Insight regarding the relation between (to humans) the most important feature and their relation to the *Probability (rank)-Frequency (rank)*-plot is of great practical value in natural language processing.

Bibliography

Ana Cardoso-Cachopo (2007). Improving Methods for Single-label Text Categorization. *PhD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa*.

Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., and Spyropoulos, C. D. (2000). An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '00*, pages 160–167.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

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

Domingos, P. and Pazzani, M. (1997). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29(1):103–130.

Řehůřek, R. (2014). Making sense of word2vec. https://rare-technologies.com/making-sense-of-word2vec/.

Eyheramendy, S., Lewis, D. D., and Madigan, D. (2003). On the naive bayes model for text categorization. *AI & Statistics 2003: Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*.

Frank, E. and Bouckaert, R. R. (2006). Naive bayes for text classification with unbalanced classes. *PKDD'06 Proceedings of the 10th European conference on Principle and Practice of Knowledge Discovery in Databases*, pages 503–510.

Georgala, K., Kosmopoulos, A., and Paliouras, G. (2014). Spam Filtering: an Active Learning Approach using Incremental Clustering. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, page 23. ACM.

Goldberg, Y. and Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, (2):1–5.

Goossen, F., IJntema, W., Frasincar, F., Hogenboom, F., and Kaymak, U. (2011). News personalization using the cf-idf semantic recommender. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 10. ACM.

Gruner, R. H. (2008). Anatomy of a lawsuit. http://www.gruner.com/writings/AnatomyLawsuit.pdf.

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

IJntema, W., Goossen, F., Frasincar, F., and Hogenboom, F. (2010). Ontology-based news recommendation. In *Proceedings of the 2010 EDBT/ICDT Workshops*, page 16. ACM.

Joachims, T. (1997). A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. *the 14th International Conference on Machine Learning (ICML '97)*, pages 143–151.

Kim, H. K., Kim, H., and Cho, S. (2017). Bag-of-concepts: Comprehending document representation through clustering words in distributed representation. *Neurocomputing*, 266:336–352.

Kosmopoulos, A., Paliouras, G., and Androutsopoulos, I. (2008). Adaptive spam filtering using only naive bayes text classifiers. *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS)*, (1):1–3.

McCaffrey, J. (2015). Test run - k-means++ data clustering. *MSDN Magazine Blog*, 30(8). https://msdn.microsoft.com/en-us/magazine/mt185575.aspx.

McCallum, A. and Nigam, K. (1998). A Comparison of Event Models for Naive Bayes Text Classification. *AAAI/ICML-98 Workshop on Learning for Text Categorization*, pages 41–48.

McCormick, C. (2016a). Word2Vec Tutorial - The Skip-Gram Model. http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/.

McCormick, C. (2016b). Word2Vec Tutorial Part 2 - Negative Sampling. http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/.

Metsis, V., Androutsopoulos, I., and Paliouras, G. (2006). Spam Filtering with Naive Bayes – Which Naive Bayes? *Third Conference on Email and Anti-Spam (CEAS)*, 17:9.

Mikolov, T. (2013). Google Groups Forum: de-obfuscated Python + question. https://groups.google.com/forum/#!searchin/word2vec-toolkit/c-bow/word2vec-toolkit/NLvYXU99cAM/E5ld8LcDxlAJ, accessed on 9th of August 2018.

Mikolov, T., Corrado, G., Chen, K., and Dean, J. (2013a). Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pages 1–12.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed Representations of Words and Phrases and their Compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mitra, B., Nalisnick, E., Craswell, N., and Caruana, R. (2016). A Dual Embedding Space Model for Document Ranking. *arXiv preprint arXiv:1602.01137*.

Nicholas M. Pace, L. Z. (2012). *Where the Money Goes Understanding Litigant Expenditures for Producing Electronic Discovery*. RAND Corporation.

Oard, D. W. and Baron, J. R. (2008). Overview of the TREC 2008 Legal Track.

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*, 2825–2830(12).

Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Puurula, A. (2012). Combining modifications to multinomial naive bayes for text classification. In *Asia Information Retrieval Symposium*, pages 114–125. Springer.

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.

Rennie, J. D. M., Shih, L., Teevan, J., and Karger, D. R. (2003). Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In *Proceedings of the 20th international conference on machine learning (icml-03)*, number 1973.

Robertson, S. (2004). Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520.

Roitblat, H. L., Kershaw, A., and Oot, P. (2010). Document categorization in legal electronic discovery: computer classification vs. manual review. *Journal of the Association for Information Science and Technology*, 61(1):70–80.

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

Sarkar, S. D. (2013). Empirical Study on Filter based Feature Selection Methods for Text Classification. *International Journal of Computer Applications*, 81(6):38–43.

Schakel, A. M. J. and Wilson, B. J. (2015). Measuring Word Significance using Distributed Representations of Words. *arXiv preprint arXiv:1508.02297*.

Schneider, K.-M. (2003). A comparison of event models for naive bayes anti-spam e-mail filtering. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, pages 307–314. Association for Computational Linguistics.

Schneider, K.-M. (2004). On Word Frequency Information and Negative Evidence in Naive Bayes Text Classification. *Advances in Natural Language Processing*, pages 474–485.

Shindler, M. (2008). Approximation algorithms for the metric k-median problem. *Written Qualifying Exam Paper, University of California, Los Angeles. Cited on*, page 44.

Socher, R. (2015). Deep Learning for NLP, Lecture Notes CS 224D. https://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf.

Stokmans, D. and van Lonkhuyzen, L. (2018). U gaat frauderen. Dat zegt de computer. *NRC Handelsblad*, (11 May).

Wang, S. and Manning, C. (2012). Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 94305(July):90–94.

Appendices

A K-means++

K-means++ is an algorithm that finds good initial starting points for the k-means cluster algorithm, in the sense that the algorithm does not perform arbitrarily bad. A simple example of such an arbitrarily bad solution can be given with k = 2 and 4 data points, as shown in Figure 13. If we initialize the k-means algorithm with the given points in the figure, the k-means algorithm will converge in the first iteration. We can stretch the rectangle of the four data points as much as we like to obtain an arbitrarily bad solution.

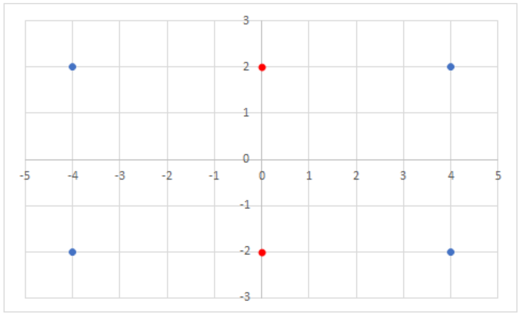


Figure 13: Example of a bad local optimal k-means solution. In blue the data points, in red the cluster centers. Convergence of k-means is achieved.

Arthur and Vassilvitskii (2007) proposed k-means++ to solve this problem. The k-means++ algorithm starts by picking an initial cluster at random from all the available data points. After this, it calculates the distance for all points to the closest cluster center, D(x). The subsequent starting clusters are chosen from the other data points, with a weighted probability distribution. The probability that data point x is chosen as starting cluster is proportional to $D(x)^2$. The calculation of D(x) and selecting starting clusters is repeated until k initial clusters are chosen. This is illustrated in Figure 14.

In a literature survey by Shindler (2008), k-means++ has shown good performance in defining initial cluster-centers for k-means clustering.

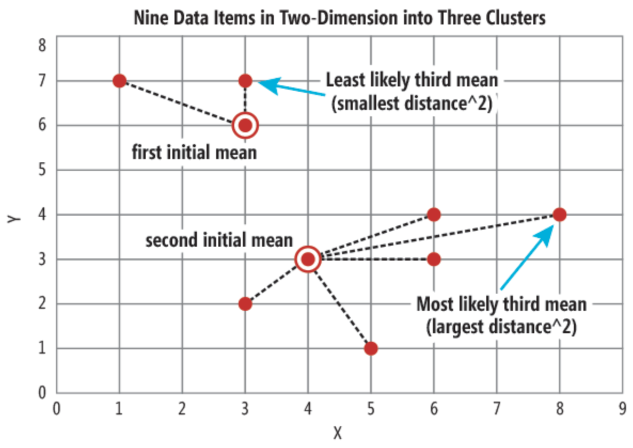


Figure 14: K-means++ initialization mechanism illustrated. Figure from McCaffrey (2015).

B Accuracy of Bag-of-Words

Table 13: Accuracy scores of different types of Bag-of-Words in the Gaming data set.

	Integer	Boolean	SK-TFIDF	log TFIDF
minecraft	**0.95**	0.95	0.87	0.95
the-elder-scrolls-5-skyrim	0.97	**0.97**	0.89	0.96
league-of-legends	0.98	**0.98**	0.93	0.98
diablo-3	0.96	**0.96**	0.91	0.96

Table 14: Accuracy scores of different types of Bag-of-Words in the English data set.

	Integer	Boolean	SK-TFIDF	log TFIDF
single-word-requests	**0.85**	0.85	0.78	0.81
meaning	0.87	0.87	**0.87**	0.87
grammar	0.88	**0.89**	0.89	0.88
word-choice	0.92	0.92	**0.92**	0.92

Table 15: Accuracy scores of different types of Bag-of-Words in the Reuters data set.

	Integer	Boolean	SK-TFIDF	log TFIDF
earn	0.98	**0.98**	0.98	0.97
acq	0.98	**0.99**	0.9	0.97
crude	**0.99**	0.99	0.96	0.98
trade	0.97	0.96	0.97	**0.98**

Table 16: Accuracy scores of different types of Bag-of-Words in the Balanced data sets.

	Integer	Boolean	SK-TFIDF	log TFIDF
minecraft	**0.76**	0.71	0.67	0.68
single-word-requests	**0.77**	0.75	0.73	0.72
earn	**0.98**	0.98	0.98	0.98

C F1-scores of Random Forest with Bag-of-Words

Below, F1-scores for the binary classification tasks using a random forest are shown. The random forest implementation of `scikit-learn` was used, using the default parameters and a maximum depth of 100. As the focus of this research lies on NB, further investigation is not performed.

Table 17: F1 scores of different types of Bag-of-Words in the Gaming data set using a Random Forest classifier.

	Integer	Boolean	SK-TFIDF	log TFIDF
minecraft	0.71	0.75	**0.76**	0.75
the-elder-scrolls-5-skyrim	**0.8**	0.74	0.78	0.8
league-of-legends	0.78	**0.8**	0.76	0.79
diablo-3	**0.67**	0.54	0.64	0.57

Table 18: F1 scores of different types of Bag-of-Words in the English data set using a Random Forest classifier.

	Integer	Boolean	SK-TFIDF	log TFIDF
single-word-requests	0.42	0.37	0.46	**0.54**
meaning	0.01	0.05	**0.06**	0.05
grammar	0.07	0.05	0.05	**0.08**
word-choice	0	0	**0.01**	0

Table 19: F1 scores of different types of Bag-of-Words in the Reuters data set using a Random Forest classifier.

	Integer	Boolean	SK-TFIDF	log TFIDF
earn	**0.97**	0.97	0.97	0.97
acq	0.91	0.91	0.9	**0.92**
crude	0.72	**0.79**	0.76	0.73
trade	0.57	0.52	**0.69**	0.55

Table 20: F1 scores of different types of Bag-of-Words in the Balanced data sets using a Random Forest classifier.

	Integer	Boolean	SK-TFIDF	log TFIDF
minecraft	**0.48**	0.36	0.44	0.36
single-word-requests	**0.56**	0.55	0.55	0.56
earn	0.97	**0.98**	0.97	0.97

D Proof that Euclidean similarity is equivalent to a linear transformation of cosine similarity

Cosine similarity is defined as $\frac{\sum x_i y_i}{\sqrt{\sum x_i^2 \sum y_i^2}}$ and for normalized vectors $\sum x_i^2 = \sum y_i^2 = 1$ holds. Therefore, cosine similarity for normalized vectors can be defined as $\sum x_i y_i$.

$$\begin{aligned}
\|x - y\|^2 &= \sum (x_i - y_i)^2 \\
&= \sum (x_i^2 + y_i^2 - 2x_i y_i) \\
&= \sum x_i^2 + \sum y_i^2 - 2\sum x_i y_i \\
&= 1 + 1 - 2\cos(x, y) \\
&= 2(1 - \cos(x, y))
\end{aligned}$$

This shows that for normalized vectors, the Euclidean similarity (and thus distance) is a linear transformation of the cosine similarity.

E Words in BoC clusters from the label *'the-elder-scrolls-5-skyrim'*

The words in the second most important cluster resulting from training `word2vec` on the Gaming data set, for the label *'the-elder-scrolls-5-skyrim*. Cluster (2,2):

```
['alternative', 'alternatives', 'anyways', 'apart', 'approach', 'appropriate', 'aren', 'awesome',
'basically', 'biggest', 'bother', 'certainly', 'cool', 'course', 'cut', 'decide', 'don',
'efficiently', 'effort', 'expect', 'goal', 'good', 'great', 'hassle', 'ideally', 'individually',
'intend', 'involve', 'kind', 'kinds', 'knowing', 'lack', 'learn', 'like', 'love', 'make', 'making',
'necessary', 'need', 'nice', 'obviously', 'ones', 'pick', 'picking', 'planning', 'practical',
```

'prefer', 'proper', 'putting', 'quickest', 'reliable', 'risk', 'route', 'safe', 'sense', 'shouldn', 'techniques', 'tedious', 'tend', 'things', 'thinking', 'ton', 'tricks', 'understand', 'unless', 'useful', 'want', 'willing', 'wonder', 'worry']

The words in the third most important cluster resulting from training `word2vec` on the Gaming data set, for the label *'the-elder-scrolls-5-skyrim*. Cluster (3,3):

['actually', 'barely', 'brief', 'case', 'cause', 'close', 'comes', 'consuming', 'counted', 'cycle', 'double', 'end', 'eventually', 'far', 'gets', 'giving', 'goes', 'half', 'happen', 'happens', 'immediately', 'indefinitely', 'just', 'lasts', 'later', 'll', 'long', 'longer', 'lose', 'means', 'million', 'mins', 'minute', 'minutes', 'moment', 'normally', 'period', 'periods', 'plus', 'point', 'real', 'reload', 'remaining', 'rest', 'round', 'runs', 'second', 'seven', 'short', 'stay', 'successful', 'taken', 'takes', 'taking', 'till', 'time', 'times', 'twice', 'wait', 'wasted']

The words in the second most important cluster resulting from Google's `word2vec` on the Gaming data set, for the label *'the-elder-scrolls-5-skyrim*. Cluster (2,2):

['advised', 'afford', 'agreed', 'allow', 'allowed', 'allowing', 'ask', 'asked', 'asking', 'choose', 'choosing', 'chose', 'dare', 'decide', 'decided', 'deciding', 'demanded', 'did', 'encourage', 'expect', 'fail', 'hesitate', 'hurry', 'inclined', 'intend', 'invite', 'let', 'looking', 'need', 'needed', 'needing', 'order', 'ordered', 'permission', 'prefer', 'preferred', 'pressed', 'recommend', 'recommended', 'recommending', 'refuse', 'refused', 'request', 'requested', 'require', 'required', 'requiring', 'seek', 'seeking', 'shall', 'supposed', 'voluntarily', 'want', 'wanted', 'wanting', 'welcome', 'willing', 'wish']