Emojination: An Approach to Predict Emojis for Tweets

Varun Sharma
varunsharma@umass.edu

Hisham Alhussain
halhussain@umass.edu

1 Abstract

Emojis are unique symbols that, when combined with words, help condense meaning, provide visual context, and more meaning to a sentence overall. Despite emojis usage growing tremendously in the past few years, and diffusing across various social media platforms, studies about emojis have received little attention from the Natural Language Community. In this paper, we try out various approaches to classify a given tweet by supplying one of the twenty most commonly used emojis on twitter. Our best performing model turned out to be a Bi-directional LSTM, and surpassed our baseline model by a large margin. We conclude the paper by providing some insight into our results, and proposing future work in the area.

2 Introduction

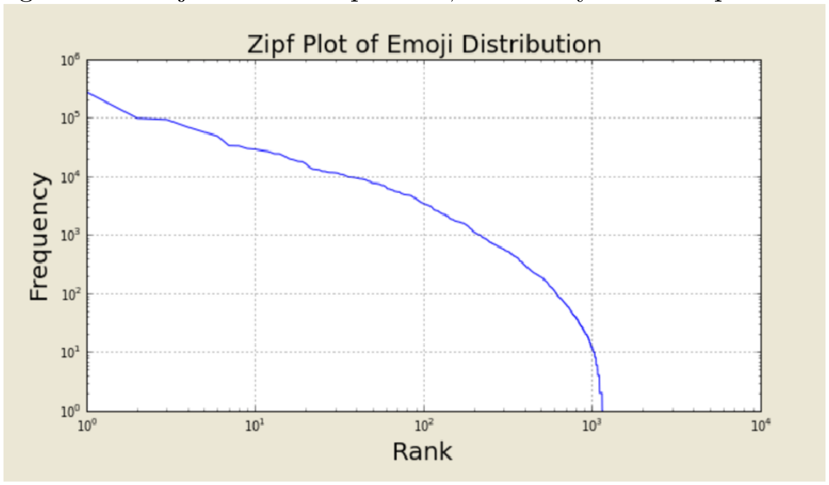
With the social media boom taking place in recent years, emojis have become a part of our everyday lives. Over ten percent of tweets contain emojis [6], and the face-with-tears-of-joy emoji was even chosen as the Oxford Dictionary word of the year [4]. Clearly, emojis have begun to play an important role in our lives, helping us communicate in a different, succinct, and fun way. The word emoji translates literally to 'picture character' in Japanese. Emojis emerged out of Japan at the end of the 20^{th} century, where they developed as the natural successors to emoticons.

Emoticons consist mainly of punctuation symbols, arranged in a particular way to resemble various human emotions. While emoticons are basically a simpler version of emojis, it is worth noting that the number of emoticons is very small, compared to the number of emojis. Since the vast majority of people choose to use emojis instead of emoticons, the rate of creation of new emoticons has somewhat stagnated, in the past decade or two. Emoji usage, on the other hand, is on the rise still, and new emojis are introduced to popular platforms such as iOS and Android, every few months.

For this project, we use a variety of features, such as, sentiment lexicons, word embeddings, and condensed TF/IDF vectors, to predict the emoji that would best fit any given tweet. This kind of a textual analysis has many applications in the real world, and some research has already been done on the subject. However, since emojis are relatively new on social media, research related to the sentiment analysis of emojis is still a nascent field. Nonetheless, predicting an emoji for a tweet can prove to be useful; considering it will add an extra layer of meaning for anyone who is trying to judge the sentiment behind the tweet. So, for this project, we plan to gain more insight into the meaning behind the 20 most frequently used emojis, and also attempt to find an appropriate emoji for each tweet that is capable of containing an emoji. We will rely on Natural Language Processing findings as well as Machine Learning models, including but not limited to, Recurrent Neural Networks and Decision Trees in order to gauge which emoji best suits a tweet. We also plan to use word embeddings derived from and several million tweets, trained using the skip-gram model. Furthermore, we will combine sentiment lexicons and links between emojis with features, such as, emoticons, and hashtags to generate a more accurate emoji prediction for a particular tweet.

Although research has been done to study the sentiment behind emojis, very little research

Figure 1: Emoji relative frequencies, ranked by most freq. to least



has been done on how emojis and words interact with each other, on a higher level. Our motivation for this project then, is to create an emoji classifier with the goal of helping the field of Natural Language Understanding, thereby contributing to the tasks pertaining to Natural Language Processing as well, such as improved text-based Data Mining, enhanced Sentiment Analysis, and even improved Machine Translation.

3 Related Work

Barbeiri et al. [2] examine the meanings of emojis, using vector space models. The dataset for this paper consisted of around 10 million tweets, and was used to create skip-gram neural embedding models consisting of both emojis, and words. This approach is interesting because it treats the emojis and the words in the same way, by mapping them to the same vector space. The paper also suggests clustering emojis to find similarities, and performing similarity matching.

Novak et al [5] built sentiment lexicons and created a map of the 751 most frequently used emojis. They manually annotated 1.6 million tweets, 4% of which contained emojis. Each emoji was mapped to one of three sentiments - positive, negative, or neutral.

Svetlana et al. [9] introduce us to the concept of sentiment analysis in detail, but on informal text messages. The two tasks attempted in the paper are, sentiment detection of informal text messages as a whole, and of individual terms or phrases in a message. The authors of the paper used the *Twitter Sentiment Analysis Dataset*, and proceeded to generate sentiment lexicons from this dataset, which they used it to train a supervised classifier. The sentiment lexicons used for this project included NRC Emotion Lexicon, Bing Lius Lexicon, and the MPQA Subjectivity Lexicon. The main takeaway from this paper, however, is the variety of sentiment lexicons used.

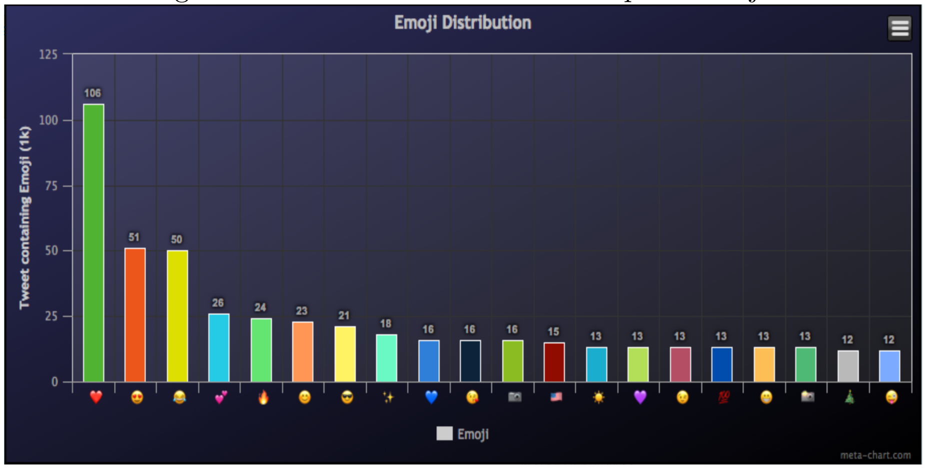
4 Data

Around the time we submitted our proposal, we did not have any access to data that relates tweets with emojis; therefore, we were planning on using the *Twitter Sentiment Analysis Dataset*. However, after being approved for SemEval task 2 (Emoji Prediction) we were granted access to multiple datasets that came with the task, including: a trial dataset of tweets and labels, a word-embedding created using a skipgram model [1], and a crawler to scrape twitter to produce a significantly bigger sample of training data. It is worth noting however, that the distribution of the labels is skewed in the dataset. One possible reason for this disparity might be the fact that people tend to use a few emojis many more times than other emojis.

4.1 The Trial Set

We started by using the trial dataset [3], which comprises 2 files: `us_trial.text` and `us_trial.labels`. The file `us_trial.text` consists of 50,000 tweets that have been scraped off twitter. These tweets have been filtered from emojis and links. Thus, any instances of

Figure 2: Distribution of 20 most frequent emojis



emojis or links have been removed to guarantee that the data set strictly contains pure text, i.e. only alpha-numeric characters. The tweets have also been wiped from any user mentions to preserve anonymity and enforce uniformity (i.e. @thebestusername would be wiped to @user). Furthermore, the second file `us_trial.labels` consists of 50,000 labels corresponding to the 50,000 tweets in the `us_trial.text` file; which means the tweet in line 69 in the `us_trial.text` file has the label in line 69 in the `us_trial.labels` file. It is important to note that these labels are numbered from 0 to 19, with 0 corresponding to the most frequent emoji (which is `_face_with_tears_of_joy_`) and 1 corresponds to the second most frequent emoji and so on. These numbered labels will then be mapped to the matching emoji (i.e. 0 will be mapped to `_face_with_tears_of_joy_`). This mapping of emojis to numbers makes it easier to perform the actual classification, by spending less time on the tricky implementation details that come with the unicode representations of emojis.

4.2 The Training Set

We were given a fully-functional crawler that scrapes twitter for tweets to use in to populate our training dataset [3]. We ran the crawler with six twitter app credentials, which include 4 main keys: a consumer key, a consumer key secret, an access token, and an access token secret. If any the previous credentials are missing, we will not have permission to scrape twitter for new data. Once everything was set up, we ran the crawler and were able to scrape around 500k tweets. We had intended to scrape around 2 million tweets, but we had found out that every time we scraped we were getting the same tweets; this was most likely due to the tweets being fetched by the same ids every time. None the less, of those 500k tweets, around 2% were not suitable for the JSON format we required; however, we still had around 490k eligible tweets that were completely compatible with our extractor. Subsequently, we used the `emoji_extractor_SemEval18.py` python file provided by SemEval, which takes in the JSON file of tweets, and returns some files: a file containing the text of all the tweets, a file containing all the labels corresponding to those tweets, and finally an extra file with unique id numbers corresponding to each tweets.

4.3 Word Embeddings

The SemEval website points its readers toward a pre-trained word-embedding [1] that consists of a file called `model_swm_300-6-10-low.w2v`, which is a collection of words (240397 words to be exact) and their respective vectors, with each vector being 300 dimensions in size, and with a window size of 6. These embeddings have been trained using a skip-gram model, on around 20 million tweets collected over the span of 4 months. As expected, word embeddings provided a significant boost in accuracy, compared to both, bag-of-words as well as TF/IDF features. We also tried using GLOVE embeddings instead of Word2Vec, but decided to stick to the latter, since the GLOVE embeddings did not provide an improvement in accuracy.

4.4 Data Split

We decided to go with an `60:20:20` split, i.e. training on 60% of the data, validating on 20%, and testing on the remaining 20% of the data. Even though we were provided with only training data, we were able to perform this split due to the abundance of tweets in the training data.

5 Approach

This section describes our approach to tackle this problem, in detail. In other words, we describe what steps we had to follow to create our dataset, the features we used, and how we used them for different models.

5.1 Preprocessing and Preparing the Dataset

Due to the procedure described in the previous section, we did not need to manually annotate tweets to populate our training dataset. Instead, we aggregated tweets onto the `us_train` dataset, as we mentioned before, by using the crawler that was provided to us by SemEval [3], along with the emoji-extractor, which provided us with data exactly like the `us_trial` dataset. Thus, all of the data annotation was performed automatically by the emoji-extractor. All we had to do is gather the tweets, using the crawler, and then feed them to the emoji-extractor, in order to receive the labelled data. We are guaranteed to have the correct annotations, since the labels (emojis) are extracted *from* the tweets, and all the tweets only have a single emoji themselves. Therefore, annotation is taken care of. Once through with the preprocessing, we moved on to feature engineering, as described in the following section.

5.2 Feature Engineering

In this section, we describe the numerous features we used, and the different combinations we experimented with for each model. In almost all cases, combining all the features resulted in the highest score for a given model. The features were first converted to matrices of the correct dimensions, and then concatenated to form one big final matrix.

5.2.1 TF/IDF

This was one of the first features we used for the baseline model (more on that in the models section). We had initially implemented the model using bag-of-word features, but decided to replace it with TF/IDF features simply because TF/IDF features are more informative than simple word counts.

Next, we decided to perform Truncated Singular Value Decomposition on the TF/IDF matrix to significantly reduce the dimensions from around $300,000$ to just 300. We decided to pursue Truncated SVD because it greatly reduces computation time by compressing the matrix. Upon trying different values to truncate the features, ranging from 250 to 400, 300 seemed to be the sweet spot.

5.2.2 Word Embeddings

Once we had a basic model running using basic features, we decided to move on to more complex features, such as pre-trained word embeddings. Word embeddings provide a mapping of words into the vector space, where each word is represented by its respective vector. Since the embeddings provided by the SemEval task were trained using the SkipGram model, the resulting vectors represent the context a given word is found in.
For our neural network models, we used the word embeddings themselves, and fed them directly to the embedding layer. For all other models, we calculated an *average embedding* for each tweet, i.e. for a given tweet, took the average of all the word embeddings. More concretely, for a given tweet T consisting of words $W = w_1, w_2, ..., w_n$, the average vector V can be calculated as follows:

$$V = \frac{w_1, w_2, ..., w_n}{n} = \frac{W}{n} \tag{1}$$

This method helps compress the information in the tweet, and proves to be computationally efficient at the same time.

5.2.3 Sentiment Lexicons

Sentiment Lexicons give insight into the evaluative nature or a given set of words. For example, a tweet can be positive, negative, or neutral, or it can contain text that signifies emotions such as anger or joy. We decided to use two sentiment lexicons, namely, the NRC Emotion Lexicon, and the NRC Hashtag Emotion Lexicon. The former is a list of words and their correlation with eight emotions (anger, fear, anticipation, trust, surprise, sadness, joy, and disgust) along with positive and negative sentiments; this list was annotated manually based on Amazon' Mechanical Turk. The latter is also a list of words and hashtags with the same associated emotions and sentiments; this list has been automatically populated from words with emotion word hashtags. To improve our classifier, we concatenated the 10 columns from both of these lexicons to our existing 700-column matrix making it a 720-column matrix, thus, adding an extra layer of emotion and sentiment analysis amongst tweets.

5.3 Models

In this section, we present the different models we implemented, ranging from linear classifiers to neural networks, along with different combinations of features applied to each of them. The major difference between LSTM based models and the others is that the B-LSTM takes individual words form the tweet into account, thus providing more context, and more information overall, whereas the other models simply take the average vector into account.

5.3.1 Baseline Model

We began with a simple baseline model, which implemented TF/IDF features, and ran SciKit-Learn's LogisticRegressionClassifier, with default parameters. The feature matrix consisted of TF/IDF features, chosen instead of simple word counts, as they provide more information. The TF/IDF matrix was retrieved using a vectorizer provided by SciKit-Lern. The classifier was trained with a one-vs-all strategy. It is also worth noting that the baseline model was trained on $40,000$ tweets, instead of $400,000$, because the full dataset was not made available to us at the time.

5.3.2 Logistic Regression Classifier

We implemented several stages of this model for the project, each building on top of, and therefore adding to the previous one. We began with the simple baseline model described above, and implemented the *average embedding* feature matrix, and concatenated it to the already defined TF/IDF matrix. Lastly, we added features built from the two sentiment lexicons described above.

The model works, based on the Sigmoid function:

$$\begin{aligned} h(x) &= g(\theta^T x) \\ &= \frac{1}{1 + e^{-\theta^T x}} \end{aligned} \tag{2}$$

where θ is the learned weights matrix, and x is a particular sample. We chose the decision boundary to be 0.5 (default). This function returns a value strictly between 0 and 1, thus classifying an example as part one class if it returns a value above the decision boundary, and part of the other class otherwise. Finally, we used a one-vs-all strategy to classify the tweets.

5.3.3 Decision Trees

The second model we tried was a SciKit-Learn's DecisionTreeClassifier. Once again, we used the same features as we did for the Logistic Regression model, but applied Singular Value Decomposition to vastly decrease the dimensions of the feature matrix. A Decision Tree Classifier essentially breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. This results in leaf nodes, which are contain classified samples of the data, and decision nodes, which eventually help classify new data samples. We chose the model because of the classifier's advantages over other models, including its effectiveness when it comes to selecting the most discriminating features, and its computational speed compared to other classifiers such as Logistic Regression.

5.3.4 B-LSTM

Given the proven effectiveness of neural networks, in addition to the work already done by Barbeiri et al. [2], we decided to try and implement a simple Bi-directional Long Distance Short Term Memory Network. The B-LSTM can be formalized as follows:

$$v = \max(0, \theta[f : b] + u) \tag{3}$$

where θ is the learned weights matrix, f is the forward encoding of the input, b is the backward encoding of the input, and u is the bias term. This vector is then used to compute the probability distribution of the emojis, given the tweet, using the softmax function:

$$P(x|v) = \frac{e^{g_x v + u_x}}{\sum_{x' \in X} e^{g_{x'} v + u_{x'}}} \tag{4}$$

where $g_{x'}$ is a vector representing the output embedding of the emoji x, and $u_{x'}$ the bias vector, and X is the set of possible emojis. The objective function being minimized is:

$$J(\theta) = -\log(P(x|v)) \tag{5}$$

We pass in word embeddings as input to the embedding layer of the network, which then passes its output onto two bidirectional layers, and finally to a dense activation layer, using softmax as the activation function.

6 Results

In this section, we aim two accomplish two goals, namely, give the reader an idea of why our best performing model performs the way it does, and also provide the reader with detailed analysis of the three major models we tried.

6.1 Method

In our experiment, we compare different models with various combinations of features, using different metrics such as overall model accuracy, precision, recall, and the F1 measure. We decided to divide the data in a $60 - 20 - 20$ split, for the training, validation, and test sets respectively. We trained and evaluated each model on the same dataset. Figure 3 gives a high level view of average model performance over the full test set.

6.2 Some Numbers

Our baseline model had an accuracy of 30.4%. We also calculated Precision, Recall, and F1 scores for different number of tweets from the test set, as can be seen in Table 1.

Table 1: Baseline Values for Precision, Recall, and F1 Measures

	P	R	F1
50	0.12	0.16	0.18
100	0.20	0.18	0.24
150	0.22	0.15	0.22
80000	0.19	0.15	0.22

The Logistic Regression model had an accuracy of 34.32%. Table 2 displays Precision, Recall, and F1 scores for different number of tweets from the test set. This is a big improvement from the previous model we trained, on $50,000$ samples. This makes sense, since in the final model we implemented all the features described above, and also trained on a much bigger training set.

Table 2: Logistic Regression Values for Precision, Recall, and F1 Measures

	P	R	F1
50	0.31	0.34	0.32
100	0.31	0.27	0.29
150	0.26	0.24	0.25
80000	0.33	0.22	0.26

The Decision Tree Classifier was the worst performing classifier among all the classifiers we tried. It achieved a model accuracy of a mere 21.9%. Our reasoning for such a low score is the fact that we implemented too many features for the model, thus leading it to overfit on the training data. Also, the training dataset was heavily unbalanced, with a fifth of the labels being the heart emoji. This must have led to the model developing a strong bias for the heart emoji class since it dominated the dataset. Once again, Table 3 shows the precision, recall, and F1 values we got for this model.

Table 3: Decision Trees Values for Precision, Recall, and F1 Measures

	P	R	F1
50	0.13	0.12	0.12
100	0.12	0.08	0.10
150	0.11	0.09	0.10
80000	0.12	0.12	0.12

Finally, the B-LSTM model was our bet performing model, achieving a model accuracy of 36.66%. We believe that this model performed better than the others because it took actual word embeddings as the input, rather than the average vector for the entire tweet, thus retaining more information. Additionally, Sentiment Lexicons played an important role in boosting the accuracy for this model, increasing it by nearly 4.5%. Table 4 depicts the precision, recall, and F1 scores for this model.

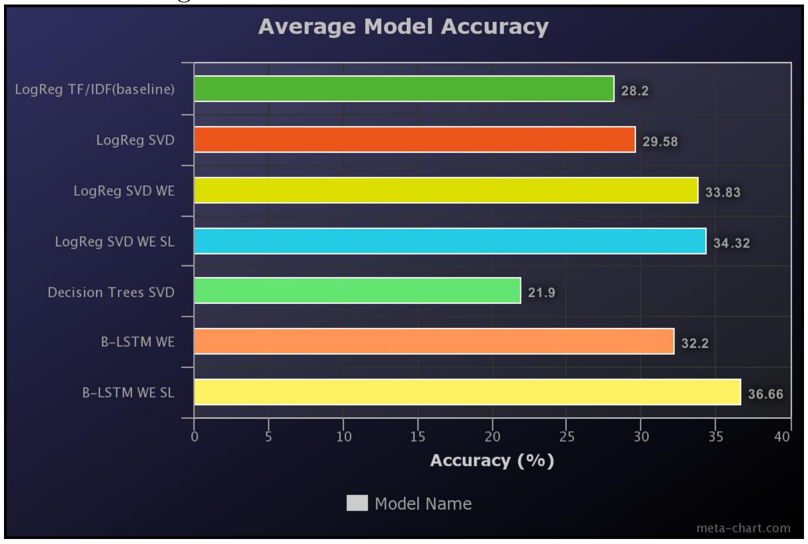
Table 4: B-LSTM Values for Precision, Recall, and F1 Measures

	P	R	F1
50	0.29	0.15	0.20
100	0.39	0.15	0.22
150	0.44	0.17	0.24
80000	0.49	0.12	0.20

6.3 Final Results

After experimenting with multiple models, we found B-LSTM to be the most precise model as shown in Table 5 below. However, its low recall score indicates that the model was

Figure 3: Accuracies of different models



not confident enough in its predictions. Logistic Regression had the best F1 score, which indicates both, good precision as well as good recall. As an aside, we note that the scores for each model are lower than the average scores for a typical classification problem. Although there are several reasons that can be put forward as explanations for this disparity, one that stands out is the fact that this is a 20-class classification problem, with a heavily skewed dataset, as compared to a typical classification problem, which involves 2 or 3 classes and a relatively balanced dataset.

Table 5: Precision, Recall, and F1 Measures for all used models

	P	R	F1	Accuracy
Baseline	0.19	0.15	0.22	28.2
LogReg	0.33	**0.22**	**0.26**	34.32
DT	0.12	0.12	0.12	21.90
B-LSTM	**0.49**	0.12	0.20	**36.66**

7 Discussion and Future Work

In this day and age, emojis have become a part of our daily lives, especially on social media. However, little is known about their use, sentiment, and semantics, because of how new they are. In this paper, we provided several architectures modelling the relationships between words and emojis. We also described our system that, given a tweet, predicts the most probable emoji to go along with it. We showed that our B-LSTM model performs significantly better than the other models, suggesting that there exists a strong relationship between the sequence of words, and its corresponding emoji.

As for future work, we plan on implementing a Convolutional Neural Network to achieve overall better precision and recall scores. We also plan to incorporate the positions of emojis in the tweets, since neighbouring words can provide important clues for predicting emojis. Finally, we intend to spend some time cleaning up the dataset, and making sure that it is not skewed, since a skewed dataset can significantly hinder a model's accuracy, as we saw in the case of Decision Trees.

References

[1] Francesco Barbieri, German Kruszewski, Francesco Ronzano, and Horacio Saggion. How cosmopolitan are emojis?: Exploring emojis usage and meaning over different languages with distributional semantics. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 531–535. ACM, 2016.

[2] Francesco Barbieri, Miguel Ballesteros, and Horacio Saggion. Are emojis predictable? *arXiv preprint arXiv:1702.07285*, 2017.

[3] Luda Zhao and Connie Zeng. Using neural networks to predict emoji usage from twitter data.

[4] Oxford's 2015 word of the year is this emoji, http://time.com/4114886/oxford-word-of-the-year-2015-emoji.

[5] Petra Kralj Novak, Jasmina Smailović, Borut Sluban, and Igor Mozetič. Sentiment of emojis. *PloS one*, 10(12):e0144296, 2015.

[6] Sean Dolinar. The most popular emojis on twitter, http://stats.seandolinar.com/popular-emoji-on-twitter, Dec 2014.

[7] scikit-learn documentation. http://scikit-learn.org/stable/documentation.html.

[8] SemEval task 2. https://competitions.codalab.org/competitions/17344.

[9] Svetlana Kiritchenko, Xiaodan Zhu, and Saif M Mohammad. Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, 50:723–762, 2014.