# Emoticon Prediction in Text

[1]

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

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

**Finde Xumara**
finde.findexumara@student.uva.nl
ID

**Arif Qodari**
arif.qodari@gmail.com
ID

## Abstract

*Refine the abstract, maybe add most important results, a bit too much 'lead'*
Emoticon prediction in text is a problem closely related to opinion mining, sentiment analysis and reputation extraction. This article discusses two different techniques for predicting emoticons. In the first case, emoticons are predicted based on a single, short message. This lead to the use of an average multiclass perceptron. In the second case, emoticons are predicted, using messages posted in the past. This leads to the previous message also determining the emotion of the message analyzed. This lead to the use of a Hidden Markov Model and the Viterbi decoding algorithm.

## 1 Introduction

This report discusses two related, but essentially different problems - emoticon prediction based on a single short message and based on a sequence of messages. In the first problem, short, unrelated messages are observed. Their emoticons are used to determine whether a message is positive or negative. Using this information, a is trained. Eventually, a new message can be passed and the corresponding emotion is predicted. In order to determine the sentiment of a message optimally and achieve generalization, messages of different authors are collected and used to train the classifier. In the second problem, 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 a more complex and interesting one, since it also takes into account the transitions from one sentiment state to another.

As a natural consequence of the above statements of the two problems, different data sets are used to approach each problem. For the first problem, any corpora of text messages containing emoticons can be used. Such corpora can contain unrelated messages. In order to find short unrelated messages, Twitter data is used. For the second problem, 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.

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 feature 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.

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

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.

Table 1: Data classes

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

As table 1 shows, known emoticons display 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 and is further mentioned in future work. . . . . . . . . .

*Finish the introduction, maybe move some parts to the Problem section*

## 2 Problem

*Find out what should be actually in this section*

## 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 averaged multi-class perceptron was 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 multiclass perceptron, the average border of all is computed. This creates one border, without any "gray"-areas.

### 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 *REFERENCE*. 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

## 4 Ubuntu emoticon prediction

### 4.1 Data

In this part of the project the Ububntu 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 couple of messages, whereas others take more active part in the conversations.

### 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 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. 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 a simple example: the probability of observing a 'sad' massage 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 features used in this part of the project correspond to the ones already discussed in section 3.4. Note that some features, such as numbers of positive or negative words in hashtags are not applicable in the current setting, since the data does not contain hashtags. Thus, such features have not been considered.

### 4.5 Experiments and Results

## 5 Conclusion

## 6 Future work

*Explore features, compare different models and parameters*

## Team responsibilities

## References

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

[2] George D. Greenwade. “The Comprehensive Tex Archive Network (CTAN)”. In: *TUGBoat* 14.3 (1993), pp. 342–351.

[3] 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*

## 7 Ubuntu Chat Corpus

More information about the Ubuntu Chat Corpus might be found at `http://daviduthus.org/UCC/`