Using N-grams and Word Embeddings for Twitter Hashtag Suggestion

Lucas Vergeest
Tilburg University (School of Humanities)
Master Track: Human Aspects of Information Technology
Thesis supervisor: Grzegorz Chrupala
Second reader: Pieter Spronck

April 25, 2014

Abstract

Many approaches have been taken in prior research to develop accurate Twitter hashtag suggestion systems. Most of these systems involve longer known NLP-techniques like topic distribution and tf-idf representations. However, none of these approaches make use of the more state-of-the-art machine learning techniques involving artificial neural networks (ANNs).

This thesis examines the possibility of using word embeddings features, which are trained using an ANN-approach, to develop a system which suggests appropriate hashtags for given tweets. The experiments demonstrated that adding word embeddings to the n-gram baseline indeed lead to a better overall performance of the system. For the best performing baseline (character 3-grams), the F1-score went up from 0.17 to 0.18 when adding word embeddings features. For word unigrams, the F1-score went op from 0.07 to 0.13 when adding word embeddings features.

Acknowledgments

First of all, I would like to thank Dr. Grzegorz Chrupala, my thesis supervisor, for offering me the opportunity to write this thesis in the first place and for supporting me throughout the entire thesis writing process. He was always very patient when trying to explain certain concepts related to the subject. Moreover, he often helped me out on the technical side, when I got stuck during one of my experiments. Without his help, I could have never fulfilled my work within this time frame.

Furthermore, I want to thank my family and friends for their involvement with my thesis progress, and acknowledging the relevance of the subject. This has been really encouraging to me.

I especially want to thank my fellow student Sander Maijers for sometimes giving me advice on technical issues, during, and also before the writing of this thesis, as he did before for my Bachelor thesis. He has motivated me to head in a more technical direction during my study and showed me the great benefits of having a decent amount of programming knowledge.

In relation to that, I would like to thank the Stack Overflow community for regularly helping me out when I got stuck during programming.

Finally, I want to thank Tilburg University in general for helping me develop new research skills and other relevant knowledge during my current Master track Human Aspects of Information Technology, which proved to be valuable during the thesis writing process.

Contents

1 Introduction

1.1 Introduction

In recent years, artificial neural networks (ANNs) have become a focus of much research in the field of computer science. Today, it is used for a lot of tasks already, like face detection/recognition, weather forecasts and speech recognition. Due to its demonstrated success, this technique is spreading to more and more research fields (Dayhoff & DeLeo, 2001), including computational linguistics. This has motivated me to explore the existing techniques, and to evaluate such a model for an NLP-related task. The task that was chosen for this research is hashtag recommendation. The reason for this choice is that it seems to be a very useful task which could be implemented in many practical applications. On a daily basis, millions of people use Twitter, Facebook, Google+, Instagram and other social networks which enable the use of hashtags, usually eased by the user interface, for instance by making these hashtags clickable. These users could save time and effort when some kind of suggestion system would be implemented in these applications. This would also add to the value of the entire social network, since finding meaningful tweets and discovering trending topics will become easier for users. Some research has already been done on hashtag recommendation systems. However, the amount of research on using ANN-based techniques for hashtag recommendatory is still relatively limited. Since ANN-based techniques seem to yield good results for a lot of NLP tasks (ie. speech recognition), it could be interesting for the research field of computational linguistics to shift attention increasingly towards this area. With this thesis, I hope to contribute to this shift in attention.

1.1.1 Research Question

The research question in this thesis will be formulated as follows: Are word embeddings features, learned with an ANN-based learning technique suitable as a basis for hashtag suggestion systems? And more specifically: Does it yield better results than using just a baseline n-gram classifier? The goal of such a system would be to suggest the most suitable hashtag for any given tweet.

In my experiments I found that adding word embeddings features to an n-gram baseline indeed leads to a better performance of the classifier. For the best performing baseline condition (character trigrams), the F-score went up from 0.17 to 0.18. These results seem promising and could possibly encourage further research on word embeddings features. Furthermore, this thesis provides an extensive overview of many of the techniques that have been used in past research on Twitter hashtag recommendation.

1.2 Thesis Structure

In this chapter, I formulated my research questions, my motivations and the main contributions of this thesis. This section will describe the structure of the rest of this thesis. Chapter 2 will start with an explanation of what Twitter is and what hashtags are. Then I will zoom in on the different approaches that have already been taken in developing Twitter hashtag suggestion systems. Chapter 3 will explain logistic regression and multinomial logistic regression, the statistical classification model that is used by the classifier in my experiments. In Chapter 4, I will go in more detail about the different classification techniques that are used in my experiments, describing both n-gram-based classification, which I used as my baseline, and word embeddings-based classification. I will also give an explanation on the functioning of artificial neural networks (ANNs). In Chapter 5, I describe all the details regarding my experiments: the used corpus, pre- and post-processing, evaluation metrics and a presentation of the results. I will finish this section with a discussion of the results. In Chapter 6, I will draw conclusions and offer some suggestions for future research.

2 Background

In this section, I will discuss what Twitter is, what hashtags are and what they are used for. Then I will discuss previous relevant research about hashtag recommendation systems. Many different approaches have been taken by different researchers. The approaches that will be described in this chapter are topic distribution, semantical similarity between words, personal user preferences, tf-idf representations and a topical translation model.

2.1 Twitter

Twitter is an inter-platform social medium which allows the user to exchange short pieces of text (called tweets) with other users in their network. These tweets can have a maximum of 140 characters. As of 2013, Twitter had more than 554 million active users (Javed & Afzal, 2013). According to Alexa, currently (February 2014) Twitter is the 10th most visited website on the web [2].
Twitter has its own meta-markup. For instance, the symbol '@', followed by a Twitter user name, indicates that the tweet is directed to that user, or that this user is mentioned in the tweet. As mentioned earlier, Twitter is available for multiple platforms, including PCs, smartphones and tablets.

2.1.1 Hashtags

The symbol '#' followed by a word transforms that word into a so-called hashtag. These hashtags are usually used to indicate the subject, location, conference or recent news event to which a particular tweet is related. This eases the search for tweets that concern a certain topic. The concept of the hashtag has proven successful in Twitter. According to Hong et al. (2011), hashtags are used in 14% of English language tweets. Its proven usefulness has inspired other social networks like Facebook, Google+ and Instagram to implement hashtags in their systems as well.

2.2 Hashtag Recommendation

In recent years, much research regarding hashtag recommendation on Twitter has already been done. Researchers have used different approaches for such systems.

2.2.1 Systems based on Topic Distribution

Godin et al. (2013) propose a method for hashtag recommendation. It is a unsupervised system, in which the classification is based purely on the content. Their approach relies on Latent Dirichlet Allocation (LDA). This system makes use of a topic distribution.

LDA is a hidden topic model. This model assumes that there is a limited set of *T* topics where a document (in this case a tweet) can be about. A topic distribution over these topics will be assigned to each tweet. In the end, the topic which has the highest value assigned to it, will be selected by the system as the valid topic for that tweet. After the topic is assigned to a tweet, five hashtags will be selected. The selection was done by using topic-term count values, determining the top words of every topic in ranked order. These top words were then converted to hashtags. Because the researchers considered evaluation a subjective task, two persons were used to independently evaluate the applicability of the suggested hashtags. In cases of disagreement, there was a discussion until agreement was reached. For the evaluation, 100 tweets were randomly picked from the original set, which contained 1.8M tweets. For 80 of the 100 tweets in the test set, at least one appropriate hashtag could be selected by the judges, out of a set of five hashtag suggestions. When using a set of ten hashtags instead of five, accuracy increased to 91 appropriate suggestions.

2.2.2 Systems based on tf-idf Representations

Zangerle et al. (2011) use tf-idf based representations of tweets, comparing each tweet to other similar tweets which already include hashtags. The formulas used for calculating the tf-idf are

(1)
$$tf\,idf_{t,d} = tf_{t,d} \times idf_t$$

(2)
$$tf_{t,d} = n_{t,d}$$

(3)
$$idf_t = \log \frac{|D|}{|\{d : t \in d\}|}$$

Here t refers to the number of occurrences of a term within a given tweet d. The inverse document frequency (idf) represents the relative relevance of a term t within the entire set of searched documents (D). It can be calculated by dividing the total number of documents ($|D|$) by the number of documents that contain the searched term. The tf/idf-score for a given tweet will then be calculated by taking the sum of the tf/idf's of all separate terms in the tweet. The more terms are matched between tweets, the higher the ultimate tf/idf-score will be. Only tweets with a score above a predetermined threshold will be matched to each other. Additionally a limit for the possible total number of results is set. All the hashtags are then extracted from this set which contains tweets that are most similar to the given tweet. The five or ten most 'appropriate' hashtags were then presented to the user. Three different approaches for the ranking of hashtags were tried. (1) Overall popularity of the hashtag on Twitter, (2) frequency counts of the specific hashtags in the similarity set and (3) similarity between the source tweet and the tweet in which the candidate hashtag appears. This last approach turned out to yield the most appropriate hashtag ranks.

They used a corpus of around 16M tweets for their experiment. The steps they took for determining tweet similarity are the following: (1) find the most similar tweets in the dataset for any given tweet, (2) retrieve the set of hashtags used in these tweets (3) calculate ranked scores for all candidate-hashtags. They performed a leave-one-out test on the resulting suggestions. They used both precision and recall to evaluate the quality of the hashtag suggestions. As mentioned before, hashtag ranking based on tweet similarity yielded the best results (approximately 45% recall and 15% precision). The recall in this research is higher (45% vs. 30%) than in the research of Kywe et al. (2011) (Section 2.2.4). Possibly this can partly be explained by the fact that a larger corpus was used for training (about seven times as large).

2.2.3 Systems based on the Semantical Similarity between Words

Pöschko (2011) has looked at hashtag co-occurrences. He argues that hashtags which are semantically related, will co-occur more often in tweets. Knowledge of hashtag similarities can also help to assign appropriate hashtags to tweets. His idea is that the intensity of a relation between two given words can be measured by calculating the shortest path between them, using a thesaurus like WordNet. However, a problem is that many hashtags would not appear as a word in WordNet, since they refer to names that consist of multiple words with unclear word boundaries. This could be solved though by simply ignoring hashtags which are not represented as words in WordNet. Pöschko bases the relation strength between two words on the relation between the two synsets (groups of synonymous lemmas) in which they occur.

$$S(h_1, h_2) := \max_s(s_1, s_2)$$

Here, S represents the similarity between two hashtags (h_1, h_2) and the two corresponding words (s_1, s_2) in the thesaurus respectively.

For calculating the similarity between two words, two similarity metrics were used:

(1) the path distance similarity, defined as

$$S_{path}(s_1,s_2) := \frac{1}{d(s_1,s_2)+1}$$

(2) the Wu-Palmer distance, defined as

$$S_{WP}(s_1,s_2) := \frac{2d(s(s_1,s_2))}{d(s_1)+d(s_2)}$$

where $s(s_1,s_2)$ represents the lowest common subsumer of s_1, s_2, and *d(s)* represents the depth of synset s in the taxonomy. The depth value $d(s_1,s_2)$ essentially represents the number of steps it would take in the web application of WordNet to get from one term to another. For instance, the distance 'house - hermitage' is 1, since they are directly linked as 'sister terms'.
The Wu-Palmer distance (2) is a metric developed by two eponymous computer scientists (Wu & Palmer, 1994). The idea behind this metric is that word similarity calculation should be based on sentence context and prior knowledge. They used this approach for improving word selection algorithms in machine translation. They introduce a formula to calculate concept similarity, with the specific aim of enabling verb similarity calculation.

$$ConSim(C_1,C_2) = \frac{2\times N_3}{N_1+N_2+2\times N_3}$$

in which C_1 and C_2 are the two concepts, and N_1+N_2 are the total number of nodes on the path from C_1 to C_2. N_3 is the number of nodes to the root.

Now the similarity between two word meanings can be calculated summing all the weighted similarities between pairs of simpler (more concrete) concepts in each of the domains where the words relate to. Pöschko did not develop a recommendation system for hashtags, however, he used a maximum entropy (MaxEnt) classifier to cluster hashtags in several tag categories, like 'geolocation', 'person', and 'organization'. This classifier is based on a multinomial logistic regression model (also known as softmax regression, explained in more detail in Section 3.2). MaxEnt models are often used as an alternative to Naive Bayes classifiers (NBCs), because in contrast to NBCs, MaxEnt models do not assume statistical independence of the used features. In many NLP-task, the used features are not or only partly independent, hence the advantage of MaxEnt classification.

Li et al. (2011) also incorporate similarities between words as an addition to tf-idf based ranking, using an Euclidean distance metric to calculate the similarity between tweets.

Their view is similar to that of Pöschko, but instead of simply counting word co-occurrences, they base their similarity judgments entirely on WordNet. Based on the found distances between words, the system predicts the hashtags. It collects a few of the most similar tweets, extracts their hashtags and then decides which hashtag to pick, based on tag ratios. When one tag gets a ratio higher than 50%, it will be chosen as the predicted tag. The system ended up predicting around 86% of hashtags correctly.

Antenucci et al. (2011) also aim to cluster hashtags in meaningful topic groups, using both co-occurrence frequencies and text similarity as features. After that, they use a Principal component analysis (PCA), dimensionality reduction and some multi-class classification algorithms to classify tweets. They used a data set of 178M tweets. First they removed all tweets containing non-ASCII characters plus all tweets that did not contain hashtags, which left them with about 16 million tweets. They noticed that less than 10 percent of all the tweets in their dataset contained at least one hashtag. However, they considered this sufficient to do meaningful training. They decided to leave out some hashtags in their training data, like #fb (meaning facebook) and #followfriday, because they considered these tags as being too general.

They created a co-occurrence list for 2000 most frequently used hashtags. Just like Pöschko, they used the Natural Language Toolkit (NLTK) to calculate Wu-Palmer distances between hashtags. For co-occurring hashtags, the value was around 0.40, for random hashtag pairs, the value was around 0.16, similar to the results that Pöschko found in his research.

They tested several similarity measures for determining the similarities between hashtags. They used the one that proved to give the best results:

$$S(A,B) = \left(\frac{n_{AB}}{\sum n_{Aj}} + \frac{n_{BA}}{\sum n_{Bj}}\right)/2$$

Here, $S(A,B)$ denotes the similarity value for hashtag A and B. n_{AB} represents the number of occurrences of B given A, and n_{BA} represents the number of occurrences of A given B. These are divided by the sum of total occurrences of these tags, similar as in tf-idf ranking.

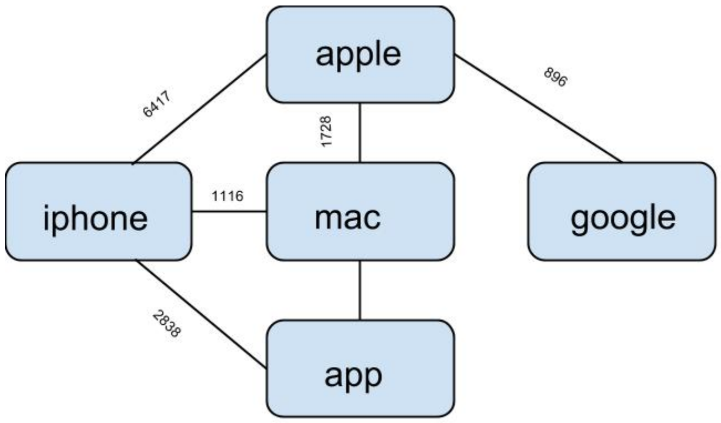

Figure 1: Example of an undirected hashtag similarity graph

Figure 1 shows one of the similarity graphs that were proposed, based on the generated co-occurrence lists. They tried several clustering methods. Spectral clustering yielded the best results. They manually evaluated how much sense the generated hashtag clusters made to them. When using spectral clustering, an average rating of 3.32 on a Likert-scale was achieved.

They used a Twitter corpus (only including tweets with hashtags) for mapping the tweets to tag clusters. They divided the corpus in 90% training data and 10% test data. The aim was not to find the most appropriate hashtag for a given tweet, but to map it to the most appropriate hashtag cluster. Existing multi-class classification algorithms were used for classification. These were Bernoulli Naive Bayes, LDA, SVM (support vector machine), majority vote, and SVM without PCA (principal component analysis). The Bernoulli Naive Bayes classifier combined with the spectrally clustered tag clusters turned out to yield the best results.

The F1-score was 0.27 for this combination. This technique was closely followed by SVM (including PCA), with an F1 of 0.25.

The F1-score is a measure that represents the accuracy of a classification model. It is obtained by calculating the harmonic mean of precision and recall. An F1 of 0 means that none of the classifications are valid, while an F1 of 1 would mean that all classifications are valid, and that all items that should be classified are actually classified (reflected by the recall). These three metrics are explained in more detail in Section 5.1.3.

2.2.4 Systems Incorporating Personal User Preferences

Kywe et al. (2011) combine tweet content with personal user preferences in their hashtag recommendation system. Their method consists of three steps: (1) extracting hashtags from user accounts with high similarity to target user, (2) extracting hashtags from tweets with high similarity to target tweet and (3) calculating ranked scores for all candidate-hashtags. A tf-idf (term-frequency-inverse document frequency) scheme was used for the first two steps. For determining user similarity, the similarity between the used hashtags was considered. The similarity between tweets was determined using a weighted vector based on words in a vocabulary, similar to the approach of Antenucci et al. (2011). They used a data set of approximately 2.3 million tweets as training data. They used 5,600 tweets from the original set as their test set. For each target user-tweet pair, they tried both top five and top ten hashtag suggestions and measured their performance, using the so-called *hit rate*, where

$$\text{Hit Rate} = \frac{\text{number of hits}}{\text{number of target user-tweet pairs}}$$

Something is considered a hit when at least one of the top recommended hashtags for a given tweet matches with an actual hashtag that was used in that tweet. Their research showed that incorporating user preferences indeed lead to a better performance of the system. A hit rate of 37.19% was achieved when using the 50 most similar tweets and the five most similar users.

2.2.5 Systems based on a Topical Translation Model

Ding et al. (2013) have developed a hashtag suggestion system based on a topical translation model. Their approximation of the problem evolves around the assumption that the content of the tweet and the descriptive meta-data (hashtag) are two different languages describing the same themes. In order to find the most appropriate hashtags, they ’translate’ the tweet content to the ’meta-language’. The LDA (Latent Dirichlet Allocation)-model is a topic model which assumes that each document is a mixture of a limited amount of topics. Out of the single words in a document, one can deduce the original topic(s). Existing LDA-techniques have been improved to make them more suitable for shorter documents like tweets.

In the research of Ding et al. (2013) a topical translation model, specifically designed for microblogs, is used to determine the topics of tweets. Standard LDA has proved to be less suitable for microblog topic analysis, because of the limited length of the documents. While LDA assumes a mixture of topics for a document, the Topical Translation Model assumes just one single topic. For unlabeled tweets, first the topic is determined. Based on the topic, the topic/background words are then determined.

The researchers use topic-specific word triggering to enable the translation step from tweet to hashtag. Topic-specific word triggering encompasses obtaining a set of words (in the case of this research five words) that are triggered by a certain topic. For instance, the word ’apple’ can trigger two different sets of topic words (related to the topic ’fruit’ or related to the topic ’technology’). This topic is determined by looking at the words in a tweet. Collapsed Gibbs sampling was used to calculate the sampling probability for a word to be a topic or a background word.

The top five triggered word are then used as hashtags suggestions. In this way, the model can suggest hashtags that do not appear as a word in the tweet. This method performed better than the other methods they tested. Standard LDA (F1 = 0.15) and Naive Bayes (F1 = 0.18) performed the worst. The TTM (Topical Translation Model) they used performed best, scoring approximately F1 = 0.36, when suggesting just one hashtag for each tweet.

Just like Ding et al. (2013), Liu, Chen and Sun (2011) use a word trigger method for their tag suggestion system. In this research, word alignment techniques are used to align the content (books and bibtex entries) with the meta-content (tags).

In IBM Model-1, the relationship between the source language $w = w_1^J$ and the target language $t = t_1^I$ is determined by a hidden variable, which describes the shortest route from source position j to target position a_j. The word alignments are represented by a_1^J. The alignment a_1^J includes empty word alignments ($a_j = 0$) as well. These empty alignments align source words to 'empty words' in the target language.

$$Pr(w_1^J|t_1^I) = \sum_{a_1^J} Pr(w_1^J, a_1^J|t_1^I)$$

Once a list op candidate hashtags is obtained, they are ranked by relevance by computing their relevance scores:

$$Pr(t|d = w_d) = \sum_{w \in w_d} Pr(t|w)Pr(w|d)$$

In this formula, $Pr(w|d)$ represents the 'trigger power' of a word w in the tweet. This trigger power indicates the importance of the word for the tweet. This system yielded results similar to those of Ding et al. (2013), with F1-scores topping at 0.37 when suggesting two tags per document.

2.3 Summary

In this chapter I gave an overview of the state-of-the-art of Twitter hashtag suggestion systems by mentioning five different approaches that have been taken so far. Some of these systems seem to perform better than others. Of these systems, the studies that used a topical translation model tended to yield the best results, with F1-scores of 0.37 and 0.38. However, it has to be noted that results from experiments that use different datasets can not really be compared.

3 Logistic Regression

The one-vs-the-rest classifier that I used during my experiments, makes use of logistic regression during training. In this section, I will go into more detail about what logistic regression and multinomial logistic regression are.

3.1 Logistic Regression

Logistic regression often plays an important part in data analysis describing the relation between an outcome variable and one or multiple predictor variables (features) (Hosmer, Lemeshow, & Sturdivant, 2013). The outcome variable is often discrete and takes one, two or more possible values. The goal of logistic regression is to find the best fitting and interpretable model to describe the relationship between an outcome variable and its predictor variables (features). Linear regression is used when trying to predict real-valued outcomes. In logistic regression - in contrast to linear regression - outcome variables are binary, which means they can only have two states. The relationship between the outcome (dependent) variable and one or more predictor (independent) variables is measured by using probability scores. Usually, these independent variables are continuous.

NLP-related problems are often classification problems. The output that is predicted can only have a limited number of discrete values (Jurafsky & Martin, 2000, p. 199). Binary classification is the simplest form of classification, judging whether a given object x falls in a class or not. In addition to only making this binary judgments, one would like the classifier to output probability values for the possibilities of object x belonging or not belonging to a certain class. The function that is used during logistic regression is called the logit function.

$$ln\left(\frac{p(y=true|x)}{1-p(y=true|x)}\right) = w \times f$$

Here y represents the binary value (1 or 0), x is the training observation, w represents the weight vector and f the feature vector. The weight vector w will be trained to maximize the probability of the observed y values in the training data, given the observations x. These optimum weights will be calculated for the entire training set:

$$\hat{w} = \arg\max_{w} \prod_{i} P(y^{(i)}|x^{(i)})$$

Classification is based on probability scores: the class with the highest probability score will be selected by the model as the right class.
Logistic functions are often used in neural network-based learning algorithms, like word representations vectorizers as described in Section 4.2. Often the nodes compute a linear combination of the input signals. After that, a logistic function is applied. The function that is used in the word representations vectorizer that I used, is described in Section 4.2.7.

3.2 Multinomial Logistic Regression

Multinomial logistic regression (or softmax regression) is a variant of logistic regression that - in contrast to standard logistic regression - allows more than two discrete outcomes. This model is used in classification situations where there are more than two classes, as is the case in many NLP-problems (Jurafsky & Martin, 2000, p. 203). This model can predict probability values for different possible outcomes of categorically distributed dependent variables. The aim of softmax regression is to predict categorical data. Softmax-based classifiers (also called maximum entropy models or MaxEnt models) are often used as an alternative to naive Bayes classifiers. In contrast to naive Bayes classifiers, MaxEnt models do not assume statistical independence of the features, which may often be an advantage in NLP-related classification tasks, since in practice, different features tend to correlate with each other. On the other hand, incorporating these dependencies also means higher computation costs. In a MaxEnt classifier, the weights are learned by using a repetitive procedure. The probability of y belonging to a particular class is calculated as follows:

$$p(c|x) = \frac{1}{Z} \exp\left(\sum_i w_i f_i\right)$$

Here, c represents the class. Z is the normalization factor. MaxEnt classifiers normally give a probability distribution over all the possible classes. If hard classification is required (choosing the single most probable class), the system can simply pick the class with the highest probability attached to it

$$\hat{c} = \arg\max_{c \in C} P(c|x)$$

The maximum entropy model is based on the principle of maximum entropy (MaxEnt). This principle states that the probability distribution with the largest entropy represents the current state of knowledge best. Entropy is a measure to express the unpredictability of information. The more equally distributed the probabilities in a probability distribution are, the higher the entropy. In other words: of all possible distributions, the equiprobable distribution has the maximum entropy (Jurafsky & Martin, 2000, p. 208). Therefore, given a set of four variables, the distribution $\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$ has the maximum entropy. The main idea behind MaxEnt models is that it should follow all constrains imposed on it, but should further make as few assumptions as possible (principle of Occam's Razor). Theoretically, an unlimited number of constraints can be added to the model. For a part-of-speech tagger, an example of a constraint would be: 'in this corpus, the chance for a phrase to be an NP is twice as large as the chance of a phrase being a VP'. As a consequence, the probability distribution for this model would be $NP = \frac{2}{3}$ and $VP = \frac{1}{3}$.
As mentioned before, the MaxEnt model is useful in cases where the outcome variables are nominal (categorical). This means that it falls in one of a set categories that can not be numerically ordered in a meaningful way. The hashtags that are handled in my experiments, are a good example of such nominal variables. Another example would be: what is the blood type (dependent outcome

variable) of person *A*, given the results of several medical tests (features). This kind of problems are called statistical classification problems. Usually, like in my own experiments, training data is used to calculate the relations between these outcome variables and their predictive features.

4 Feature Types

In this chapter, I will describe several different methods of feature extraction that I used during my experiments. First, I will describe the basic n-gram features, which have already been used for some decades in NLP-related research. Then I will give a general description of neural networks. I will finish the chapter by describing the more state-of-the-art technique that I used, which makes use of word embeddings features.

4.1 N-grams

An n-gram is a string of *n* units which is part of a larger string (Cavnar & Trenkle, 1994). These units can be both characters and entire words. In this case, a word is simply defined as a string of *x* characters, separated by a space on both sides. When using character n-grams for classification, spaces are mostly also included, since word borders are also important information. A unigram is the smallest possible kind of n-gram. One unigram consists of one character or one word. Machine learning based on only word unigrams is called a 'bag-of-words' representation, because it simply takes separate word frequencies as its features. When making use of bigger n-grams, one can also take phrases and other multi-word expressions into account. In this way, word order can also become a feature [1].

N-gram-based matching can be used for a lot of purposes in computational linguistics. For instance, it can be used for interpreting postal addresses and for text retrieval (Cavnar et al., 1994). When n-grams are counted that are common to two strings, you get a measure of their similarity. This measure is resistant to a lot of textual errors, including spelling errors. Usually, in machine learning, n-grams are used as features to predict certain attributes of elements in a dataset. In the case of hashtag prediction, all the tweets in the training set can be converted to n-grams. After that, for instance, n-gram frequency counts can be used to discover patterns in the data. Concretely, specific n-gram frequencies for tweets could be linked to specific hashtags. In this way, a hashtag prediction system can be developed. It is also possible for such a system to extract all n-grams within a certain range. For instance: one can let the algorithm extract all n-grams in the range from 1-grams to 5-grams. Often, when enlarging the n-gram range, the overall performance of the system will increase. However, there always is a limit to this increase in performance. At some point, adding higher n-gram levels will no further improve the performance. The performance will then flatten out or decrease. This limit largely depends on the kind of data and the quantity of the used data.

4.1.1 Feature Extraction

One of the vectorization algorithms provided by scikit-learn is the so-called *CountVectorizer*. This algorithm converts a set of one or more text documents to a matrix of token counts. With standard settings, it is just a bag-of-words-based learning algorithm. However, it can be manually configured to include n-gram extraction as well. Furthermore, it is able to produce a sparse representation of the matrix, reducing the amount of RAM that is required to store the matrix. This has the purpose of speeding up the calculation process. Since most tweets (or any relatively short document) will usually include just a fraction of all the words in the vocabulary, many of the features in the resulting vector will have a value 0 (usually more than 99% of all the features) [1]. With sparse representations, such matrices can be radically compressed. With standard settings, the number of features will be determined by the amount of unique words encountered in the training data. The algorithm enables for both character level and word level n-gram extraction. A lot of other parameters, like ignoring stop words and non-ASCII characters can also be predetermined in this algorithm. For my research, both character and word features have been explored, without any further special parameters like stop word skipping. Since machine learning algorithms usually expect numerical feature vectors, it is not possible to feed raw textual data into the algorithm directly. The symbols first need to be converted into a numerical feature vector. These vectors need to have a fixed size. This is done in scikit-learn by executing three main steps:

- Tokenizing strings. This means that separate strings (i.e. words) are recognized and labeled with a unique ID. White spaces and punctuation marks are often interpreted as token separators.
- Counting the occurrences of all the unique tokens in the training data.

- Normalizing and weighting the tokens: tokens that are more frequent throughout the data are given a lower weight than tokens which occur only rarely.

In this algorithm, each token is considered a feature, which is represented in a feature vector. The vector which contains all the features for a given document is considered a multivariate sample. Vectorization is thus the general process of turning textual data into numerical feature vectors, which can be handled by machine learning algorithms.

4.2 Word Representations

4.2.1 Artificial Neural Networks

Artificial neural networks are computational models which are inspired by the way a biological brain processes information. In this way, a system should be able to learn and recognize patterns in ways similar to a brain, using interconnected 'neurons' to make calculations. They are used for solving a wide variety of tasks that are almost impossible to solve (or at least computationally very intensive) with classical rule-based algorithms. They also offer a new approach to a wide range of classification problems.

ANNs are models with a parallel computation architecture. Because of this architecture, ANNs are very flexible and suitable for a wide variety of tasks, just as biological brains are. ANNs consist of a big network of interconnected simple nodes. There are several general tasks for which ANNs are used.

In pattern classification, the system tries to assign inputs to one of many classes. In function approximation, the system tries to find an underlying function which can explain patterns in data. When such a function is found, missing data can be calculated. This is applied in several scientific and engineering tasks. Other tasks that can be solved with ANNs are data prediction and optimization problems.

4.2.2 Neurons

A neuron is a biological cell that processes information (Jain et al., 1996). According to recent estimates, there are around 86 billion neurons in an average male human brain (Azevedo et al., 2009). Each neuron is connected to 10^3 to 10^4 other neurons. This adds up to a total of around 10^{15} inter-neural connections in the entire brain. In ANNs, researchers try to replicate this system in the form of nodes which are interconnected via layers. This will be explained more fully in the next section.

4.2.3 Feed-Forward Networks and Recurrent Networks

ANNs can be seen as a network of nodes (neurons) and connections between these nodes (directed edges with weights). There are two main kinds of ANN architectures, based on the different ways in which the nodes are connected: feed-forward networks and recurrent (feedback) networks.

In feed-forward networks, there are no loops. In recurrent networks though, loops occur. This enables feedback during learning. In most common feed-forward networks, nodes are divided into layers with undirected connections between them. They are static, which means they only produce one set of output values from a given input. They consume relatively few memory because just the neuron states of one layer at a time have to be stored in RAM. Recurrent networks, on the contrary, are dynamic. In such an architecture, the output of a neuron can theoretically reenter the neuron an infinite number of times as a new input, via its feedback path.

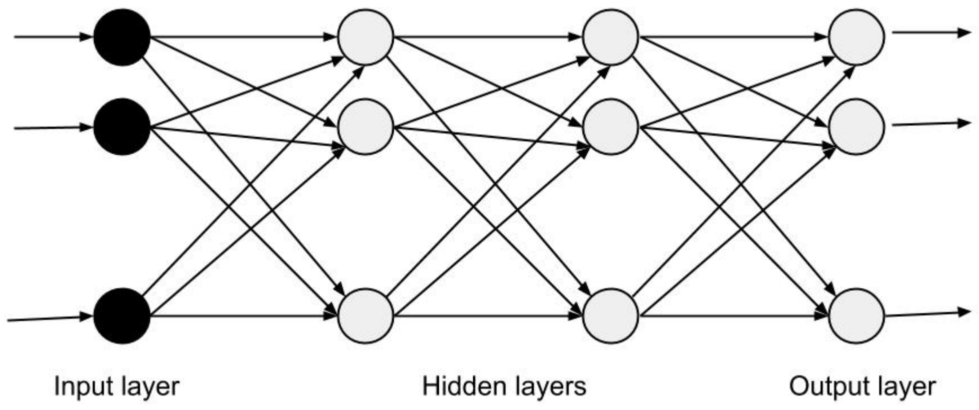

Figure 2: Schematic view of a feed-forward ANN architecture

Just as feed-forward networks, recurrent networks are trained by using training patterns. By reiterating the feedback loop over and over again, the system retrieves the optimum weights between all the nodes. Instead of following preprogrammed rules, it tries to discover the rules itself, by looking for patterns in big sets of data. In principle, adding more hidden layers to the system will yield better results. However, there usually is an optimum number of hidden layers which yields the best results for a specific task. It is not always easy to determine the optimal number of layers and number of nodes per layer. In practice, these parameters are often determined by trial and error (Jain et al., 1996). It is also important to consider that when the number of layers increases, the computation time increases proportionally.

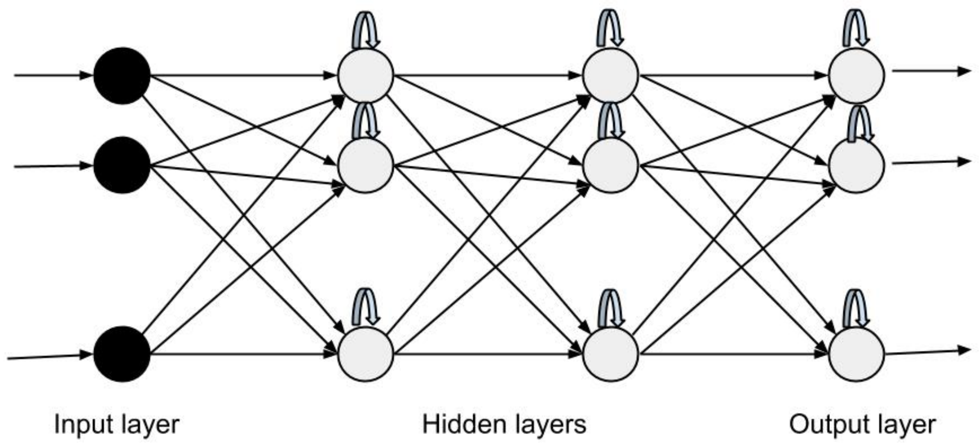

Figure 3: Schematic view of a recurrent ANN architecture. Each node can be reentered repeatedly.

In Figure 2, a scheme of a typical three-layer feed-forward perceptron can be seen. Usually, an L-layer feed-forward network has an input layer, $L-1$ hidden layers and an output layer. As can be seen, in such a system there are no feedback loops nor are there connections between the nodes within a single layer.

In Figure 3, a recurrent ANN can be seen, where nodes can be reentered. Theoretically, these nodes can be reentered an infinite number of times. It has to be noted that here are different possible recurrent ANN-architectures. For instance, in some cases the nodes within one hidden layer are linked to one another as well. Sometimes there's backpropagation from the output layer back to the last hidden layer. However, Figure 3 comes close to the model that is used by Mikolov et al. for their word embeddings vectorizer, the Recurrent Neural Net Language Model, described in more detail in Section 4.2.6.

Vectorized distributed word representations (DWR) can help machine learning algorithms to produce better results in several natural language processing (NLP)-tasks (Mikolov et al., 2013c). In these representations, similar words are grouped together. DWR has been used in a lot of NLP-problems. For instance, Collobert et al. (2008) have used it for multi-task learning. The aim of their model is that for a given sentence, it would output multiple language predictions, namely part-of-speech tags, chunks, named entity tags, semantic roles, semantically similar words and grammatical and semantical probability scores. This is unique, because mostly, researchers focus on only one or two of these topics, instead of trying to cover them all in one model. All the outputs are based on the language model, which is generated by unsupervised learning on unlabeled text (the entire English Wikipedia corpus). The used training technique is called weight-sharing, which is an example of multi-task learning. Here the term 'weight' refers to the weights that are given to inter-neural connections in an ANN. Weight sharing is usually applied in Convolutional Neural Networks (CNNs), which are often used for solving vision problems. Images often contain repetitive features. When using weight sharing, these features can easily be detected regardless of their position in the visual field. In this way, the number of free parameters that have to be learned is reduced significantly. The features are automatically trained, using backpropagation in hidden layers, as explained earlier in this section. Using this approach, the model is very flexible and can be used for many different NLP tasks.

4.2.4 Word Embeddings

Until recently, it was common for many NLP tasks, like sentence prediction, to only make use of n-gram features. However, also manually selected features and dictionary-based features were commonly used. Usually, this feature selection is based on prior linguistic knowledge, combined with trial and error during the supervised learning: the features that turn out to be the best predictors for a specific tasks, are selected (Collobert et al., 2008). For relatively simple, specific tasks, these methods may be sufficient. However, for more complex tasks, like hashtag recommendation, some more sophisticated learning techniques could be useful.

One of the techniques which can be used is incorporating distributed word representations during the training phase. Concretely, this means that the algorithm takes the semantic relatedness between words into account during training. The aim is that this will lead to better pattern recognition, since now it does not only look for character or word (co-)occurrence patterns, but also for semantical patterns. For instance, the system will be trained to know that the words 'man' and 'boy' are semantically more related than the words 'man' and 'cat'. It does this by means of statistical inference: the words 'man' and 'boy' will probably occur more often in similar contexts than the words 'man' and 'cat'. Based on that observation, the system will infer that they are semantically more related. So if the system encounters for instance the sentence 'The man is ill', and somewhere else it encounters the sentences 'The boy is ill' and 'The cat is ill', it will 'know' that the first two sentences are closer together semantically. This knowledge can be useful during the classification of sentences: for instance during hashtag prediction for tweets, as in my own experiment. Moreover, since in recent years much more processing power has become available for affordable prices, using more complex features on larger training sets has become more feasible as well (Mikolov et al., 2013a).

4.2.5 Latent Semantic Analysis

Many different architectures for DWR have been used. One famous approach is the Latent Semantic Analysis (LSA). This is a technique which analyzes relationships between several document. Note that a single document in the context of my research can be a tweet as well. LSA analyzes this relationships by looking at the terms that are used in all the documents and abstracting over these terms by generating concept clusters. The model assumes that words which are similar semantically, will occur regularly in the same contexts within a text. A matrix is generated from a corpus. This matrix contains word counts including their context data. After that, the words are compared by taking the cosine of the angle between the two vectors of any two rows. If the value is close to 1, the words are very similar. If the value is close to 0, the words are very dissimilar.

Another well-known approach is Latent Dirichlet Allocation (LDA), which was already described in Section 2.2.1.

4.2.6 Feed-Forward and Recurrent Neural Network Language Models

There are two main kinds of neural networks: the Feed-Forward Neural Net Language Model and the Recurrent Neural Net Language Model. These models are based on the architectures described in Section 4.2.3.

The Feed-forward Neural Net Language Model (NNLM) is a model in which a feed-forward network with a linear projection layer and a non-linear hidden layer are used to learn word representations (Mikolov et al., 2013a). Additionally, it generates a statistical language model. This is called a neural probabilistic language model. The parts it consists of are input-, projection-, hidden- and output layers. At the input layer, N previous words are encoded. 1-of-V coding is used, where V represents the size of the vocabulary. Then the input layer is projected to projection layer P, with dimensionality $N \times D$, using a shared projection matrix. For $N = 10$, the size of the projection layer P may be 500 to 2000 nodes. The size of the hidden layer H is usually 500 to 1000 nodes. The hidden layer is used to calculate probability scores for all the words in the vocabulary. This results in an output layer with dimensionality V, where V is the size of the vocabulary.

The Recurrent Neural Net Language Model (RNNLM) has been introduced in an attempt to overcome some of the practical problems of NNLM (Mikolov et al., 2013a). Because RNN models do not have a projection layer, they can theoretically calculate more complex patterns than NNLM in the same amount of time. In RNN models, analog to biological brains, the hidden layer is connected to itself. This accounts for some kind of 'short term memory', because the hidden layer learns by combining past hidden layer states with new input. This is achieved by using 'time-delayed connections', which means that past information can be represented in the hidden layer state by combining new input with the previous hidden layer state. More formally, the input vector $x(t)$ will be a concatenation of vector w (representing the current word) and the output from neurons in context layer s at time $t-1$. A visual representation of this model can be seen in Figure 4.

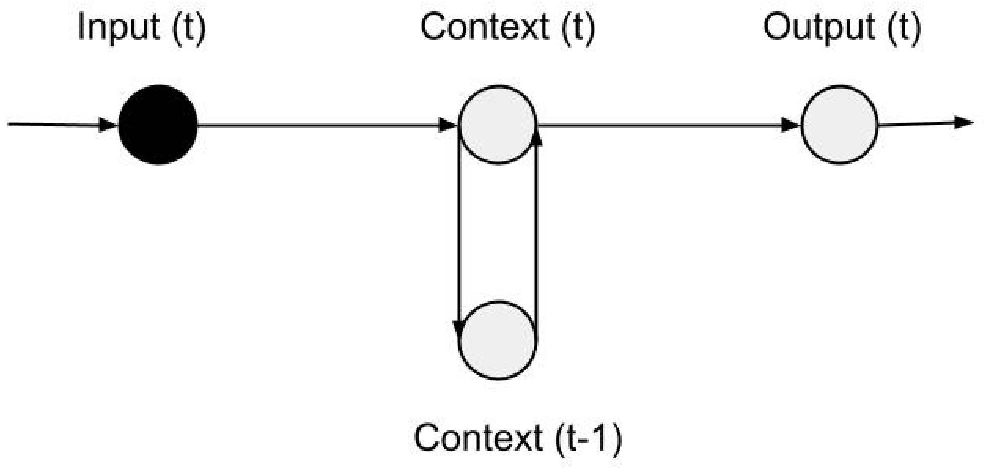

Figure 4: Schematic view of the Recurrent Neural Network-based Language Model (based on Mikolov et al., 2010)

4.2.7 Continuous Bag of Words and Continuous Skip-Gram Model

The novel models introduced by Mikolov et al. (2013a) are CBOW and CSG. The Continuous Bag-of-Words Model (CBOW) is similar to the NNLM model. It only takes word occurrences into account while determining word similarities between words, not the word order of the context words. The best performance was obtained by using the four words both before and after the current word for predicting the current word. The Continuous Skip-gram Model (CSG) is similar to CBOW, but instead of predicting the current word based on the context, it predicts the context (within a certain range) based on the current word (see Figure 1). It was found that the bigger the range was, the better the system performed. However, increasing the range also means increasing the computation time. This model does not involve dense matrix multiplications. Therefore the training can be done a lot quicker than in other neural net based learning algorithms. It can theoretically train on more than 100B words per day. This results in a great improvement of the word similarity representations, especially for rare words and phrases. They also found that when they subsampled high-frequency words, this benefited the general speed of the system. The mapping of low-frequency words also improved. The most important factors affecting the performance of the system were choice of the model, vector size, amount of subsampling, and the size of the vocabulary.

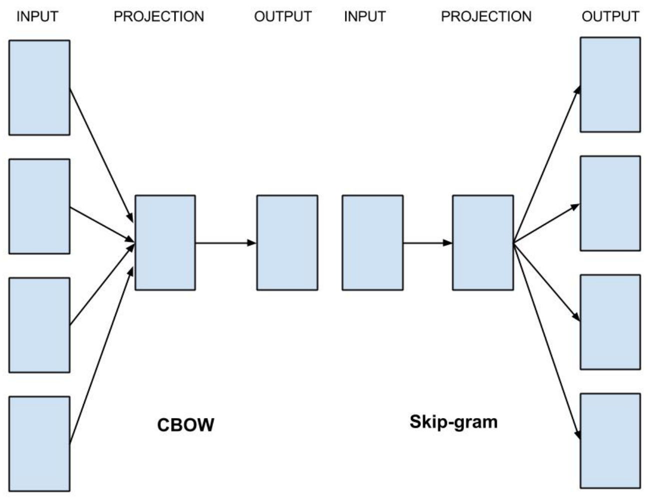

Figure 5: In the CBOW model, context is used to predict a single word. In the Skip-gram model, it's the other way around: words are then used to predict its context.

During training with the Skip-gram model (CSG), the aim is to find the words that are most useful for predicting their context words. Mathematically, the objective of CSG is to maximize the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\left[\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)\right]$$

where c is the size of the training context, w_t represents the center word, and T represents the amount of training words. A larger c means more training samples, which might lead to a better performance. However, the amount of training time will increase proportionally with it.

In their experiment, Mikolov et al. (2013a) used a Google News corpus of about 6B tokens for training the word vectors. They restricted the vocabulary size to the 1M most frequent words.

It has to be noted that all the models described in this chapter use linear predictor functions.

4.2.8 Evaluation of the Models

Mikolov et al. (2013a) compared the different models by letting them perform a word similarity task. Five types of semantic questions and nine types of syntactical questions were asked. The questions were created in two steps. First, a list of word pairs was created. Then a list of questions was generated by connecting two word pairs. An example of a semantic question could be what the capital city of a given country is. Using 640-dimensional word vectors, the Skip-gram model turned out to perform significantly better at answering semantic questions than the other models. It had a 55% accuracy, as compared to 24% accuracy for the CBOW algorithm, 23% for NNLM and only 9% for RNNLM. However, CBOW performed a bit better at syntactic questions: 64%, as compared to 59% for Skip-gram, 53% for NNLM and 36% for RNNLM.

4.2.9 word2vec

The word2vec (word to vector, since it transforms words into vector representations) algorithm is based on distributed word representations. It is an open source tool developed by Google for research purposes. This tool was used during the experiments of Mikolov et al. It implements the CBOW and Skip-gram architectures, developed by Mikolov et al., which I described before. Just like many other training algorithms, it takes a textual corpus as its input and generates numerical feature vectors from it. It constructs a vocabulary based on all the unique words it encounters in the training data. Based on all the collected data, it calculates words similarity matrices. These matrices can then be used to determine the similarity (expressed in the output as Cosine distances) between several words.

The similarity calculations are based on word co-occurrence frequencies. In addition to word similarity measurements, inherently a lot of linguistic regularities are also captured by this algorithm. For instance, vector(king) - vector(man) + vector(woman) yields a vector which is close to vector(queen). By default, it represents each word with a vector consisting of 200 values. In addition to single word representations, similarities between phrases can also be captured with the word2phrase algorithm. For instance, it could be useful to interpret proper nouns such as San Fransisco as a single unit, clustering with other single- or multi-word toponyms. However, this word2phrase tool was not used during my experiments.

4.3 Summary

In this chapter I described n-gram-based classification and classification based on word embeddings features, to provide a clearer image of the models that underly the algorithms that I used in my experiments. I also gave a general description of the working of ANNs, which are also used during training by the word vectorization algorithm (word2vec).

5 Experiments

In this chapter, I will describe my experiments, going into more detail about their specific technical aspects. I will start with explaining why I chose to use a Twitter-based corpus for my experiments.

After that, I will go in more detail about the pre- and post-processing I did for the experiments and the evaluation metrics that were used to evaluate the performance of the different systems.

Finally, I will present the results and discuss them.

5.1 Settings

5.1.1 Corpus

Twitter data was chosen as a corpus for my experiments, because it is a widely used social network with a lot of corpus data freely available on the web. Its users make great use of meta-tagging in the form of hashtags, which makes it an ideal source for research in the field of tag suggestion.

The used corpus consists of 1M English tweets [3]. This corpus is made freely available by Illocution Inc., a consortium of industry professionals working in fields which require big sets of textual data. The 1M tweet data set is a subsample of a bigger sample called the Illocution Inc. Twitter Stratified Random Sample, which consists of more than 12M tweets. However, this bigger corpus could not be obtained for free. Therefore, the smaller 1M corpus was chosen for use in my experiments. Illocution collects tweets non-stop from the so-called Twitter stream. These 12M tweets, of which the used 1M corpus is thus a subsample, were collected between January 1 and December 31 of the year 2012.

5.1.2 Pre- and Post-Processing

In this section I will describe the steps that are taken during my experiments. The first step is obtaining the data and extracting the most frequent hashtags from it. Then comes the feature extraction. Here the CountVectorizer does its work: it will generate 1- to *x*-grams, where *x* depends on the parameter entered in the command line, prior to execution. As mentioned before, these will be character or word n-grams. Step 3 consists of training. Now the vectorizer learns the vocabulary of the corpus (basically generates a dictionary) and returns the count vectors. Each unique hashtag represents a unique class. When the training is finished, the classification (step 4) of the newly presented data (development set or test set) starts. It uses a one-vs-the-rest (OvR) classifier to classify all the tweets in the new data. This essentially means that the systems attends one hashtag to every tweet in this dataset. It does this by applying logistic regression. When this is done for all tweets, it counts the relative amount of correct classifications. Finally, during step 5, precision, recall, mean average precision (MAP) and F1-scores are calculated. I will now go into more detail on all of the aforementioned steps.

After I downloaded the 1M tweet dataset, I first filtered out all the tweets that contained hashtags. 17.6% of all tweets turned out to contain one or more hashtags, which is slightly more than the 14% that Hong et al. (2011) found in their study of English language tweets. Then I automatically extracted the 500 most frequently used hashtags from the dataset. I chose the number of 500 because I thought that a system that is able to predict the 500 most frequently used hashtags would be sufficient for most cases. The result was a list of the 500 most frequent hashtags, ordered by frequency count in the corpus. Table 1 shows the ten most frequently occurring hashtags in the corpus, just to give an impression of this data.

After obtaining the 500 most frequent hashtags, the tweets containing those hashtags were written to a file, together with their corresponding hashtag(s). This resulted in a document containing 26,841 tweets plus their corresponding hashtags,

Table 1: Overview of the ten most frequently occurring hashtags in the used Twitter corpus

Rank	Hashtag
1	#teamfollowback
2	#np
3	#oomf
4	#nowplaying
5	#ff
6	#rt
7	#jobs
8	#porn
9	#s
10	#1

which is 15.3% of all tweets containing hashtags. In other words, the top 500 most frequently used hashtags represent 15.3% of all tweets containing hashtags.

The file containing these 27K tweets was split into three separate parts: the first part being the training set, the second being the development set, and the third being the test set. In all three cases, the hashtags in the tweets themselves were removed from the data, to ensure a fair training and classification process. Table 2 shows the division of the data over the different sets during the experiments. As can be seen, the training set is the biggest, to offer the opportunity to the different systems to do training with a decent number of training samples. The development and test set are relatively small, but still considered big enough to be a representative sample for evaluation.

Table 2: Division of the data over the different sets

Set	Number of tweets
Training set	15,000
Development set	5,000
Test set	6,841

Firstly, a logistic regression model using n-gram features was executed on the dataset to be used as a baseline. This model was used for training on the training set, extracting both character and word unigrams, bigrams, trigrams, till 9-grams from the training set.

Upper case and lower case characters were set to be treated as distinct characters, which might be of particular relevance for character n-grams. All other parameters were set to their default.

When all features are collected, the one-versus-rest (OvR)-classifier tries to predict the correct labels for all the tweets in the develop set. It does this by applying multinomial logistic regression (as described in Section 3.2).

After the classification process is finished, precision, recall, F1-scores and mean average precision (MAP) are calculated, comparing the gold standard hashtags to the predicted hashtags. These evaluation metrics will be described in more detail in Section 5.1.3.

The most frequently found confusions in the data were outputted as well, ordered by frequency count. Apart from that, a confusion matrix was generated.

The word embeddings features were trained by using the word2vec algorithm. This algorithm makes use of the CBOW and Skip-gram architectures described in Section 4.2.7. It was trained on the originally obtained 1M tweet corpus. Several vector files were generated with different parameter settings. These were all tested on the development set. In Table 3, the tested parameter settings are displayed.

Table 3: Tested parameter settings for the training algorithm

Setup	**Architecture**	**Vector size**	**Maximum skip length**	**Softmax**
1	CSG	200	5	yes
2	CBOW	500	5	no
3	CSG	500	5	yes
4	CSG	500	10	no
5	CSG	1000	10	no
6	CSG	1000	5	yes

The best performing vector file (with F1 = 0.11 and MAP = 0.11) was used for all further experiments. The parameter settings (Setup 5 in Table 3) that yielded this best performing vector file were as follows: It used a relatively high dimensionality of 1000 dimensions per word. When using this setup, the training time was 188 minutes.

The number of used dimensions (D) is 1000. This means there are 1000 nodes in any of the layers of the network (input, hidden and output layer) $= 3 \times 1000 = 3000$ nodes in total. The complexity (Q) of the ANN per training example can be calculated as follows:

$$Q = H^2 + H \times V$$

where H denotes the number of nodes in the hidden layer and V denotes the size of the vocabulary (Mikolov et al., 2013a).

The developers claim that higher dimensionality will in general lead to better results. However, the downside of higher dimensionality is that learning will take longer. This is the case for both the training of the word embeddings themselves, as well as the later classification which makes use of these features. Nonetheless, the advantage of using a relatively small corpus, as I did, was that it was not such a problem to use some heavier parameter settings. The chosen architecture was the Skip-gram architecture, because this performed best on my data. The maximum skip length was set to 10, which the developers recommended for the Skip-gram architecture. The training algorithm was set to make use of negative sampling (better for frequent words) instead of hierarchical softmax. Sub-sampling of frequent words was turned off. This can be useful for bigger datasets (especially for computational efficiency reasons), but it turned out to be of no added value for my relatively small dataset. The outputted vectors were set to be real-valued.
After all the baseline experiments were finished, the same experiment was conducted using only word embeddings features, with the best performing vector file and without the n-gram baseline. After that, the main experiment, combining n-grams with word embeddings, was conducted. In this experiment, the n-gram matrices were concatenated with the word embeddings matrices.

Finally, the two best performing conditions for both baseline and word embeddings conditions were re-executed on the test set. The best performing conditions however turned out to be on the same n-gram levels for both the baseline as the baseline combined with word embeddings: the character n-grams baseline performed best for trigrams. When adding word embeddings, trigrams still performed best. For the word n-gram baseline, unigrams performed best. Here again, when adding word embeddings, the unigrams remained the best performing condition (see Figure 10).

5.1.3 Evaluation Metrics

The evaluation metrics that were used to evaluate the performance of the baseline and the additional word embeddings classifier are mean average precision (MAP) and F1-score. These will be explained in more detail in this section.

Precision and recall are values which can be used to evaluate the performance of any classification system. In the context of my experiment, precision is the relative number of tweets that where correctly labeled as x (where x is a hashtag) by the system:

$$Precision = \frac{\text{Tweets that are correctly labeled } x}{\text{Total number of tweets that are labeled } x}$$

Recall is the number of tweets that is correctly labeled as x, relative to the number of tweets that should have been labeled as x:

$$Recall = \frac{\text{Tweets that are correctly labeled } x}{\text{Tweets that should have been labeled } x}$$

The F1-score (or F-score) is another measure that can be used to compute the accuracy of a test. For obtaining the F1-score, one has to calculate the harmonic mean of the precision and the recall.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

The average precision is obtained by plotting precision *p(r)* as a function of recall *r*. The average precision is then calculated by computing the average value of *p(r)* over the interval $r = 0$ to $r = 1$.

$$AveP = \int_0^1 p(r)dr$$

The mean average precision (MAP) is the mean of all the average precision scores for all queries. In the case of this experiment, each tweet represents one query.

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

where Q is the number of queries.

5.2 Results

In this section, I will present the results of my experiments. I will start with a table presenting precision and recall scores. Then I will present the results of the two baseline conditions: character n-grams and word n-grams. After that, I will mention the results of the experiment using only word embeddings without a baseline. Then I will show the results of the experiments in which I combined the n-gram baseline with word embeddings. These are all results of experiments executed on the development set.

I will then show the results of rerunning the best performing conditions on the test set, again comparing baseline against added word embeddings. I will finish the chapter with a confusion matrix, showing the most frequently confused hashtags of the classifier.

Table 4: Precision, recall, F1 and MAP scores for the best performing conditions for the baseline and with added word embeddings.

	Precision	Recall	F1-score	MAP
Character 3-gram baseline	0.55	0.13	0.17	0.18
Character 3-gram + word embeddings	0.47	0.14	**0.18**	0.19
Word 1-gram baseline	0.52	0.05	0.07	0.09
Word 1-gram + word embeddings	0.41	0.11	**0.13**	0.15

Table 4 shows precision and recall scores for both the best performing baseline and best performing conditions when adding word embeddings. The highest F1-scores are marked in boldface. It is noteworthy that for both character and word n-grams, precision goes down when adding word embeddings, while recall goes up. For character n-grams + word embeddings the following can be noted: even though precision drops with 8 percentage points, and recall increases with only 1 percentage point compared to the baseline, the overall F1-score increases with 1 percentage point. This can be explained by the fact that F1-scores represent the harmonic mean of precision and recall. Therefore, improvement of the lower value (recall in this case) will usually have a more radical influence on the F1-

score than the decrease of the higher value (precision in this case).

In Figures 6 and 7, the F1-scores and MAP-scores for the character n-gram and word n-gram baselines are shown.

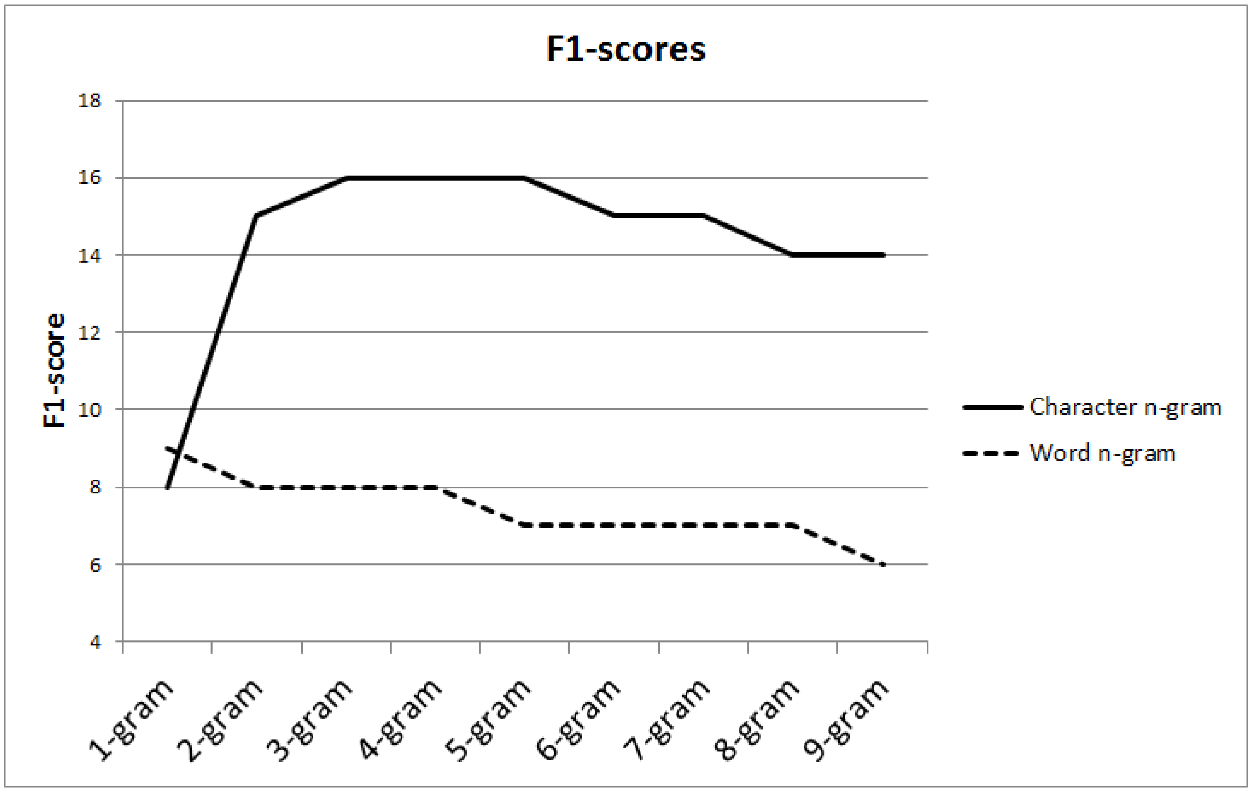

Figure 6: F1-scores for the character n-gram and the word n-gram baselines

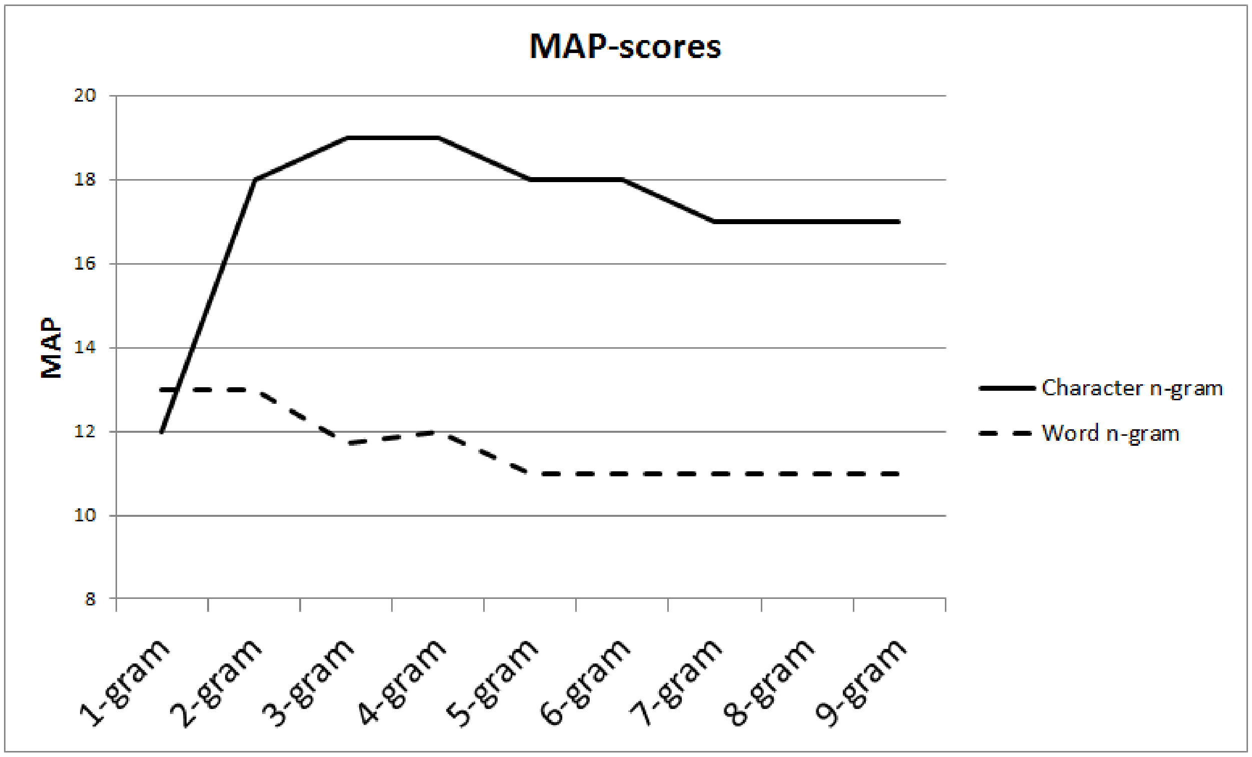

Figure 7: MAP-scores for the character n-gram and the word n-gram baselines

The experiment was also executed without the n-gram baseline, using only word embeddings as features. The results of this experiment are presented in Table 5.

Table 5: Results of word embeddings experiment without baseline

F1	**MAP**
0.11	0.11

In Figures 8 and 9, the F1-scores and MAP-scores for character n-grams and added word embeddings are presented.

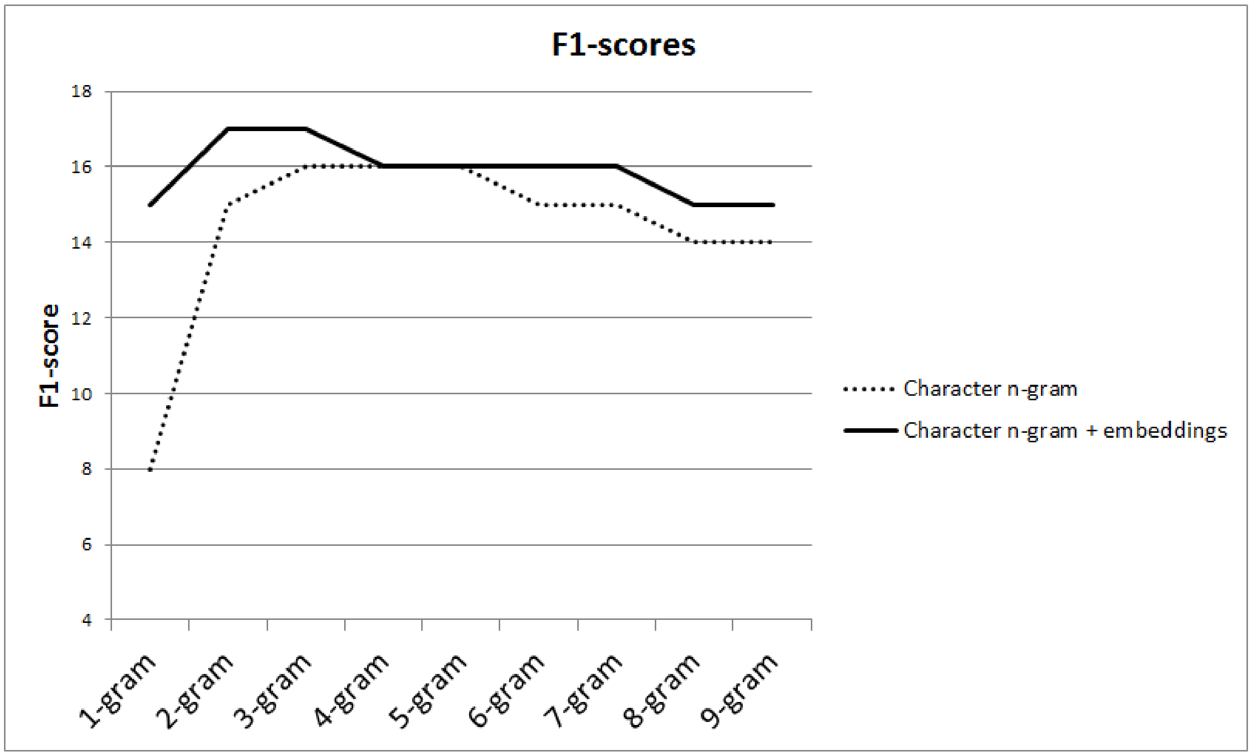

Figure 8: F1-scores for the character n-grams baseline and the additional word embeddings features

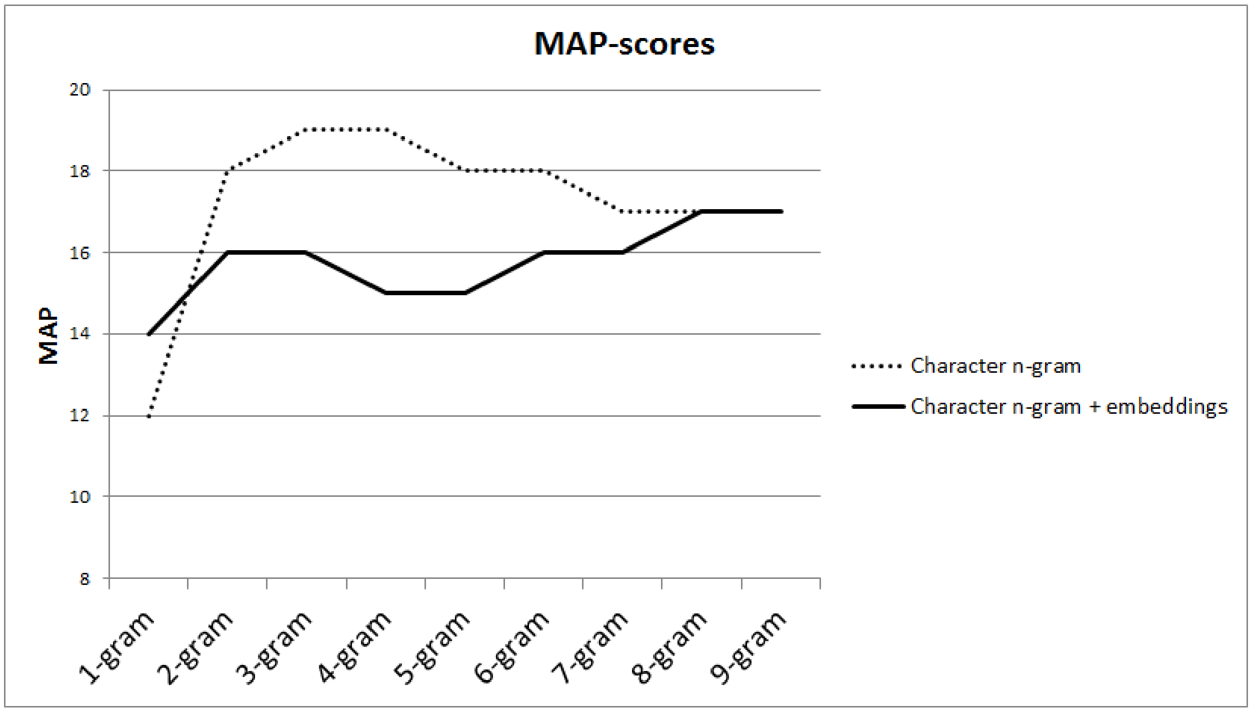

Figure 9: MAP-scores for the character n-grams baseline and the additional word embeddings features

In Figures 10 and 11, the F1-scores and MAP-scores for word n-grams and added word embeddings are presented.

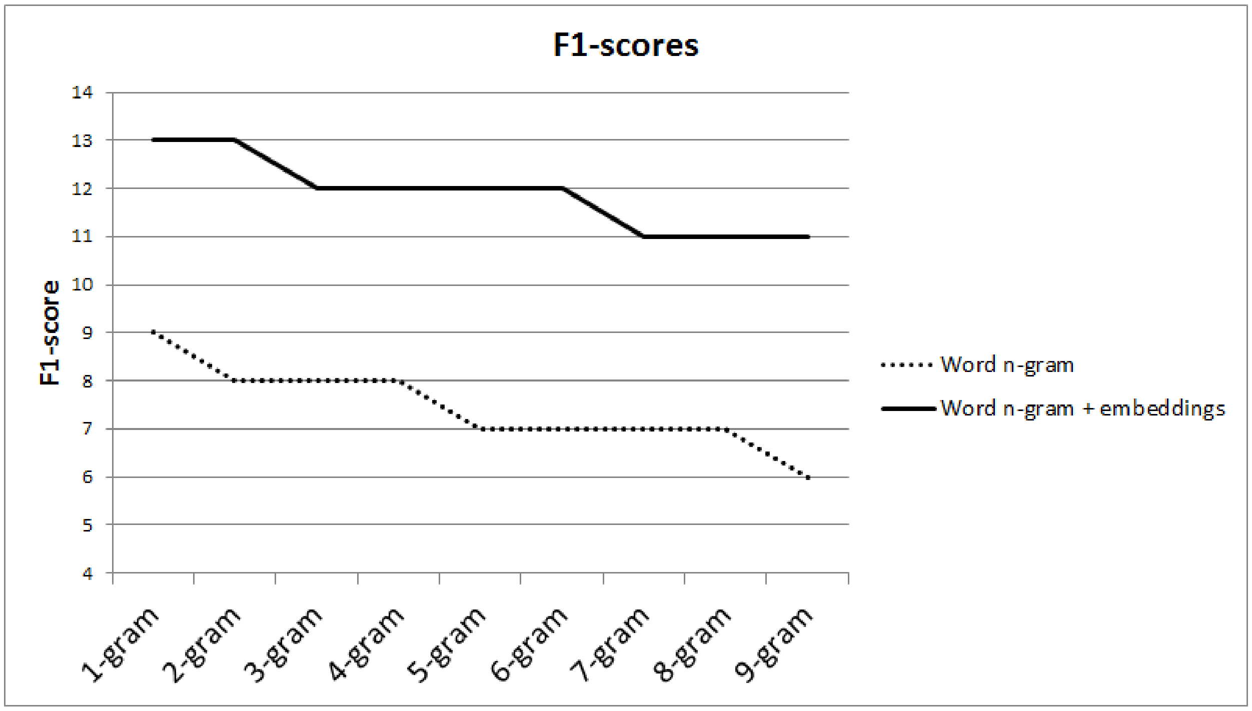

Figure 10: F1-scores for both the word n-grams baseline and the additional word embeddings features

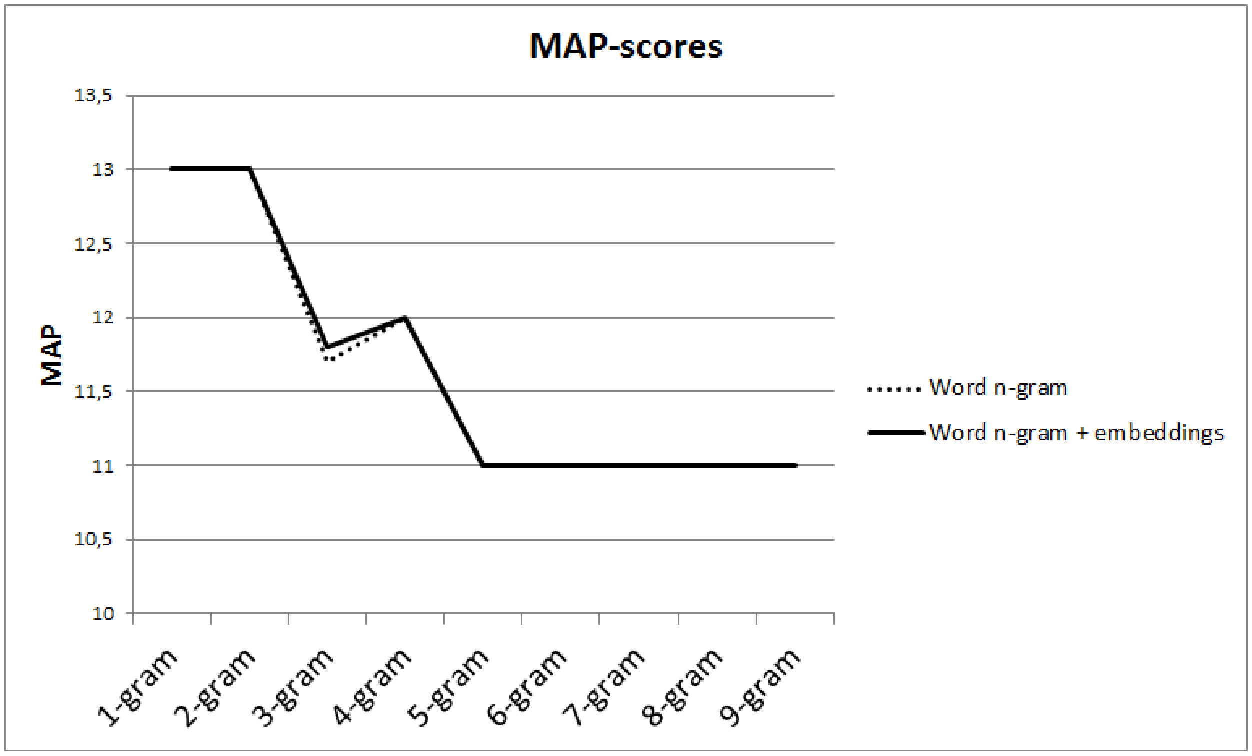

Figure 11: MAP-scores for both the word n-grams baseline and the additional word embeddings features

Finally, I ran an extra experiment on the two conditions which performed best on the development set. These were character 3-grams and word 1-grams for both the baseline and baseline plus word embeddings. This yielded the results as presented in Figure 12.

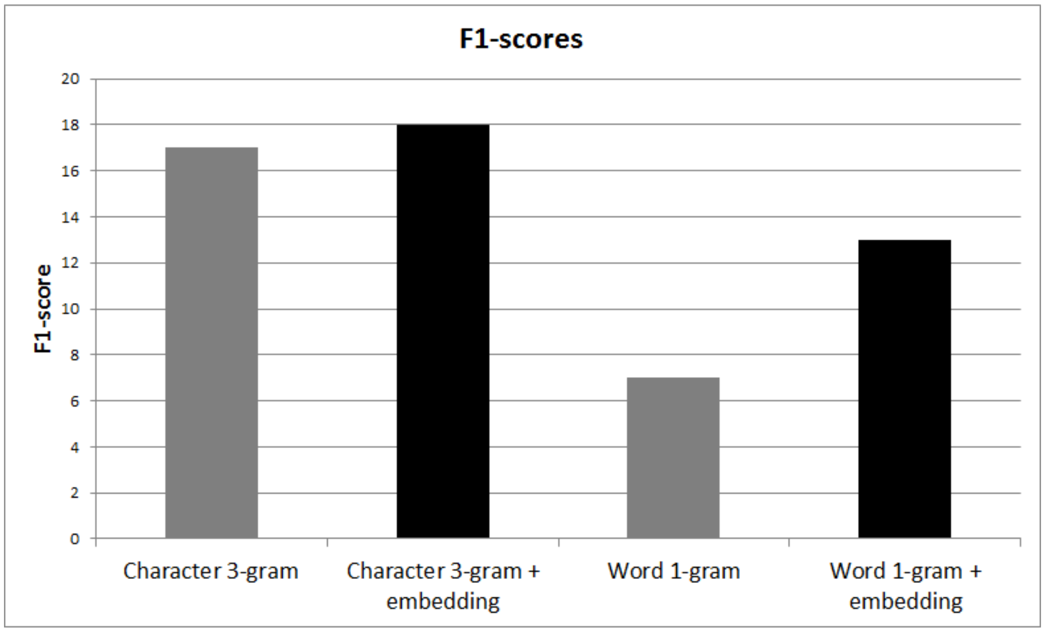

Figure 12: F1-scores for optimum conditions: baselines and baselines plus word embeddings

To make clear which hashtags are often confused, I generated a confusion matrix for the most frequent confusions. It is displayed in Table 6. The most frequent confusions are displayed in boldface. The vertical axis contains the actual gold standard tags, the horizontal axis contains the wrongly predicted tags that are put in its place. It has to be noted that this matrix purely focuses on false classifications: for clarity reasons, correct classifications are not displayed in the matrix.

Table 6: Confusion matrix showing the most frequent confusions of the classifier

	#game-insight	#android	#ff	#500-aday	#follow-back	#ipad-games	#must-follow	#album	#now-playing	#np
#game-insight	_	**196**	0	0	0	0	0	0	0	0
#android	0	_	0	0	0	0	0	0	0	0
#ff	0	0	_	**16**	**8**	0	0	0	0	3
#500-aday	0	0	0	_	0	0	0	0	1	0
#follow-back	0	0	0	2	_	0	0	0	0	0
#ipad-games	2	0	0	0	0	_	0	0	0	0
#must-follow	0	0	0	0	5	0	_	0	0	0
#album	0	0	0	0	0	0	0	_	0	4
#now-playing	0	0	0	0	0	0	0	0	_	**31**
#np	0	0	0	0	0	0	0	0	**16**	_

It has to be noted that some of the confusions as displayed in Table 6 are no actual confusions. For instance, #nowplaying and #np (abbreviation) refer to the exact same concept. There are more of such instances, which will be explained in more detail in Section 5.3.7.

5.3 Discussion

5.3.1 Character- vs. Word N-grams

First of all, it has to be noted that character n-grams perform considerably better in general. The only exception are the unigrams: word unigrams perform a little bit better than character unigrams, as can be seen in Figure 6. The fact that character n-grams generally perform better might be explained by the nature and size of the used data set: tweets are only small bits of text (max. 140 characters) and there are 'only' 15,000 of them in the used training set. When classifying such data, character n-grams might provide an advantage in the sense that they need less data for meaningful pattern recognition. This can easily be explained as follows: when an algorithm looks for trigrams in a given data set, it will only look for a maximum of about $50^3 = 125,000$ combinations of characters (in case of English, the average Twitter user will use about 50 different characters: the letters of the alphabet, numerals, interpunction characters and meta-characters like '@' and '#'). 125,000 might seem like a lot, but considering that there are 15,000 tweets in the training set, with an average length of about 50 characters, there are about 750,000 trigrams present in the data. This number is a factor six bigger than the number of possible trigram combinations, which means that most trigrams (especially the more frequent ones) will occur many times in the data. For word trigrams, it's different. An average Twitter user will have a vocabulary of around 20,000 words (certainly when hashtags, misspellings and slang are included). This means there are $20,000^3$ possible word 3-grams, which is a very large number. For that reason, word n-grams generally perform better on bigger data sets. Another reason why the character n-grams perform better might be the fact that an average 50-character tweet consists of 50 character trigrams, and only around 10 word trigrams. This gives character-based classification an extra advantage for short documents.

5.3.2 Baseline Results

As can be seen in Figure 6, the best performing conditions are unigrams for words and trigrams for characters. The latter is also the best performing condition in general. Character unigram performance is relatively poor. From there it goes up until it reaches its peak with trigrams. From there, performance steadily drops again. Word n-gram performance starts relatively good for unigrams and steadily drops from there. As Figure 7 shows, the trends of the MAP-scores are pretty similar to those of the F1-scores, although the values of the MAP-scores are several percents higher for both character and word n-grams. This can be explained by the overall lower recall under all experimental conditions, which has a negative influence on the F1-scores (see Table 4).

5.3.3 Word Embeddings Results

Table 5 shows that using only word embeddings without the baseline still gives pretty good results. Both the MAP and F1-score are 0.11 for this condition. This means that the standalone word embeddings features lead to a better performance than the word n-gram baseline, but worse than the character n-gram baseline.

5.3.4 Character N-grams + Embeddings

As can be seen in Figure 8, when word embeddings are included, the system always performs better than the character baseline. Some seem to perform equally, but in fact, the performance when adding word embeddings was still at least some fractions of a percent better than the baseline performance. It can be seen that character unigrams benefit most from the added word embeddings (an improvement of 7 percentage points). For all the other n-grams, the improvement is about 1 percentage point. It has to be noted that the MAP of the baseline is generally higher though (see Figure 9). When adding word embeddings, the F1-score goes up, while the MAP drops. In other words: recall increases while precision decreases. There is no clear explanation for this fact. Possibly, the recall increases because more candidate hashtags get above the critical probability threshold, which leads to more tag predictions. For the character trigram baseline, the number of predicted tags is 2740. When adding word embeddings, this number raises to 3225. This higher number of predicted tags comes at a price: although more of the gold standard tags are now found, there are also more incorrect tag predictions, hence the lower precision. More extensive examination of the output data would be necessary to be able to substantiate this explanation.

As noted earlier, the lower precision is compensated by a higher recall. This explains why the F1-scores are still superior when adding word embeddings features (see page 44).

5.3.5 Word N-grams + Embeddings

Figure 10 shows that word n-grams always perform better when word embeddings are added. The increase of the F1-score seems to be relatively constant for all conditions (about 4 percentage points). Figure 11 shows that the MAP-scores for both conditions are practically identical. Given the higher F1-scores, one can infer that recall raises when adding word embeddings features.

5.3.6 Extra Experiment on the Test Set

The experiment was re-executed on the test set with the optimum conditions. As mentioned before, these were unigrams for words and trigrams for characters. Figure 12 shows that for characters, when adding word embeddings, the performance increases about 1 percentage point (0.9 percentage points to be exact). This is an improvement of approximately 5.3%. For words, the performance increases with 6 percentage points, an improvement of approximately 85.7%. These results are consistent with the results for the development set, which showed similar improvements for these conditions. An important thing to note is that word n-grams tend to benefit a lot more from adding word embeddings as features: 6 percentage points versus just 1 percentage point for character n-grams. There is no certain explanation to be given for this fact. Probably it can be explained by the fact that the word n-gram baseline is weaker than the character n-gram baseline: adding useful features will in general give bigger improvement over a weak baseline than over a stronger baseline.

5.3.7 Confusion Matrix

The confusion matrix in Table 6 shows the tags that are most frequently confused by the classifier. As can be seen, the confusion #gameinsight – #android is the most frequent confusion by far (196 times). Other quite frequent confusions are #nowplaying – #np (31 times) and vise versa (16 times) and #ff – #500aday (16 times).

5.3.8 More Tolerant Evaluation

After qualitative analysis of the data I concluded that a lot of the confusions as shown in Table 6 are actually no real confusions. Table 7 shows all tag groups that essentially represent the same tag. There are five big tag groups of this kind present in the data. As can be seen, the alternative tag is normally just an abbreviation of the original tag (ie. #nowplaying, #np), a plural form (ie. #job, #jobs) or a variation of the word (ie. #porn, #porno).

Table 7: Tag groups in the training data

1	#nowplaying, #np
2	#job, #jobs
3	#oomf, #oomfs
4	#porn, #porno
5	#quote, #quotes

Table 8 shows hashtags that are not to be considered identical in the same sense as in Table 7, but are still highly related semantically. This clustering was done by reasonable judgment.
Group 1 consists of tags related to gaming. Game Insight is a well-known developer of games for mobile platforms. Android and iOS (the OS of Apple's iPad)

Table 8: Highly related tags

1	#gameinsight, #ipad, #android, #ipadgames, #androidgames
2	#ff, #500aday, #1000aday, #followback, #ifollowback, #teamfollowback, #mustfollow
3	#giveaway, #win
4	#np, #nowplaying, #album, #popartist, #1, #soundcloud, #music
5	#jobs, #hiring, #sales

are the most popular mobile operating systems present, so these tags are arguably very related.
For several years, there has been a meme on Twitter in which people recommend users to follow to each other. This phenomenon is often indicated with the hashtag #followfriday or simply #ff, probably because of the alliteration that sounds well. All hashtags in group 2 refer to this phenomenon. It is also usual on Twitter to share the music you are currently listening to with your followers. The hashtags in group 4 are related to this habit.

In best performing circumstances on the test set (character 3-grams plus word embeddings), according to automated evaluation, there were 814 correct tag predictions (out of 6,841 tweets) by the classifier. These are only literal matches between gold standard tags and predicted tags. (Some of the tweets in the training set got multiple tags. In these cases, each tweet-tag combination would be evaluated separately.)

However, my analysis of the data showed that a lot of the predicted tags that were marked as wrong (because they were no literal matches with the gold standard tags), were actually correct tags, since they belonged to the same tag group (see Table 7) as one of the gold standard tags. Further analysis showed that many of the predicted tags marked as wrong were tags that were highly related semantically to their gold standard tag (see Table 8). In many practical cases, such a tag will probably be equally suitable for a given tweet as the gold standard tag. When taking these tag groups into account during evaluation, for character trigrams the actual precision would be 0.55. The recall would raise to 0.22, which yields an F1-score of 0.31. For word unigrams, precision would raise to 0.33 and recall to

0.25, which yields an F1-score of 0.28. These results are presented in Table 9. For character trigrams, the F1-score raises 13 percentage points as compared to strict evaluation. For word unigrams, the F1-score raises 15 percentage points.

Table 9: Precision, recall and F1-scores for the best performing conditions for the baseline and with added word embeddings, using tolerant evaluation.

	Precision	Recall	F1-score
Character 3-gram + word embeddings	0.55	0.22	0.31
Word 1-gram + word embeddings	0.33	0.25	0.28

5.3.9 Unlabeled Tweets

During classification, a lot of tweets were left unlabeled: 5,127 of the 6,841 tweets (75%) in the test set remained unlabeled. Classification only occurs above a certain probability threshold. When none of the candidate hashtags gets a probability score above this threshold, the tweet will get no hashtag at all. This explains the relatively poor recall (0.15) for the current classifier. In Section 6.3.2 I will go into more detail about possibilities to tackle this issue.

6 Conclusion and Future Work

In this chapter I will draw a conclusion, based on the literature and my own experimental findings. I will then draw a comparison to the studies I mentioned in Chapter 2. Section 6.2 will give an overview of the scientific and practical relevance of this thesis. I will conclude the chapter by doing suggestions for future research.

6.1 Conclusion

The research question as formulated in the introduction was as follows: Are word embeddings features, learned with an ANN-based learning technique suitable as a basis for hashtag suggestion systems? And more specifically: Does it yield better results than using just a baseline n-gram classifier?

As can be concluded from the discussion section, the word embeddings classifier proved indeed to yield better results than the n-gram baseline. However, the improvement was only small when comparing the baseline maximum with the word embeddings maximum: the F1-score raised from 0.17 to 0.18 (0.9 percentage point to be exact). The word n-gram baseline tends to benefit more from adding word embeddings than the character n-gram baseline. This can probably be explained by the weaker performance of the word n-gram baseline, as already mentioned in Section 5.3.6.

Character 3-grams plus word embeddings are the best performing conditions of all the tested conditions. This yielded an F1-score of 0.18. With more tolerant evaluation, as described in Section 5.3.8, the F1-score goes up to 0.31.

6.1.1 Comparison to other Studies

It is not always easy to compare the results of my own study to the results of other studies in the field, that are described in Chapter 2. Some of these researchers used different evaluation methods, which are hard to compare to the methods used in my own experiments. A global comparison can still be made with most of the studies though.

Godin et al. (2013) used a topic distribution system for hashtag recommendation. They mentioned a precision of 0.80, where the criterion was that one of the five top suggestions was appropriate, according to human judges.

Zangerle et al. (2011) used tf-idf based representations. They ended up with a precision of only 0.15. However, their recall was relatively high at 0.45, which yields an eventual F1-score of 0.23.

Li et al. (2011) used a system which combines semantical word similarity with tf-idf based ranking. This system performed with a precision of 0.86.

Kywe et al. (2011) combined personal user preferences with tf-idf based ranking. They ended up with a hit rate (which is kind of analog to precision) of 0.37.

Ding et al. (2013) used a topical translation model for their hashtag prediction system. They achieved a maximum F1-score of 0.36. Liu et al. (2011) used a similar technique, but also used word alignment to align tweet content. They achieved an F1-score of 0.37.

The evaluation in the aforementioned studies was relatively tolerant in most cases. Mostly, a classification was considered correct when one out of the top five suggestions matched the gold standard. This seems similar to my own more tolerant evaluation, where I divided hashtags in several groups of high semantical similarity (see Table 8). With this more tolerant evaluation, I ended up with a precision of 0.55. However, one still has to be careful to conclude that my system performs better than other systems. In the end, most of the other studies looked at the top five or ten hashtag suggestions provided by the system and looked if one of them matched to the gold standard hashtag. What I did in my more tolerant evaluation, was looking at all the hashtag suggestion, trying to fit it in a hashtag group, and only after that decide whether it matches with one of the gold standard tags for the given tweet.

The studies which used a topical translation model achieved the highest F1-scores of 0.36 and 0.37. It has to be emphasized however that these evaluation results are very hard to compare to those of other research: not only are the metrics and measurement criteria different, the quantity and nature of the used data also varies greatly.

6.2 Scientific and Practical Relevance

The findings of this research suggest that classifiers which make use of word embeddings features can indeed yield good results, which give improvement over a simple baseline. Nonetheless, there still is much room for improvement. It would be useful to do more research in this direction, possibly incorporating some of the suggestions that are mentioned in Section 6.3.

In practice, this kind of classification technique could be implemented in real life applications. In first instance, it would of course be interesting to implement this kind of functionality in Twitter. However, as already mentioned in Section 1.1, there is an increasing number of social networks in which users make use of hashtags. All these networks could benefit from a good performing hashtag suggestion system.

6.3 Future Work

In this section I will discuss the possible future work that could be done to further improve the performance of hashtag suggestion systems.

6.3.1 More Data

In general, one way to improve classification performance is to use larger amounts of training data. Logically, when more training samples are present, the overall performance of the system should improve. For instance, a training set of 1M tagged tweets instead of the current 15K could be used for training. It has to be noted that enlargement of the training set will result in longer training time though, approximately proportional to the increase in size. However, the longest training time with current settings was around 120 minutes per experiment, so training times would probably still be practically acceptable.

Not only the classifier will benefit from adding more training data to the training set. The word embeddings training algorithm (word2vec) will probably also be enabled a better performance when it is trained on a bigger corpus than the current 1M tweet dataset.

6.3.2 Improvement of the Classification Process

As already mentioned in Section 5.3.9, with current settings the classifier only classifies tweets when the candidate labels are above a certain probability threshold. In practice this means that a lot of the tweets remain unclassified. The advantage of this is that it accounts for better precision: the classifier will only attach a hashtag to a tweet when the probability value for that tag is very high. Indeed, the precision of the classifier is pretty high. The downside however is that recall stays behind. In future research it would be good to tweak the threshold settings in order to find the perfect balance between precision and recall (maximum F1-score).

6.3.3 Other Features

There are some other features that could be used for hashtag classification. For instance, user characteristics, as used by Kywe et al. (2011) could be an interesting opportunity to improve the performance of the system. Another option would be to take the time and date of the tweet into account. For instance, a tweet that is written around 6 PM will probably have an above average probability to have *dinner* as its topic.

6.3.4 Other Classification Techniques

In this research, a one-vs-the-rest classifier (OvR) is used. There are other classification techniques available though. One of these is the one-vs-one classifier (OvO). Instead of the OvR, the OvO constructs only one classifier per pair of classes. The OvR on the other hand fits one classifier per class, which is computationally more efficient, but usually less accurate than OvO. It would be interesting to see whether OvO would perform any better than OvR.

6.3.5 Other Languages

Possibly this research could also be expanded to other languages. There are a lot of other languages for which sufficient Twitter training data can be obtained. It would be interesting to see which hashtags are frequently used in other languages and whether the performance of the classifier would be similar for these languages as the performance for English tweets.

6.3.6 Other Amount of Hashtags

It is also possible to take another starting point regarding the amount of unique hashtags used in the experiments. For my experiments, I decided to use the 500 most frequent hashtags for training and the classification experiments. This is essentially an arbitrary amount. It would also be possible to start with a lower amount of hashtags (ie. 200). Chances are high that when using less hashtags, the performance of the classifier would go up, simply because the number of possible classes is reduced, which increases the chances of correct classification per sample. Increasing the number of hashtags is probably not very interesting for practical purposes, unless a bigger training corpus would be used. Additionally, it is thinkable that for many of the tweets that do now have a non-top-500 hashtag, an alternative top-500 hashtag could be provided that is equally appropriate for the given tweet. Further research should be done to be able to validate this claim.

References

[1] 4.1. feature extraction scikit-learn 0.14 documentation, http://scikit-learn.org/stable/modules/feature_extraction.html#sparsity, March 2014.

[2] Alexa - the web information company, http://www.alexa.com/topsites, February 2014.

[3] Illocution, http://illocutioninc.com/site/products-data.html, February 2014.

[4] word2vec, tool for computing continuous distributed representations of words, https://code.google.com/p/word2vec/, February 2014.

[5] Doland Antenucci, Gregory Handy, Akshay Modi, and Miller Tinkerhess. Classification of tweets via clustering of hashtags. 2011.

[6] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Roberto Lent, Suzana Herculano-Houzel, et al. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[7] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[9] Judith E Dayhoff and James M DeLeo. Artificial neural networks. *Cancer*, 91(S8):1615–1635, 2001.

[10] Fréderic Godin, Viktor Slavkovikj, Wesley De Neve, Benjamin Schrauwen, and Rik Van de Walle. Using topic models for twitter hashtag recommendation. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 593–596. International World Wide Web Conferences Steering Committee, 2013.

[11] Lichan Hong, Gregorio Convertino, and Ed H Chi. Language matters in twitter: A large scale study. In *ICWSM*, 2011.

[12] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. Wiley. com, 2013.

[13] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks: A tutorial. *Computer*, 29(3):31–44, 1996.

[14] Iqra Javed and Hammad Afzal. Opinion analysis of bi-lingual event data from social networks.

[15] Harry Jerison. *Evolution of the Brain and Intelligence*. Elsevier, 1973.

[16] Daniel Jurafsky and James Martin. Speech and language processing: an introduction to natural language processing, computational linguistics, and speech. 2000.

[17] Su Mon Kywe, Tuan-Anh Hoang, Ee-Peng Lim, and Feida Zhu. On recommending hashtags in twitter networks. In *Social Informatics*, pages 337–350. Springer, 2012.

[18] Tianxi Li, Yu Wu, and Yu Zhang. Twitter hashtag prediction algorithm. In *ICOMP11-The 2011 International Conference on Internet Computing*, 2011.

[19] Allie Mazzia and James Juett. Suggesting hashtags on twitter, 2009.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[21] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048, 2010.

[22] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[24] Jan Pöschko. Exploring twitter hashtags. *arXiv preprint arXiv:1111.6553*, 2011.

[25] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[26] Eva Zangerle, Wolfgang Gassler, and Günther Specht. Recommending #-tags in twitter. In *Proceedings of the Workshop on Semantic Adaptive Social Web (SASWeb 2011). CEUR Workshop Proceedings*, volume 730, pages 67–78, 2011.