Emoticon Prediction in Text

Iva Gornishka
iva.gornishka@student.uva.nl
10415548

Cassandra Loor
cassandra.loor@student.uva.nl
10624635

Finde Xumara
finde.findexumara@student.uva.nl
10690832

Arif Qodari
arif.qodari@gmail.com
10711996

Abstract

Emoticon prediction in text is a problem closely related to opinion mining, sentiment analysis and reputation extraction. This article discusses the technique of predicting emoticons in a sequence of messages. This sequence of messages contains multiple messages which are related to one another. Using the predicted emoticon of the previous message, the emoticon of a message can be predicted. This lead to the use of a Hidden Markov Model and the Viterbi decoding algorithm.

1 Introduction

This report discusses emoticon prediction, based on a sequence of messages. In order to find messages that are related to one another and can transition in positivity to negativity, an entire conversation is observed. Each message is classified by not only considering the features of a single message, but also by considering the previous messages of the same user. This problem is complex and interesting one, since it also takes into account the transitions from one state to another.

In order to find conversation messages with such transitions, a corpora containing longer sequences of text messages are used - these might be conversations, such as an e-mail exchange or simply chat data, as long as the corpora contains a sufficient amount of emoticons. As emoticons appear in chat data more often, Ubuntu chat data is used. Unfortunately, due to the lack of emoticons in this corpora, the emoticons of all other sentences must also be determined. In order to label the corpora, another model must be trained. In order to train this model, a corpora of short, unrelated messages filled with emoticons is used. In order to find such messages, Twitter data is used. Therefore, the Twitter data is used to train a model. This model is then used to label the Ubuntu chat data.

After the data is collected, it is first preprocessed. Then, appropriate features are extracted and a model is trained. The current report does not discuss the choice of features in detail. Basic features were chosen for this project without intentional aim on improving them. Since this task on its own can take up years, it is left as future work. However, suggestions about improving features is mentioned in the following sections.

In order to classify the data, classes must be determined. Three classes are used to classify the data. Table 1 displays these classes and the corresponding emoticons, contained in this type of data.

As table 1 shows, this selection of emoticons displays positive, negative and neutral emotions. However, this data can be split further into extremely positive (:D) and extremely negative (>: ( ) emoticons for more specific classification of messages. However, this is beyond the scope of this paper.

This project is part of the Natural Language Processing 1 course taught at the University of Amsterdam in 2014-2015

Table 1: Data classes

DATA CLASS	EXAMPLE EMOTICONS
positive	:) :] =) :D
neutral	:| o_o
negative	:(:[=(;(

2 Problem

The intent of this research is to determine the transition of emoticons in a conversation. Such transitions are found in sequential messages. As stated before, chat data from Ubuntu is used to analyse transitions. This leads to the following problem:

How can emoticons be predicted in a sequence of messages, using the emoticon of the previous message?

This leads to several, obvious, steps that must be taken. These are:

- Preprocess corpora
- Extract features
- Implement a model

However, the selected corpora is not optimal. The Ubuntu chat data does not contain enough emoticons. Therefore, in order to train a model, each message must be classified. This leads to implementing another model which labels the Ubuntu chat corpora. This model must determine the emoticon of a single, independent message. This means that a different model must be trained on a different corpora. Using Twitter data, the use of emoticons can be used to train a such a model. This model will be used to label the Ubuntu data. Then, the emoticons of the sequential messages must be predicted.

As stated before, both corpora must be preprocessed. As the messages in both corpora are written by users in their spare time, the presence of grammatical- and spelling errors and slang is quite real. There are also other sub-optimal aspects of the data that must be handled. After preprocessing, the corpora are ready for feature extraction. These subjects will be discussed where necessary.

When training a model to predict emoticons on independent messages, there are several models that can be used. A support vector machine (SVM) and perceptron can be used. Therein, different perceptrons can be used. For this research, a perceptron was used. Two perceptrons were implemented: the average multiclass perceptron (AMP) and the multilayer perceptron (MLP). The AMP can be used to classify the data. However, the MLP can yield better results due to the presence of hidden layers. This research will show which yields best results.

The model used to predict emoticons on sequences is the Hidden Markov Model (HMM). There are two versions of the HMM: the first-order HMM and the second-order HMM. The first-order HMM uses the emoticon of the previous message to predict the emoticon of the message that is being analysed. The second-order HMM uses the two previous messages to predict the emoticon of the message that is being analysed. To conduct this research, the first-order HMM is used.

It is expected that the two models will work separately. Even if the first model, used to relabel the sequence-based data, performs poorly, the performance of the HMM will not be affected. As the HMM is based on mislabelled data, it will learn these labels and continue building upon this. This will lead to sub-optimal results, overall, but the HMM will still perform well. Also, the sequence-based data in itself is slightly biased. As this data was collected from a support forum, it is expected that most data will be negative. This can lead to biased prediction. This must be prevented, by creating a balanced dataset of positive, negative and neutral data.

The dataset must be balanced for both models. However, creating a balanced dataset is easier the single-message prediction as there are no transitions there. This means that selecting data to train the HMM on must be performed very well to create a purely balanced dataset.

To summarize: in order to predict emoticons in sequence of messages, the positivity, neutrality or negativity of each message must be determined. By preprocessing each message and extracting features, the ”gist” of each message is extracted. This ”gist” is used to train a first-order Hidden Markov Model, which can then be used to predict the

emoticon of messages. However, in order to label the training data correctly, first all messages must be labelled. To do this, a perceptron is used. When training all of the models, it is imperative to select balanced data to train each model in order to achieve optimal prediction.

3 Emoticon prediction based on short messages

This section discusses how Twitter data is used to train a classifier to predict emoticons.

3.1 Data

In order to collect data from Twitter, a crawler was used. The Twitter crawler searched for training and test data, based on emoticons. These messages ('tweets') contain much information - the name of the author, text, hyperlinks, hashtags (sequences of the form '#word_sequence'). As there are no official restrictions, messages often contain plenty of intentional or - more often - unintentional spelling and grammar errors. Messages also sometimes contain words from other languages, which makes preprocessing and analyzing the data even harder.

3.2 Preprocessing

Before any of the data can be used, the messages need to be preprocessed. This process includes three basic steps. Each one of them is discussed in more details in the following paragraphs.

Splitting the message.

The message, itself, is split into different components that will help in training. In this case, the text, hashtags and emoticons are extracted. Emoticons in messages are used to add emotion, but not text. The text itself can be used to determine whether a message is positive, negative or neutral. Therefore, the text without the emoticons can be used to determine which emoticon can be assigned to a certain message. Also, the hashtags can be used to further determine the emotion of the message. Hashtags often contain extra information, such as 'sad' or 'ecstatic'. This can also help determine the emotion of a message. In this case, however, the hashtags were not used. Therefore, it is important to research the effect of this feature in the future.

Spelling and grammar correction.

Short messages are often posted as a reaction or something to share with friends quickly. This means that many messages contain spelling and grammar errors. In order to optimally use the Twitter data, all messages must be fixed spelling and grammar wise. There are several ways of performing these corrections, however these fall beyond the scope of this research. Therefore, names, links, emoticons and hashtags are disregarded when processing data, but the spelling and grammar errors remain. It is imperative to solve these errors in the future for optimal performance.

3.3 Model

A message can be classified as a positive, negative or neutral emotion. In order to classify these messages, an average multi-class perceptron and a multilayer perceptron were implemented. A multiclass perceptron is an algorithm that allows supervised classification. As the name of the algorithm implies, the input does not have to be binary allowing input to be classified as one of many classes. As the Twitter data has to be classified as positive, neutral and negative before assigning an emoticon to the message, this algorithm does as desired.

Basically, a multiclass perceptron finds a border between two classes and draws a border between the two classes. By repeating this, several borders will appear between classes and are used to classify input. This leads to certain "gray"-areas, as different borders assign input data to different classes. In the case of the average multiclass perceptron, the average border of all is computed. This creates one border, without any "gray"-areas.

As for the multilayer perceptron, hidden layers are added to the perceptron. These hidden layers help classify the data more accurately. Unfortunately, this model is very sensitive to change. Therefore, tuning the model is very difficult. This has lead to an unstable model, leading to the use of the AMP to label the sequential messages.

3.4 Features

Once the data is collected and preprocessed, its features are extracted. Selecting good features can significantly improve the performance in many cases, but this task is not a trivial one. Due to time constraints, certain basic features were selected to conduct this research. However, it is believed that research conducted in the future can explore a bigger (or different) set of features. Table 2 presents the features which were extracted and used for classification. Since most of the features' names are self-explanatory, only the some of the features are discussed in details in the following paragraphs.

Table 2: Extracted features

FEATURE NAME	DESCRIPTION
words	total number of words in the text
positive words	total number of positive words in the text
negative words	total number of negative words in the text
positive words hashtags	total number of positive words in the hashtags
negative words hashtags	total number of negative words in the hashtags
uppercase words	number of words containing only uppercase letters
special punctuation	number of special punctuation marks such as '!' and '?'
adjectives	number of adjectives in the text

Extracting features such as **positive words** and **negative words** requires the usage of a predefined lists of 'positive' and 'negative' words, which were downloaded from [5]. Both these lists contain words that are positive or negative and are used to classify the emotion of a short message. While implementing this aspect, the question of what effect extremely positive and extremely negative words have arose. Due to time constraints, this could not be researched. However, this is interesting to research in the future.

Features, such as the number of **adjectives**, were extracted by first performing part-of-speech (POS) tagging on each sentence and then counting the adjectives in the tagged sentence. Due to spelling and grammar errors in the messages, POS tagging does not always perform well. As a consequence, slight inaccuracies are possible for this feature, though not significant enough to negatively influence the performance.

3.5 Experiments and Results

As the AMP needs to be used to label data, different experiments were run to test the performance. Sample results are shown in table 3 and are be discussed in detail in this section.

3.5.1 Prediction with varying number of classes

To see how well the perceptron predicts data on a varying number of classes, the algorithm was run while predicting multiple classes. Initially, it is interesting to see how well the data is classified using two classes; positive and negative. There is also a third class representing neutral data. There are also classes which are defined as extremely positive and extremely negative. In total, these are 15 classes. The performance of the model is also tested in this number of classes. To test the model, K-fold validation is used. Using 80% of the data, 20% is used as a test set to grade the performance of the model. In order to ensure a proper resemblance of test results, the results are averaged over 10 tests.

Classes	Data per class	Iterations	Train Accuracy	Test Accuracy
2	500	50	52.60	51.95
3	500	50	38.93	37.73
15	500	50	10.62	9.01

Table 3: AMP accuracy

Classes

Unsurprisingly, the results show that the more classes are used the worse the performance becomes.

In order to test the performance on two classes, the messages containing different'happy' and 'sad' emoticons were split into two general groups - 'negative' and 'positive' data. Since the data in these classes is significantly different, especially with respect to features such as count of positive or negative words, distinguishing differences seems to be much easier.

For testing the performance on three classes, data containing no emoticons or neutral emoticons were added to the training set as 'neutral' data. In this case the performance dropped dramatically. The lack of an emoticon in a message does not necessarily mean that this message is not a positive or a negative one, which is reflected in the results. For example, some people do not use emoticons to express their emotions, but hashtags instead. Thus, neutral data is often classified mistakenly as positive or negative or vice versa, which significantly decreases the overall performance of the perceptron.

For testing purposes, the data was eventually split into 15 different classes, each of them containing messages with a different emoticon - i.e. messages containing emoticons such as ':)' and ':D' were not both considered 'positive' anymore, but treated as different classes instead.

Features

The effect of different features was also explored. In order to see clear results, two classes were used to classify the data. These results are displayed in table 4.

Features Excluded	Data per class	Iterations	Train Accuracy	Test Accuracy
None	500	50	52.60	51.95
Positive/negative words	500	50	53.88	52.90
Positive/negative hashtags	500	50	54.40	53.70
Amount of words	500	50	51.78	50.90
Uppercase words	500	50	51.51	49.65
Special punctuation	500	50	51.91	51.25
Adjectives	500	50	52.39	50.25

Table 4: AMP accuracy

Tests show that some of the features are not meaningful enough and including them only decreases the performance. For example, the use of hashtags should improve the performance, but it appears that when emoticons are used, hashtags aren't. Therefore, the features that represent the hashtags are mainly empty. This does not help distinguish positive data from negative data and should be excluded from the feature set. As this model is used to label data which does not contain hashtags, this should not make a difference. Also, the sequence-based data hardly contains words represented solely in uppercase. Therefore, this feature can also be excluded from the feature set. This leads to the following results

Features Excluded	Data per class	Iterations	Train Accuracy	Test Accuracy
Positive/negative hashtags Uppercase words	500	50	56.96	56.25

Table 5: AMP optimal accuracy

Excluding the positive and negative hashtags, as well as the uppercase words, yield the best results and will be used in labelling the sequence-based data.

4 Ubuntu emoticon prediction

4.1 Data

In this part of the project the Ubuntu Chat Corpus was used as training data *SOME REFERENCE HERE*. This corpus consists of archived chat logs from Ubuntu's Internet Relay Chat technical support channels *Reference*. The logs contain continuous conversations, where numerous people take part in the conversation. Some people only post a few messages, whereas others participate more actively in the conversations.

Relabeling the data

Since this data contains mainly technical conversations, there appeared to be a significant lack of emoticons. This means that the majority of the messages were labeled as 'neutral', which in turn implies a terrible prediction performance. In order to still be able to use this data and test the prediction performance, a decision was made to relabel the data. For every 'neutral' message, an emoticon was predicted using the AMP discussed in section 3. This lead to the possession of data containing enough emoticons to properly train a model and predict new examples. In order to avoid such artificial relabeling of the data in other projects, corpora from a more appropriate source should be used - e.g. Skype conversation logs or any other group chat logs.

4.2 Preprocessing

Since the Ubuntu Chat corpus is well structured, and hardly any hashtags, hyperlinks, capitalized sentences or other special message parts appear, preprocessing the data would not significantly increase the performance. This lead to the decision that the preprocessing step for this data set would be omitted for the purpose of this project.

4.3 Model

To be modified and corrected

A first-order Hidden Markov Model was chosen as most appropriate for the purpose of this project. This section summarizes the steps which were taken in order to train the model and predict classes for unseen examples afterwards.

As explained earlier, after the data has been pre-processed, a feature vector is extracted for each example message. This feature vector is similar to the feature vector extracted from the Twitter data, excluding hashtags and uppercase words. Thus, the training data for the model consists of pair (X, y), where X is a feature vector and y is the corresponding class label. The next step is to compute the transition and emission probabilities, i.e. the probability of transitioning from one class state to another and the probabilities of the different feature vectors given a class state.

Consider the following: the probability of observing a "negative" message right after a "neutral" message has been observed is a transition probability. Furthermore, the probability of the feature vector of the message 'I need to get some sleep now' given the neutral class is an emission probability. This means that the following general formulas might be used in order to calculate these probabilities:

$$\mathcal{P}(\textit{transitioning from class y' to class y"}) = \frac{\textit{\# of times class y" is observed after class y'}}{\textit{total \# of training examples}}$$

$$\mathcal{P}(\textit{feature vector } X \textit{ given class } y) = \frac{\textit{\# of times the pair } (X, y) \textit{ occurs in the training data}}{\textit{total \# of training examples from class } y}$$

Since the total number of feature vectors X which can be observed is too big, not all possible vectors would be observed in the training data. This would make predicting unseen messages impossible, since no emission probabilities would be present for these vectors. In order to avoid this issue, the following solution is implemented: the training data is first clustered, emission probabilities are computed for each cluster, then given an unseen message, its feature vector is first assigned to a cluster and finally, a label is predicted for this message.

4.4 Features

The corpus of this section does not contain hashtags and hardly contains uppercase words. Therefore, these features are excluded from the feature set. However, all other features described in the previous section are considered.

4.5 Experiments and Results

Different experiments were run to test the performance of the Hidden Markov Model. Sample results are shown in tables and are be discussed in more details in this section. Table 6 shows the accuracy of two classes (positive and negative) and all three classes. *RESULTS NOT CHANGED FOR 2 CLASSES AS THIS IS BROKEN*

Classes	Features	Data per class	Clusters	Train Accuracy (%)	Test Accuracy (%)
2	all	500	50	73.65	72.66
3	all	500	50	39.23	38.37

Table 6: HMM accuracy

Classes

Table 6 shows that the use of two classes yields best results. However, this happens due to the fact that the results are biased. As the chat data comes from a support forum, most messages are indeed negative or neutral. As the emission and transition values are calculated by counting labels, the HMM itself will most likely transition to negative or neutral. This leads to data labelled as positive always being correct. However, this does not happen often, due to the aforementioned calculations. Finding a better way to select data to train on will be imperative in the future in order to produce better results. As the data of three classes is least biased, this will be used in further testing.

Features

As with the AMP, the effect of the features on the HMM were tested. As this model trains on a sequence of messages, it could be that the features have a different effect.

Features Excluded	Classes		Iterations	Train Accuracy	Test Accuracy
None	3	50	39.23	38.37	
Positive/negative words	3	50	38.69	38.62	
Amount of words	3	50	38.61	38.02	
Special punctuation	3	50	38.81	38.25	
Adjectives	3	50	38.66	37.73	

Table 7: AMP accuracy

Table 7 shows that the exclusion of more features shows no improvement. This confirms the suspicion of poorly selected features. However, this does confirm that the current selection of features works optimal and should be used for further testing.

Amount of clusters

Different amounts of clusters were used to test their effect on the model. The results are displayed in 8.

Classes	Data per class	Clusters	Train Accuracy (%)	Test Accuracy (%)
3	500	10	38.27	37.94
3	500	20	38.53	38.36
3	500	30	39.00	37.45
3	500	40	38.99	38.22
3	500	50	39.23	38.37

Table 8: HMM amount of clusters

The results in 8 shows that an increasing amount of clusters yields yields varying results. This could mean that the model overfits. However, there is not much difference between implementing few clusters and many clusters. The fact that the model does not yield good results can be attributed to the fact that the selected features are not sufficient. The minimal differences between the results introduces the suspicion that there is a bug in the implementation. However, due to time constraints, this bug was not identified and fixed.

5 Conclusion

The selected corpora for this research was not sufficient and needed labelling in order to find the positivity, negativity or neutrality of a message. This means that the selected corpora was not sufficient for the research at hand and must be selected better in the future.

The preprocessing of the corpora was effective, but can be improved in the future. Minimal correction grammar- and spelling errors was sufficient enough to find positivity, negativity or neutrality in a message. Perhaps when extreme positivity and negativity are also taken into account, these corrections need to be more specific. For the this research, this was enough.

All tests have shown that the selected features help predict emoticons. However, when excluding features, no exclusion lead to a dramatic rise or drop in accuracy. This means that the selected features are not sufficient and other and additional features must be determined for better performance.

When selecting the amount of clusters to determine the transition and emission probabilities, an increasing amount of clusters leads to worse results. It appears as though the model overfits. However, as the best performance and the worst performance with a certain amount of clusters hardly seems to make a difference, it is believed that there is a bug in the the implementation. Due to time constrains, it isn't possible to find this bug and to solve this problem.

6 Future work

6.1 Data selection

As discussed in section 6.3, the Ubuntu Chat Corpora does not contain a sufficient amount of emoticons for training. It has not been proven whether the artificial relabelling of the data has had an impact on the performance of the Hidden Markov Model, but it is believed that tests should be also performed using another corpora in order to compare the performance. Also, this alternate dataset should be balanced in such a way that there are equal transitions between positive, negative and neutral. This will lead to more general transition and emission probabilities. There sequence-based corpora that can be used. Facebook and Skype often contains the information that is necessary as stated here. However, this is not open and free to use, which makes finding a sufficient corpora difficult.

6.2 Feature selection

As mentioned earlier in this paper, feature selection is crucial for the emoticon prediction in text, just like it is in every prediction problem. It is believed that further exploration of the feature space could lead to much better results. A bigger set of features is discussed in [2]. Although a different type of data is used for this research, some of the discussed features might be applicable in other settings too. Last but not least, a set of features for sentiment analysis in Twitter data is discussed in [1].

6.3 Model selection

For the purpose of this project an Average Multiclass Perceptron (AMP) and a Hidden Markov Model (HMM) were selected as most appropriate. It has not been tested whether different models yield better performance, hence a future work in the area might include exploring different models. Different sources, such as [4], suggest using an SVM or Naive Bayes instead of the AMP for emotion prediction, and we believe these could be also appropriate for emoticon prediction. Furthermore, using a second order HMM could improve the performance for predicting whole sequences.

Team responsibilities

The four authors contributed equally to this project.

References

[1] Apoorv Agarwal et al. “Sentiment analysis of twitter data”. In: *Proceedings of the Workshop on Languages in Social Media*. Association for Computational Linguistics. 2011, pp. 30–38.

[2] Cecilia Ovesdotter Alm, Dan Roth, and Richard Sproat. “Emotions from text: machine learning for text-based emotion prediction”. In: *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2005, pp. 579–586.

[3] Edward Loper Bird Steven and Ewan Klein. *Natural Language Processing with Python*. 2009, publisher =.

[4] Taner Danisman and Adil Alpkocak. “Feeler: Emotion classification of text using vector space model”. In: *AISB 2008 Convention Communication, Interaction and Social Intelligence*. Vol. 1. 2008, p. 53.

[5] Minqing Hu and Bing Liu. “Mining and summarizing customer reviews”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, pp. 168–177.

[6] Saif M Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. “NRC-Canada: Building the state-of-the-art in sentiment analysis of tweets”. In: *arXiv preprint arXiv:1308.6242* (2013).

[7] David Uthus and David Aha. *The Ubuntu Chat Corpus for Multiparticipant Chat Analysis*. 2013.

This is example reference. Include the real ones in the .bib file

Ubuntu Chat Corpus

The Ubuntu Char Cospus was used in this project as train and test data. More information about the Ubuntu Chat Corpus might be found at `http://daviduthus.org/UCC/`. Furthermore, details about the corpus might be also found in [7]

Natural Language Toolkit (NLTK)

The Natural Language Toolkit for Python was used in this project during the feature extraction stage. More information about it might be found on `http://www.nltk.org/`. Furthermore, [3] provides a practical introduction to programming for language processing.