# Emoji Recommendation

By William Dahl

## 1. Abstract

Emoji's are used everyday, hundreds of times a day, and by millions of people. The current library of emoji choices is at 2,823[1]. Searching for the perfect emoji to help convey your messages point may be a bit difficult. An application that recommends the perfect emoji to use maybe helpful in situations such situations. This paper explores the application of both a self made modular regression model and the use of the Linear SVM classify offered by "sklearn" to the recommendation of emojis based on a given text message. The use modular regression gave a testing accuracy of 50% and the use of Linear SVM gave a testing accuracy of 58%. The training accuracy results for the modular regression yielded 100% accuracy and likewise, the Linear SVM model yielded an accuracy of 99% on the training data. This pointed to a possible over fitting of the data which could have resulted in poor performance than otherwise. Another, possible source for error would be in that punctuation is not counted for as well as other symbols commonly used in text messages such as the "#". In the end the Linear SVM model was chosen as it had the better overall performance.

## 2. Introduction

The goal of this project is to recommend an emoji for someone to send along with their text message. Adding this feature to any messaging app will result in better usability and a better user experience causing more people to want to use it. A challenge in this project is the analysis of the different meanings of text messages as used by humans to communicate. Working around things such as satire, text message shorthand, and the use of punctuation to accent a persons emotions can be difficult. Cultural changes in language can also provide a hurdle to analyzing text dialog between humans. Another challenge In this project was the breaking up the data to get useful features out of them and how to model the classifications of the data so that they could be used together within the model. A personal challenge for me in this project was that it was my first time using the library's offered by "sklearn". Thus a large amount of research needed to be done into these library and to learn how they can be used. This is evident in my "Mode_Regression_model" code. Here I made everything from scratch. I separated the data file into separate training and testing files as well as separated the messages and their labels. After researching "sklearn" I was able to use a lot of their own methods when implementing the Linear SVM model, making my code much cleaner and simpler.

## 3. Related Works

Within recent time, messages apps have begun to recommend emoji use. These platforms include apps such as imessage, Facebook Mssenger, Whats-app, and Snapchat. The paper "Neural Emoji Recommendation in Dialog Systems" by Ruobing Xie, Zhiyuan Liu, Rui Yan, and Maosong Sun of the National Lab for Information Science and Technology, Tsinghua University, Beijing, China and the Natural Language Processing Department, Baidu Inc., China Try to use Neural networks as a method for classifying dialog between two people and then recommend an emoji[2]. The app Dango uses Deep learning techniques to recommend the top 4 emojis to use given a message typed by the user, the app even extends this to gifs and memes as well[3]. DeepMoji is an open source AI project where when given a word message will recommend 5 emojis to use in its place[4]. It even has the ability to understand sarcasm and has access to millions of emojis.

## 4. Methods

At its roots, this is a semantics analysis problem. The Goal of the model to use the features that are extracted from the messages and predict the sentiment of the message. To begin getting features from the message I broke the message down into n-grams (a grammar that of length n). Each message was broken up into a series of 1-grams, 2-grams, …, 4-grams.

### 4.1 Mode Regression

These grammars are saved as keys into a dictionary where  the class of the message that it appearances in is saved as the values. The classes are stored into an array and the class that appears the most, the mode, is the end value for that n-gram. When a message is classified using the "Mode regression" model it is broken down into n-grams. Each n-gram is classified independently and its class is stored in an array along with its weight. The overall class of the message is then computed and predicted.

### 4.2 Linear SVM

In this model the n-grams are stored into a dictionary with the n-gram being the key and the number of times the n-gram appears in other n-grams as the value. The dictionary is then vectorized  using a DictVectorizer. The vectorizer is fit to the features and then the features are transformed into a vectorized dictonary. The vectorized dictonary is then used to predict the class of the message using the Linear SVM model.

## 5. Experiments

I received my data from the DeepMoji git repository[5]. There are 7480 text messages in this data set. The Data is organized with a vector of length 7. Each index within the vector represents a different emotion such as [joy, fear, anger, sadness, disgust, shame, guilt]. And then followed by the respective message.

### 5.1 Mode Regression

#### E.x:

[ 1.  0.  0.  0.  0.  0.  0.] During the period of falling in love, each time that we met and especially when we had not met for a long time.

The above example has a joy sentiment.

My first step was to separate the messages from their labels so that I could treat the labels as a matrix as well as divide my data set into my training data and my testing data. I did this by doing a 80% and 20% split respectively.

The next step was to create n-grams. These are a combination of up to n words in a message.

#### E.x:

In the message "I love you" a 1-gram would be ['i', 'love', 'you]. A 2-gram would be ['i love', 'love you'] and 3-gram would be ['i love you'].

For each message I created  n grams from 1 to 4 in order to account for negation such as ‘happy’ versus ‘not happy’ and ‘love you’ versus ‘don't love you’.

Next was to classify the n-grams. I did this by analyzing the mode of the occurrences of a phrase or word trough out the  training data and it’s label.

### E.x:

the class distribution for the phrase “happy today” may look like this [0,1,2,3,4,5,6,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,3,0,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,]

here the mode of the class distribution is 0 which maps to the feeling of joy.

Next is to classify a given message. I accomplish this by breaking the message into n-grams of size 1 through 4. I then compare the n-grams from the message to my already classified ones and create a distribution of possible classes to the message. I also apply a larger weight to the longer n-grams as clearly ‘not happy’ should weight more than ‘happy’. I accomplish this by putting the class for the phrase into the class distribution n times.

### E.x:

“i am so happy” will have a class of 0 (joy) and is a 4-gram thus 0 will be written into the class distribution 4 times

I then take the mode of the class distribution and use that as my predicted value for the class of the message.

## 5.2 Linear SVM

For the Linear SVM model I kept all the data together in a matrix 7480x2 matrix with the classification in the first column and the text message in the next.

### E.x:

The first column of my data matrix:

```
['1. 0. 0. 0. 0. 0. 0.', ' During the
period of falling in love, each time
that we met and especially when we had
not met for a long time.\n']
```

Next, was to create a vector of dictionaries for each message and a vector of the classes for each message.

### E.x:

The Dictionary created for the message “i am happy”:

```
{'i': 2, 'am': 2, 'happy': 2, 'i am': 1,
'am happy': 1, 'i am happy': 1}
```

Here the key is each n-gram created when extracting the features of the message. The value is the number of times the n-gram appears in other n-grams through out the message. As you can see ‘i’ has the value of 2 as it also appears in the n-gram ‘i am’ as well as the n-gram ‘i am happy’. To accomplish this I used the Counter module[6].

The class for this message as the vector would be:

[1 0 0 0 0 0 0]

I mapped this to a string representations of the classification so in the class vector the value would be “joy”.

Next, I split up my data into a training and testing set, with a 80% training and 20% testing split. To accomplish I used the “train_test_split()” function from the “sklearn.model_selection” module.[7]

I then vectorized the and fit the message dictionaries using the “DictVectorizer()” fucntion from the “skleatn.feature_extraction” module[8].

### E.x:

After being vectorized the dictionary for the message “i am happy” is in the form of:

```
(0, 23667)    2.0
(0, 116642)   2.0
(0, 129264)   2.0
(0, 129485)   1.0
```

The first number in the tuple reference the message number, in this case it is 0 as it is the only message given. The 2nd number in the tuple is the number of times the n-gram appears in the the entire testing set. The number with the tuple is the number of times the n-gram appears in the other n0grams in the message.

After the message dictionaries have been vectorized, the Linear SVM model is fit with them and the classes of each of the messages. I used the “LinearSVC()” function from the “sklearn.svm” model[9].

## 6. Results

My initial results using the “Mode Regression” method was a 100% accuracy when applied to the training data and a 50% (374 out of 784) accuracy when applied to my testing data. This implies that my model is probably over fit. Similarly, the Linear SVM model had a training accuracy of 99% and a testing accuracy of 58%. This model is also probably over fit then.

Possible sources of error could be due to my neglect of punctuation of and special symbols may have also lead to a loss in precision as “happy” and “happy!” will not be seen as the same word in the model due to the punctuation.

Also, My attempt to consider negation on adjectives appears not to have been successfully as phrases like “not happy” and “not sad” still map to joy and sadness respectively.

As the Linear SVM model had the better overall performance when compared to the Mode Regression model I decided to use the Linear SVM model for the actual application. First I train a new model using all of my data and not just the training data that I had before to hopefully make a better performing model. I also re fit my DictVectorizer to fit all of the messages from of data set. In order to increase efficiency so that the model does not need to be trained every time the app is run, I dump both the trained Linear SVM model and the DictVectorizer into a pickle file using the “pickle” module[10]. When the app is ran it will check if the Linear SVM model already exist with in a file and then run the GUI application using it for the the user to enter message to get a recommended emoji.

## 7. Conclusion

In order to Fix some of the errors with in the current model, I would have to work on generalizing it more. Normalizing the data matrix could help to generalize the data more and hopefully increase the accuracy on new data. Another way to possibly increase the accuracy of the model would be to account for punctuation so that less separate keys are created so that there are more hashable objects.

Emoji recommendation serves as way to further an understanding of sentiment analysis and natural language processing. With the vast and ongoing expansion of the emoji library a emoji recommendation feature is almost necessary. It is a useful feature to add to any messaging application and can add to your users experience.

## 8. References

[1] " FAQ." *Emojipedia FAQ*, emojipedia.org/faq/.

[2] Xie, Ruobing & Liu, Zhiyuan & Yan, Rui & Sun, Maosong. (2016). Neural Emoji Recommendation in Dialogue Systems.

[3] Snelgrove, Xavier, and Whirlscape CTO. “Your Emoji Assistant.” *Dango*, getdango.com/emoji-and-deep-learning/.

[4] MIT Media Lab. “DeepMoji.” *DeepMoji*, deepmoji.mit.edu/.

[5] Bfelbo. “Bfelbo/DeepMoji.” *GitHub*, 9 Sept. 2018, github.com/bfelbo/DeepMoji.

[6] “8.3. Collections - High-Performance Container Datatypes¶.” *8.3. Collections - High-Performance Container Datatypes - Python 2.7.16 Documentation*, docs.python.org/2/library/collections.html.

[7] “Sklearn.model_selection.train_test_split¶.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[8] “Sklearn.feature_extraction.DictVectorizer¶.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html.

[9] “Sklearn.svm.LinearSVC¶.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.

[10] “Pickle - Python Object Serialization¶.” *Pickle - Python Object Serialization - Python 3.7.3 Documentation*, docs.python.org/3/library/pickle.html.

[11] TetsumichiUmada. “TetsumichiUmada/text2emoji.” *GitHub*, github.com/TetsumichiUmada/text2emoji/blob/master/project.ipynb.