What's your emoji

CS 585, UMass Amherst, Fall 2017

Ronit Arora, Shubham Mehta, Vedant Puri

rarora@umass.edu, shubhammehta@umass.edu, vedantpuri@umass.edu

Abstract

In today's world where short messages and tweets are at the core of communication, emojis have become major forms of expression of ideas and emotions. Emojis penetrate language barriers and allow people to express a lot with very few words. With the increasing use of emojis in our daily life, sometimes we lose context of text and aren't really sure about which emoji to use based on the text. Our project aims to suggest emojis based on the given text by analyzing the sentiment of the given text and predicting relevant emojis for it.

Introduction

Communication is key to everyday life today, and within recent decades, the modes by which a person is able to express himself have only exponentiated. To elucidate, we have evolved from dry letters to electronic mails and finally to social media today. A single person's reaction has the ability to send a shockwave throughout the world within seconds. Moreover, individuals have the option of expressing their reactions in almost every way imaginable. Stickers, GIFs, styled words and emoticons are all methods by which a person can express himself. This is why we have taken on the task of understanding these forms of communication. Specifically, we have decided to understand the sentiment of tweets by predicting a suitable emoji for it. Our emoji prediction is restricted to the twenty emojis listed below:

0 ❤ red heart
1 😍 smiling face with heart eyes
2 😂 face with tears of joy
3 💕 two hearts
4 😊 smiling face with smiling_eyes
5 😘 face blowing a kiss
6 💪 flexed biceps

```
7  😉  winking face
8  👌  OK hand
9  🇪🇸  Spain
10 😎  smiling face with sunglasses
11 💙  blue heart
12 💜  purple heart
13 😜  winking face with tongue
14 💞  revolving hearts
15 ✨  sparkles
16 🎶  musical notes
17 💘  heart with arrow
18 😁  beaming face with smiling eyes
19 🔝  TOP arrow
```

This problem is particularly interesting for numerous reasons. First off, out of the more-recent advancements in pictorial-forms of expression, emoticons are the most common. They are cross-cultural and used daily by users of social media apps. Likewise, Twitter is one of the most popular sources of social media today. The content that is available in tweets is diversified and realistic. Moreover, the content on Twitter is always up-to-date as the platform has millions of users that are constantly reacting to their Twitter feeds. Furthermore, and rather the most intriguing part of this problem, is that sentiment is highly complex. The vast amount of subtleties associated with emotions is mind-boggling and representing tweets in a form that is understandable to computers is a challenging task. If computers are able to understand the intricacies associated with the sentiment of expressions that are as complicated as tweets, it is indeed a solid advancement. It is the key to unlocking an oasis behind which smarter software systems lie. In fact, the extent to which applications could interact with humans if the sentiment of humans could be accurately captured is unimaginable. From text-prediction in mobile systems to automated agents, it would be a new dimension of human-computer interaction.

To tackle this problem, we have employed various models. Our experimentation started off with a Naive Bayes Classifier, after which we moved to Decision Trees and finally Long Short-Term Memory Networks. Our best results have been with Bidirectional-LSTMs using pre-trained word2vec embeddings. The rest of this paper dwells into our adventure with this problem and the results that we have observed.

Related Work

Before tackling the task of emoji prediction, we decided to consult previous works in an effort to gain a deeper understanding of the problem, and also to learn about the types of models that would be suitable for this task. This section explains the various research papers that we have examined regarding previous works.

Francesco Barbieri et al.[1] use a huge dataset of tweets and firstly extract the ones which include one and only one of the top 20 most frequent emojis, then remove the emoji from the tokens and use it for both training and testing.The paper uses Bidirectional Long Short-term Memory Networks which is a model based on Recurrent Neural Networks to predict an emoji based on a given text. Given the fact that considering the sequence of words is essential to bolster the meaning of the text, the B-LSTM always outperforms the baseline representations such as BOW (Bag-of-words) representation as the B-LSTM takes into consideration the context of a message. One of the shortcomings of this paper was that it only predicted a single emoji for a given tweet and we feel that there could be various possible emoticons for a given tweet.

In Paper [2], Luda Zhao and Connie Zeng use Long-Term Short Memory Recurrent Neural Network and Convolutional Neural Network models which seem to be effective in other closely related tasks like sentiment classification and language modelling. The tweets were parsed and the latest occurring emoji was used as the label for that tweet. This approach produced decent accuracy with pre-filtered data but did not work with noisy data. Also, in situations where words were replaced with emojis, the approach used in the paper was not quite effective. Our project first extracts relevant features from the given text and uses a combination of sentiment analysis and neural networks to predict the emoji. Also, our project will propose another model using word-by-word analysis for the emoticons.

In Paper [3], Balabantaray, Mohammad and Sharma use opinion mining and text classification techniques to assign emotions to tweets. Before performing classification, some of the pre-processing steps that were involved include URL removal, removal of foreign-language tweets and timestamp removal. Likewise, the researchers made use of seed words. Particularly, these seed words were the six emotion classification given by Ekman. For classification, a multi-class SVM model was used and for extracting features, seven tags (Six Ekman emotions and one neutral tag) were used, along with stemming, dependency parsing between pronouns and verbs and POS tagging. They used Cohen's Kappa to measure the pairwise agreement of the labels. Overall, the paper claims that accuracy of this approach was 72.34 %, whereas similar works had only achieved 65.5%. For our project, we can

make great improvements to this approach by using more features and by developing a finer-grained analysis. The features and seed words in this study are very limited. For instance, a great emphasis is given on Ekman's six emotion categories in this approach, but in reality we know that the dependencies among emotions is quite vast and intricate.

Data

Tweets

In our experiment we have decided to use the SemEval 2018 (Task 2) dataset. This dataset is particularly suitable for our purposes because it consists of labelled tweets. This dataset has also helped us expand the language domain of our project because it contains both English and Spanish tweets, of which there are 50,000 and 10,000 respectively. Each tweet is classified into one of twenty emojis, which are denoted by the numbers 0 -19 (pictorial representation is given in the introduction section of the report). The mapping from numbers to its corresponding emoji is given in the file *us_mapping.txt*.

The essence of this dataset is distributed among 4 files. The first two are *us_trial.text* and *us_trial.labels*. The file *us_trial.text* contains 50,000 tweets and *us_trial.labels* contains 50,000 labels. The tweet-to-label correspondence is at a line by line level. To elucidate, line 1 of *us_trial.text* has a tweet whose respective emoji label is given by line 1 of *us_trial.labels*. The emoji is denoted by a number, which as explained above, can be decoded using *us_mapping.txt*. Similarly, *es_trial.text* and *es_trial.labels* have the same line-to-line correspondence for 10,000 tweets. The number-to-emoji correspondence is given in *es_mapping.txt.*

Pretrained Word Embeddings

In our experiments, we have made use of pretrained word2vec embeddings obtained from the github repository for SemEval 2018 (Task 2). The data consists of two files, *model_swm_300-6-10-low.w2v* and *model_swm_300-6-10-nolow.w2v*. The "low.w2v" file contains word embeddings for lowercase words, whereas the "nolow.w2v" accounts for case-sensitivity. These embeddings are in the form of skipgram vectors that have a dimension of 300 and have been trained on 20 million tweets. For our purpose, we have made use of *model_swm_300-6-10-low.w2v* because our tokenizing procedure converts all tokens to lowercase. Additionally, to load the

embeddings file, we have used Gensim's KeyedVectors library, which loads the embeddings in the form of a dictionary for easy use.

GloVe Embeddings

We have also experimented with GloVe embeddings (Global Vector for Word Representation), made available by the Stanford NLP Group. These embeddings were trained using an unsupervised learning algorithm called GloVe, as specified by the title. Although there are various embeddings available, we have used *glove.840B.300d.txt,* which are cased embeddings that have been trained on 840 billion tokens and a vocabulary size of 2.2 million. The vector representation of these embeddings is also of 300 dimensions, same as the SemEval embeddings. Likewise, we used Gensim's Glove2Word2Vec library to convert these embeddings into Word2Vec format, after which we again used Gensim's KeyedVectors library to obtain a dictionary representation of the embeddings.

Method

Baseline Model

As a baseline model, we deployed Naive Bayes Classification. Our model assumed conditional independence among words. Although by nature, tweets have an enormous amount of intricacy and subtlety, Naive Bayes was an effective means of determining a baseline accuracy.

Naive Bayes Classification

1. Bag of Words:-

Our Naive Bayes Classifier consists of 20 classes, which were obtained directly from *us_mapping.txt* and *es_mapping.txt*. For prediction, we have used a bag-of-words model. Specifically, our Naive Bayes Classifier consists of the following data structures: *total_tweets_per_class, word_counts_per_class, words_per_class and vocabulary*. The descriptions of each are as follows:

- *Total_tweets_per_class :-* As mentioned, our Naive Bayes Classifier consists of 20 classes. This data structure is a dictionary that stores the count of tweets per class. In our input file, each line is a separate tweet.

So, this data structure can also be thought of as the number of lines that pertain to each emoji label.

- *Word_counts_per_class:-* This is the number of words that pertain to each class. In other words, let a certain emoji class have *x* number of tweets with its label. If each of those *x* tweets contain *n* words, then the total number of words that pertain to that emoji class is *xn*. Thus, this data structure is a dictionary with key as the emoji class and value as the number of words pertaining to that particular emoji class.

- *Words_per_class:-* This data structure is a dictionary of dictionaries. The key is an emoji class, and its value is a count of each of the individual words that pertain to that emoji class. In other words, it is a more granular version of *word_counts_per_class*. For instance, if a certain emoji has a total word count of 100, *words_per_class* will give a breakdown of the individual counts of words that make up those 100.

- *Vocabulary :-* As the name suggests, it is a set of all the distinct words we have seen in the data set.

2. Tokenization:-

Now that we have understood the specific counts that are being kept track of, let us look at the way the dataset is being tokenized. To minimize the amount of noise and to ensure that our analysis incorporates words that actually have meaning, we have applied the following refinements on our dataset:

- Removal of stopwords: Stopwords are irrelevant to our analysis because they do not contribute to emotion.

- Elimination of words beginning with @ and # :- Our analysis pertains to the text of a tweet

- Consideration of only words consisting of 2 or more alphabetic characters:- We are considered only with text that has meaning and shows emotion. Words that have a lot of punctuation and numbers are not accurate signs of human sentiment.

- Lowercasing of words :- In order to have a uniform representation of our tweets, we converted all of the words to lowercase. This is particularly helpful for getting rid of case-sensitivity complexities when working with embeddings and also when creating your own tweet representation, such as bag of words. For example, you don't need to have separate representations for the word "hello" and "Hello."

3. Classification:-

To classify a tweet, we take its bag of words representation and take the argmax over all the probabilities for each label. We also make use of pseudocount smoothing because we are using logarithms to compute the required probabilities. The pseudocode for our method is shown below:

1. *Probability of word given label and pseudocount ← (Count of word for the given class + alpha) / (Total number of words for the class + alpha * size of vocabulary)*

2. *Log_likelihood ← for each word in the bag of words representation of the tweet, we take the log of the probability yielded by step 1. We then sum over all these probabilities*

3. *Log_prior ← log (total number of tweets for the class / number of tweets that are being used for training)*

4. *Unnormalized _log_posterior ← Sum of Step 2 and Step 3*

5. *CLASSIFICATION ← Compute the bag of words representation for the given tweet and for each label (emoji), compute the unnormalized_log_posterior as defined above. Our predicted emoji is the emoji with the highest unnormalized_log_posterior.*

4. RESULTS:-

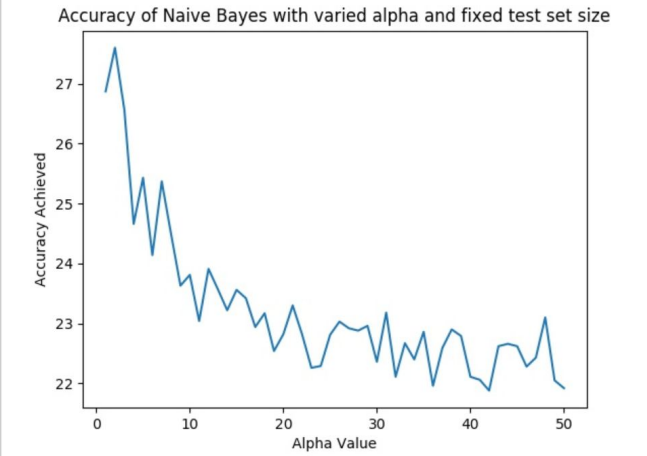
Using the Naive Bayes Classification described above, we were able to achieve an accuracy of 26.00 % for English tweets in 18.55 seconds and an accuracy of 23.87 % for Spanish tweets in 3.65 seconds. These results were obtained on a Stand-alone Ubuntu-server-12.04 running in VMWare on Windows 10 (64-bit) with 1GB of memory.

As mentioned previously, our dataset contains 50,000 English tweets and 10,000 Spanish tweets.

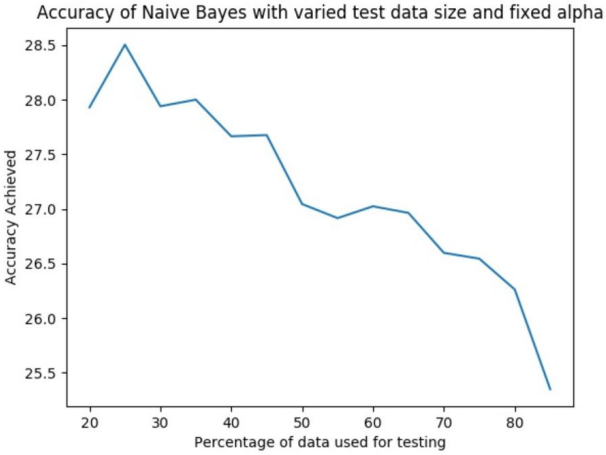
Graphical Results

Additionally, the value of alpha used for pseudocount smoothing and the percentage of our dataset that was used for testing and training are important factors in our experiments. The following graphs portray the influence of these factors on our results:

The following graphs pertain to our results for English tweets



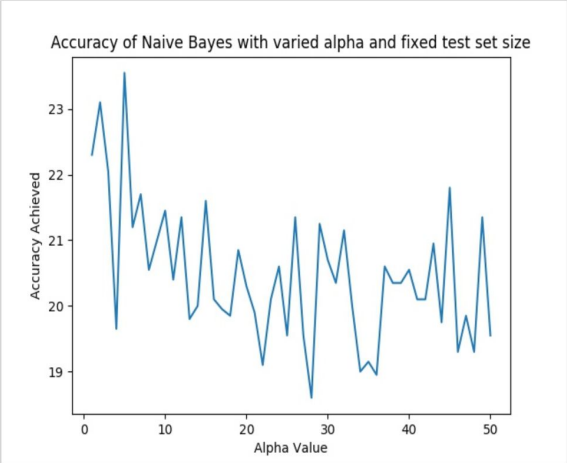
The value of alpha is inversely proportional to the accuracy.
Test set size = 20%



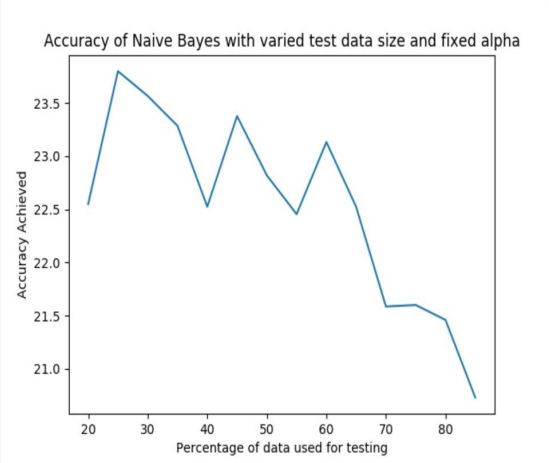
The data consists of 50,000 tweets, of which the percentage of data used for testing is given on the x-axis.
Alpha = 1.0

As the graphs depict, low values of alpha and testing on 20 - 30 % of the data is where our experiment has achieved the highest accuracy

The following graphs pertain to our results for Spanish tweets



In general, lower alpha values have yielded higher accuracy, but the relationship is not linear
Test set size = 20%



The data consists of 10,000 tweets, of which the percentage of data used for testing is given on the x-axis.
Alpha = 1.0

For Spanish tweets as well, low alpha values and testing on 20 - 30% of the data is where the highest accuracy was achieved.

Decision Tree Classifier

Implementation

Moving forward, we decided to explore models where we wouldn't have to assume conditional independence among the words in a tweet since clearly, the sentiment that pertains to a tweet can(and usually does) have dependencies on the context of the tweet. The next model that we performed our emoji prediction task with was Decision Tree Classification.

We implemented our Decision Tree using Scikit-Learn's DecisionTreeClassifier library. In particular, we used the fit and predict function for training and testing. Our method of tokenization is the same as what we did for Naive Bayes (e.g. Stopword removal, # and @ removal, extraction of words with 2 or more alphabetic characters and

lowercasing of words). Likewise, the representation for a tweet is again bag of words, which we constructed by first creating a vector of 0s for each of the 50,000 tweets. Each of these vectors is as large as the size of our vocabulary, meaning their dimension is of 58,581. Each index of this vector corresponds to a word in our vocabulary. To populate the counts, we iterate through our tweets dataset, and update the counts at the index of the vector that corresponds to the word that is present in the tweet. Thus, at the end of this process, we have a collection of 50,000 vectors, each with information about the counts of words in a tweet.

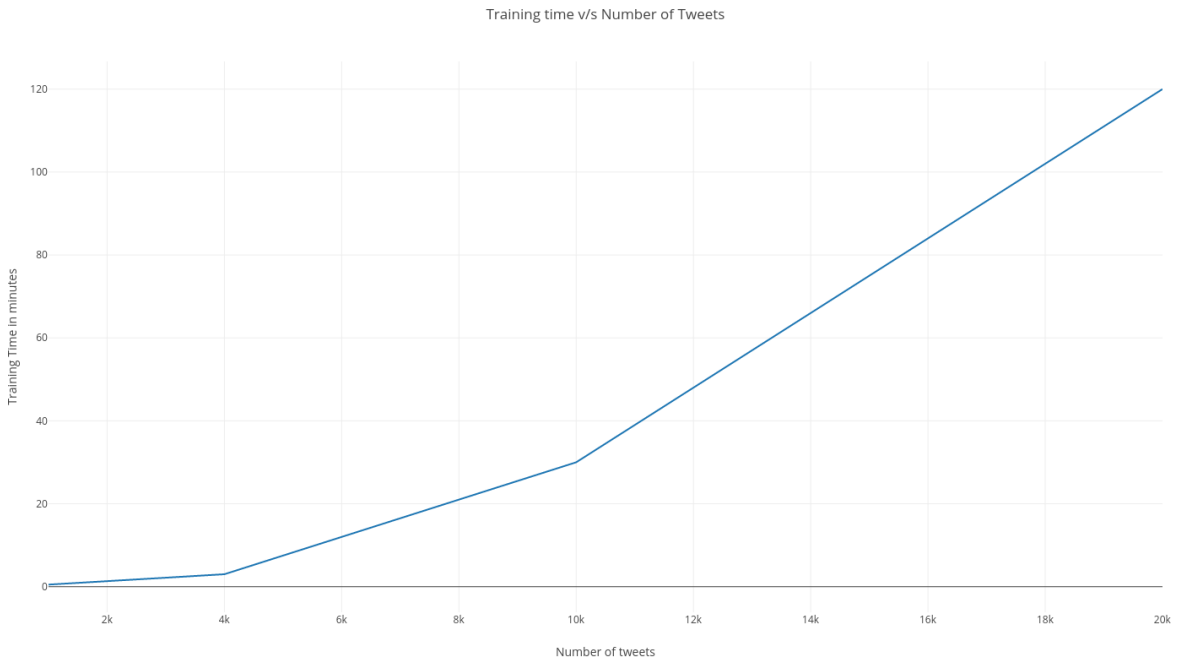
Continuing on, we trained our decision tree classifier on 800 tweets by calling the fit function, provided by SciKit-Learn. The maximum depth to which we trained our tree was the SciKit-Learn default, that is until all leaves are pure or until all leaves contains less than a minimum number of samples (1 by default). We then tested this model on 200 tweets by calling the predict function for every tweet in the test set and obtained an accuracy score by dividing the number of correct predictions with the total number of tweets in the test set.

Outcomes

Although we were quite hopeful that this new model would outperform our Naive Bayes Classification, it turned out that with Decision Trees, the maximum accuracy achieved was less than that of Naive Bayes. Decision Tree Classification yielded a maximum of 23.3 % accuracy, and also the high dimensionality of our bag-of-words vector representation (58,581) prevented us from being able to utilize the entire dataset for training and testing. Nonetheless, the outcomes of this experiment proved to be an insightful experience. We believe that overfitting is the root cause for decision tree's inability to predict accurately. The decision boundary that results for the task of emoticon prediction can be highly complex due to the immense amount of complexities that are associated with the task. The sheer amount of variation that each tweet induces is surely to result in very tight and convoluted decision boundaries. Therefore, the decision boundaries that a decision tree learns will be highly tuned to the training data, and will not be able to accurately classify new, unseen data as the boundaries have been firmly adjusted to the training set. In fact, in our experimentation, the decision tree was able to classify the training set with 99.5% percent accuracy. The high accuracy percentage, and the drastic drop in accuracy on the test set, is highly indicative of overfitting.

Additionally, the amount of time that our decision tree was taking to finish training was a clear sign of the fact that our tweet representation had to be changed. 58,000

dimensions (approximately), is too large and sparse. A more concise and elegant representation had to be thought of.



The graph portrays the exponential increase in train time with respect to the size of the training data

Long Short-Term Memory Networks (LSTM, BLSTM)

In this section, we will be explaining our neural network based approach to our emoji prediction problem. We decided to experiment with this model because as mentioned, the decision tree model was highly prone to overfitting and the Naive Bayes model assumed conditional independence among words, which is certainly not the case for tweets. Additionally, we change our tweet representation from bag-of-words to word embeddings in an effort to reduce dimensions and sparsity. Our implementations of both an LSTM and BLSTM have been done using Keras's neural network library with Tensorflow as the backend.

Embeddings Layer

Let us first understand the representation of each tweet and how it is being processed by the neural network model. To start off, the features of each tweet will

be the word embedding vector for each of its words. For instance, if a certain tweet consists of the words “It is sunny outside #summer.” The word embedding vectors for each of the 5 tokens that are in this tweet will be shipped off as features for that tweet. In other words, the tweet has now been converted to a list of vectors, where each vector corresponds to the word embeddings vector of the tokens in that tweet. However, instead of feeding the LSTM a sequence of vectors, we have made use of the Keras embeddings layer. This layer allows us to supply the embeddings for the entire vocabulary of the training corpus. To do this, we have constructed a weight matrix and encoded the tokens of the tweets to correspond to entries of the weight matrix. To elucidate, the weight matrix is a matrix whose dimensions are *(size of vocabulary of dataset x dimension of the word embeddings vector).* Hence, at every index from 0 to |Vocabulary|, there is a word embeddings vector. To ensure that every token in a tweet maps to the correct index in this weight matrix, and therefore receives the correct embedding vector, we have used Keras’s Tokenizer library to transform the tweets into a sequence of numbers such that every number corresponds to the index at which its corresponding token’s embedding vector lies in the weight matrix. For example, the tweet “eat good food” might translate into [1, 5, 7] where index 1 in the weight matrix contains the embedding for “eat”, index 5 contains the embedding for “good” and index 7 contains the embedding for “food”. Thus, our input data is a now a sequence of numbers and the weight matrix is the data structure that will give meaning to these sequence of numbers. The connection between the two will happen in the Embedding Layer of the LSTM. We initialize the Embedding Layer by feeding it the specified weight matrix, and upon receiving an input sequence, the Embedding Layer will perform the translation.

Word2Vec and GloVe

Next, we will discuss the actual embeddings that were used to populate the weight matrix. In the Data section of this report, we have already mentioned details about the pre-trained word embedding files and how they were transformed into a dictionary data structure where keys are words and values are the embedding vectors corresponding to the word. Hence, while populating the weight matrix, we traverse every word in the vocabulary and lookup its vector embedding in the word2vec dictionary. These vectors are of 300 dimensions (both for GloVe and SemEval’s Word2Vec), and for words that don’t have an embedding available for them, we use a 300 dimension vector that consists of zeros as placeholders to ensure that the dimensions of our weight matrix are proper.

In our experiments, we have found that using the pretrained Word2Vec embeddings from SemEval has yielded higher accuracy results as compared to GloVe embeddings.

We suspect that this is because the embeddings provided by SemEval have been trained on Twitter data, and our dataset consists only of tweets, so the context given by those embeddings fits to be a closer match. For example, tweets consist of different kinds of symbols and informal English, which are likely to have a more proper representation in embeddings that have been trained on tweets itself rather than some other source. We have also experimented with using both pre-trained embeddings at once, in which if an embedding for a word is available in SemEval's Word2Vec, we plug that into the weight matrix, and otherwise we use a GloVe embedding if it is available for that word. Although this reduces the number of placeholders, that is vectors containing zeroes for words that don't have an available embedding, it simply creates more noise. The embedding values don't correspond as they are from different sources, and lower accuracy results. Our results suffered a drop of roughly 1.2% in accuracy when we pursued this approach.

Trainable Embedding Layer

Additionally, Keras also offers the option of allowing your Embedding layer to learn the embeddings for all of the words in the training set. This was an ideal solution to dealing with placeholders in our weight matrix because the Embedding layer could itself learn the embeddings for those words that did not have a representation in Word2Vec, whilst using and revamping the initial weights passed to it by our weight matrix. In fact, we achieved our highest accuracy using this combination of SemEval's Word2Vec embeddings for weight matrix initialization and a self-trainable embedding layer for placeholder improvisation.

Tokenizing Revisited

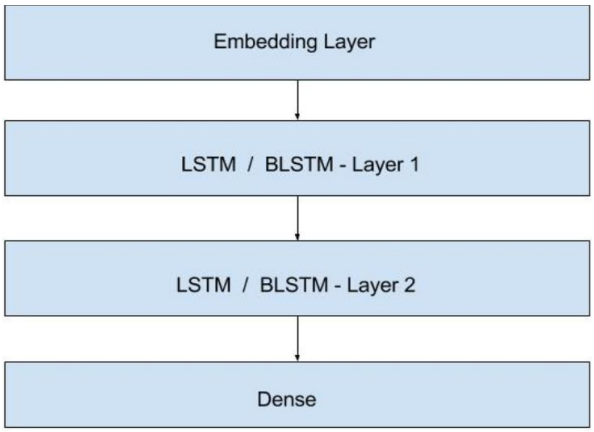
In our previous experiments using Naive Bayes Classification and Decision Trees, we used the same tokenization procedure that was given at the beginning of this report. However, here we will relax some of the constraints in our tokenization process. With SemEval's Word2Vec embeddings, it is not necessary to filter tokens beginning with hashtags since it was trained on Twitter data and therefore has an embedding available for such tokens. In fact, this is beneficial to our model because it can learn more information. Likewise, we no longer eliminate tokens consisting of less than 2 alphabetic characters. Instead, we just split on spaces (" ") and filter out punctuation. Removal of stopwords, words beginning with @ and lowercase tokens still remain. This revision allowed us to use more of the embeddings that are available in SemEval's Word2Vec and obtain a 1.5% boost in accuracy.

Padding

Recall that our dataset has been transformed into a sequence of numbers for this experiment. Tweets are of variable length, i.e. each sequence has a different size. Since the training and test set must be matrices of concise dimensions, we apply padding to each sequence to ensure that they are all of the same length. The pad length that we are using is the average length of a tweet sequence, which is 10 for our dataset. If a certain tweet's sequence length is greater than this value, it will get truncated and if it is less than this value, it will be padded with 0s. It does not make sense to use the maximum length of a tweet sequence as the pad length because it results in a large number of extraneous 0s, which is detrimental to the performance of our model due to the noise created. To illuminate, the maximum length of a tweet sequence in our dataset is 32, which is far from the average length of most tweets.

Layers

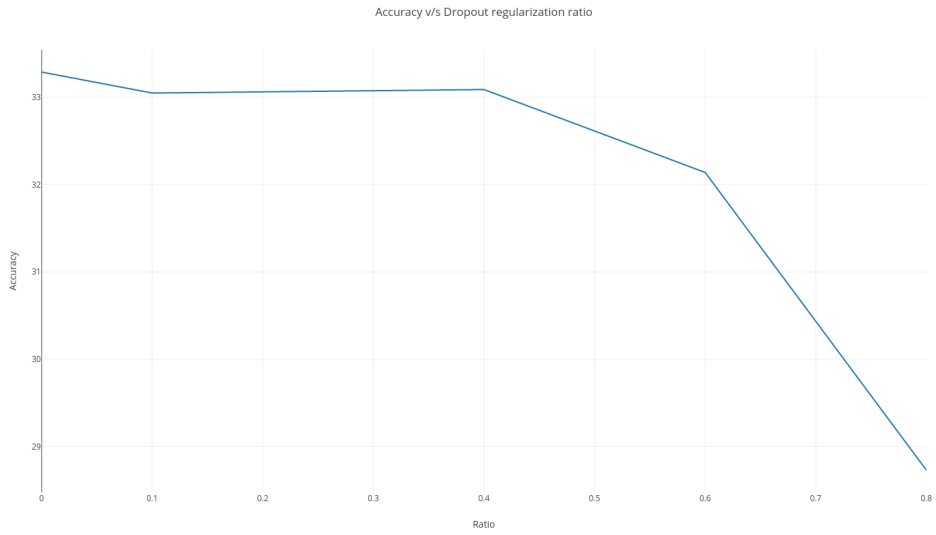
We have deployed both an LSTM and a BLSTM. The structure of our model is below:



Thus, it is a multi-layered neural network where we are returning the full sequence from the first layer to the second. Each layer consists of 128 cells. Likewise, the activation function that we have employed is softmax, along with categorical cross-entropy for the loss function. Since we are predicting one of twenty emojis, labels are given as bit vectors of 20 dimensions in which a label is denoted by a 1 at its respective index. Finally, we have applied dropout regularization to our model as well.

Outcome

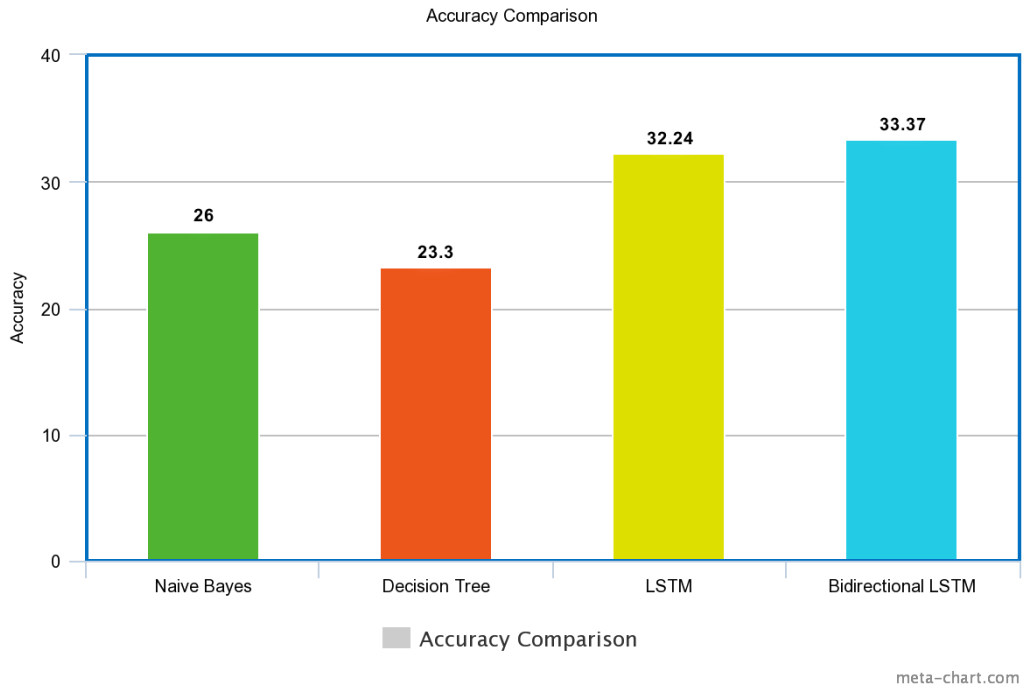
The BLSTM was able to outperform the LSTM by approximately 1.1% in terms of accuracy. The following graph displays the performance of the BLSTM with respect to different dropout values:



The graph portrays that high dropout values have a negative impact, but low values are able to provide a slight improvement in accuracy

Overall Results

In summation, the neural network approach stood superior to all other models. With a training time of less than 20 minutes on a Windows 10 (64-bit) machine with 8GB of memory, BLSTM yielded the highest accuracy of 33.37%. The bar graph highlights the results of the various models that we have deployed throughout our project:



As a recap, we started off with a Naive Bayes Classification as a baseline model and although it assumes conditional independence, it formed the basis by which we could judge other models. In fact, the sole reason by which we made the judgement that Decision Tree Classification was below expectations was because we had already achieved higher accuracy with a simpler and much faster Naive Bayes Classifier. Nonetheless, Decision Tree Classification provided insight about the complex decision boundaries of our emoji prediction task and pointed the urgent need to reduce the dimensionality and sparsity of the features that we were using to represent each tweet. This is what led us ultimately to a neural network model, where we tried both LSTM and BLSTM along with word embeddings to achieve our highest accuracies.

Discussion and Future Work

Certainly, the journey doesn't end here. Although we have explored the task of emoji prediction with multiple models and approaches, many implications remain. To elucidate, there is still a great amount to be done in terms of feature exploration. Currently, we have deployed bag of words and word embeddings. However, other features, such as sentiment lexicons and various other concise vectorized representations are all possibilities that we plan on looking into. Likewise, we also wish to explore more models. For example, Convolutional Neural Networks are promising for accurately determining sentiment, as per researchers. There are also

various other additions and variants of LSTMs that we wish to ponder upon. To elucidate, the addition of an Attention layer on a BLSTM is known to produce higher accuracies. Tree-LSTM is another LSTM variant that may help us in more effectively tackling this task. Another important factor is training our models for longer periods of time on faster machinery. In fact, with more computing power, we would be able to harness more data and churn effective models.

Thus, there is still much more to be done. The task of emoji prediction, or sentiment analysis, is indeed a very intricate task. As a matter of fact, even humans themselves have difficulty understanding the sentiment of other individuals. Regardless though, making computers comprehend sentiment is an exciting task. Indeed if it would be possible to reach 100 percent accuracy, the world of computing would be revolutionized.

References

[1] Barbieri, Francesco and Miguel Ballesteros. “Are Emojis Predictable?”(2017)

[2] Zhao, Luda and Connie Zeng.“Using Neural Networks to Predict Emoji Usage from Twitter Data.”(2017)

[3] Mohammad, Mudasir, et al. “Multi-Class Twitter Emotion Classification: A New Approach .” (2012)

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[5] Fvancesco/acmmm2016. Retrieved December 22, 2017, from https://github.com/fvancesco/acmmm2016