**Group Members (5 members):**

Alejandra Martinez .609 |Section 11087

Mingjie Song.1500 | Section 8395

Mimi Cai. 697 | Section 11087

Robert Wetzler.4 |Section 11087

Stuti Shah.1440 | Section 11086

The problem we decided to work on for this project involved the predicting of related emojis based on a given text block. The inspiration for this work stemmed from a competition of CodaLab where they cited the importance of emojis as visuals providing extra meaning to plain text in a world of popular text driven social media, such as Twitter.

For the environment of the problem we have user input alongside the test data. The user input is plain text and the related emoji is what is predicted to the inputs. This environment will be fully observable and single agent. Once the user inputs text, an emoji will be predicted. For another emoji prediction the user will need to resubmit a new text block, making the environment deterministic, static and discrete. Lastly, this environment is episodic where each block of text and its predicted emoji are independent from other text blocks and their predicted emoji. For this problem, we decided to use the Naïve Bayes algorithm and a Neural Network as the two machine learning algorithms we wanted to experiment with for the emoji prediction.

The Naïve Bayes algorithm was the first algorithm we implemented to predict the emoji used in a text. The implemented algorithm consisted of seven methods aside from the main method: getData, trainData, getListForLabel, classVocab, prediction, find\_prob, and test. getData converted urls containing the test data into a dataframe, trainData trained the model using the methods getListForLabel and classVocab. getListForLabel turned the dataframe and a given class number into a list of tweets corresponding to the given class, while classVocab helped to find the vocab words and corresponding counts for each class. prediction used the method find\_prob to create a list of predictions of the emoji used in tweets from the test data using the trained model, while find\_prob helped determine the class that a tweet most likely belonged to. Finally, test compared the predictions to the true labels and output the model’s accuracy.

This implementation had the labels and corresponding texts put into a dictionary to avoid having to write functions for each of the 19 individual labels. The algorithm at its core is still the Naïve Bayes algorithm, but the code has been modified to allow the code to work for any number of labels in the training data. The acquired data contained a mapping from emoji to number. In order to modify the code, these numbers became the keys in most of the dictionaries. The values in the dictionaries varied from the number of words in a class, to a dictionary of vocab words in each class with their counts, probabilities of each class, and so on.

This implementation of the Naïve Bayes algorithm produced an accuracy of 21.23% with an alpha value of 10. Lower values of alpha resulted in a lower overall accuracy, as an alpha value of 1 resulted in an accuracy of 17.25%. Larger values of alpha did improve the accuracy, but even alpha values as large as 1000 yielded an accuracy around 22%. The figures below show a potential reason why this occurred - as the alpha value increased, the only accurately predicted emojis was the heart. This is because, with a higher alpha value Naive Bayes began to only predict heart emojis. However, with a lower alpha, the model predicted a more diverse set of emojis at the cost of a lower overall accuracy (Appendix 1). This shift towards only predicting heart emojis occurred when using the Spanish data set as well.

In an attempt to improve the accuracy, about 100 of the most common words such as “the”, “I”, “to”, “a”, and other common words that likely wouldn’t impact the emoji were excluded when training and testing the data. However, this reduced the accuracy compared to when the common words were included. The accuracy with an alpha value of 1 was 16.59%, compared to an accuracy of 17.25% when the 100 most common words were included. When alpha was raised to 10, the accuracy when words were excluded was 20.89%, compared to 21.23% when the 100 most common words were included.

Even multiple combinations of excluded words were tested to try and improve the accuracy and didn't result in any combination where the accuracy improved. Thus, in regards to the Naïve Bayes algorithm, we determined that 21.23% was likely to be the highest accuracy we could reasonably achieve using the Naïve Bayes algorithm. Despite this shortcoming, we implemented an extension to the algorithm, where the model would predict the emoji based on user-input text using primarily the same methods discussed earlier.

The second algorithm used the convolutional neural network. The entire process can be divided into three steps, data preprocessing, word embedding, and the neural network model.

**Data preprocessing**

First we cleaned the data by removing punctuation, stop words and any words with a minimum occurrence below 2, and then we transferred each remaining tweet to a list of tokens. In order to feed the text data into the neural network model, we need to turn them into numerical values. So we used Keras Tokenizer class to map each unique token to an integer.

Another problem is that the input of the neural network needs to have the same length but the length of each sentence is different. We can solve the problem by Keras function pad\_sequence() so that each tweet will be padded with value 0 to the same length.

We also represented tweets’ label using one-hot encoding so that it matches the output shape of our neural network model.

**Word Embedding**

A word embedding is a way for representing words and documents using a dense vector representation. Compared to the traditional bag-of-word model where large sparse vectors were used to represent each word, the word embedding is a projection of the word into a continuous vector space and it can preserve the spatial information of each word. The size of the vector can be decided by ourselves and in our model, each unique token will be represented using a vector of size 100.

The process of learning the embedding representation for each token can be treated as a layer and integrated into the neural network model where the embedding is learned along with the model itself. In our project, we applied python Keras library to add an embedding layer on top of the convolutional layer (Appendix 2).

**Neural Network Model**

A convolutional neural network is a special type of a feed forward neural network and its main advantage is local feature detection ability. For example, given different tweets, it can learn distinctive features for each sentence and to match it to related emoji.

Above is the overall structure of the Neural Network Model. The input layer is an embedding layer and the shape is 1\*28, where 28 is the maximum length among all the tweets. The output of the embedding layer has the shape of 28\*100, which means each token in the sentence has been represented with a vector of size 100. The convolutional layer is defined with 32 filters and a kernel size of 4 with the relu activation function. Then it is followed by a pooling layer that reduces the output of the convolutional layer by half. The output layer contains 20 nodes, each node represents an emoji and contains the possibility that the emoji matches the input tweet. The highest possibility node would be our prediction result. We used the built-in categorical\_crossentropy function to calculate the loss and use stochastic gradient descent to update the weights.

We tuned many parameters for experiment and the best accuracy we reached on the test dataset is 29.5%. Below are the accuracy for each emoji and the number of training samples of each emoji that included in the training set. It seems that our model did a good job at predicting emojis like heart, lol, fire, flag, sun and tree but haven’t learned well on how to predict the other emojis.

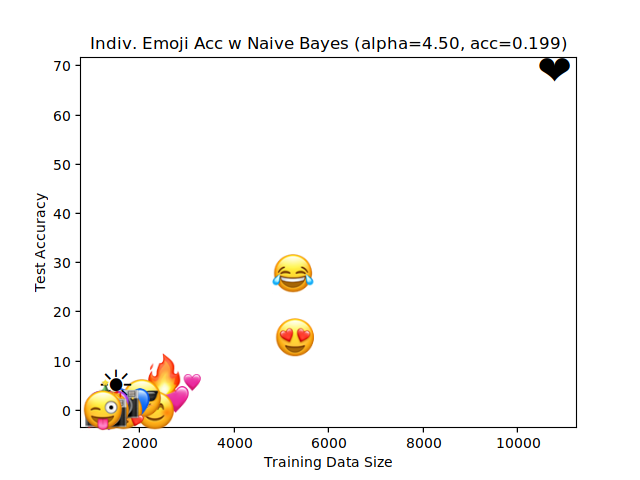
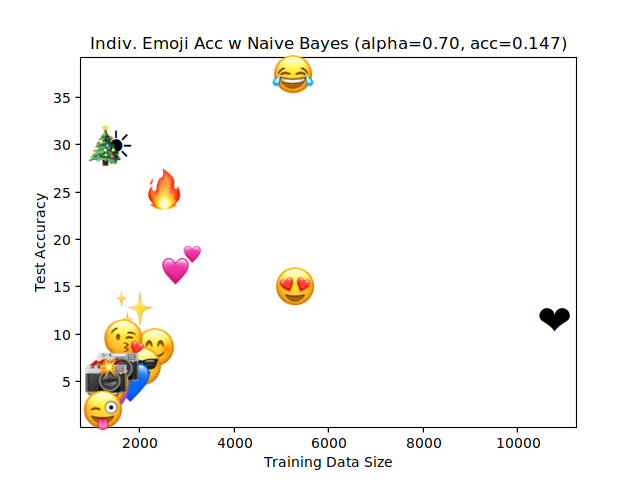
Comparatively, the CNN reached a higher accuracy of 29.5%, whereas Naive Bayes reached the accuracy of 21.23%. This discrepancy in accuracy was quite reasonable since the CNN model is more complicated and it usually takes more time to train. However, the accuracy is both quite low in general.

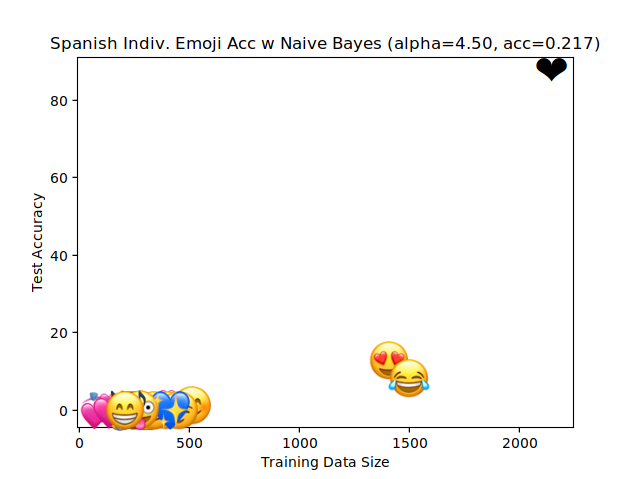
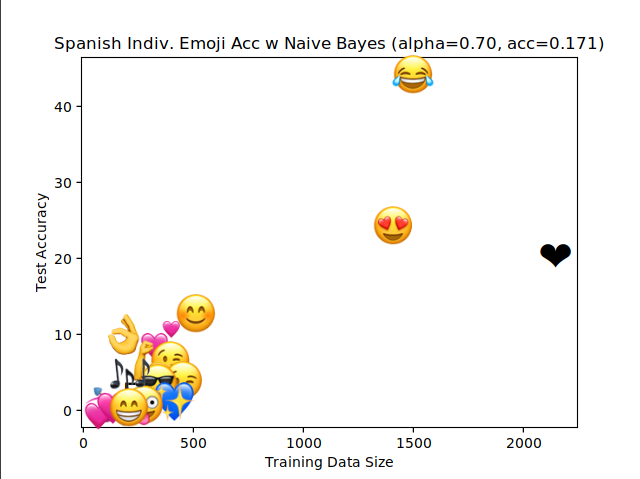
In Appendix 3, we provided the data visualizations of the accuracies for each individual emoji, and as seen in each graph, there was definitely a bias towards the heart emoji, not only in test accuracy but also in training set size. More than 20% of the tweets have the heart emoji, but only 2% of the tweets have emojis like blue\_heart or winking\_face. This was potentially one main reason which the overall accuracy was so low, but some attempts at reducing the number of training data sets that contained the heart emoji only decreased the overall accuracy. The unbalanced training set will affect the performance of both algorithms.

Another potential reason for a low accuracy is that a majority of the emojis are positive emojis, such as a sun, a laughing face, heart eyes, and various other hearts. Thus, the words used in tweets containing the positive emojis are likely to be similar. As a result, since the heart emoji is the most common emoji, and positive tweets are likely to have similar words, the algorithm seems to be predicting the heart more frequently than any other emoji. According to the graph, we can conclude that the larger the training data size, the better the accuracy. This was also seen in the disparity between the English and Spanish training set data, since the accuracy for the English predictions were slightly higher than the Spanish predictions on both algorithms.

Overall, the results of both the methods we implemented for emoji prediction did not yield accuracies as high as we would’ve liked, which may have been caused by many differing factors. The limitation of using Naive Bayes is that independence is assumed, and in any type of natural language, there shouldn’t be assumption of independence between certain keywords. Other models to explore might be Markov Models, or using a more sophisticated BERT (Bi-directional Encoder Representations from Transformers) language model. These more language- specific models may take into account the dependence between words and phrases, potentially resulting in higher accuracy of the predicted emoji.

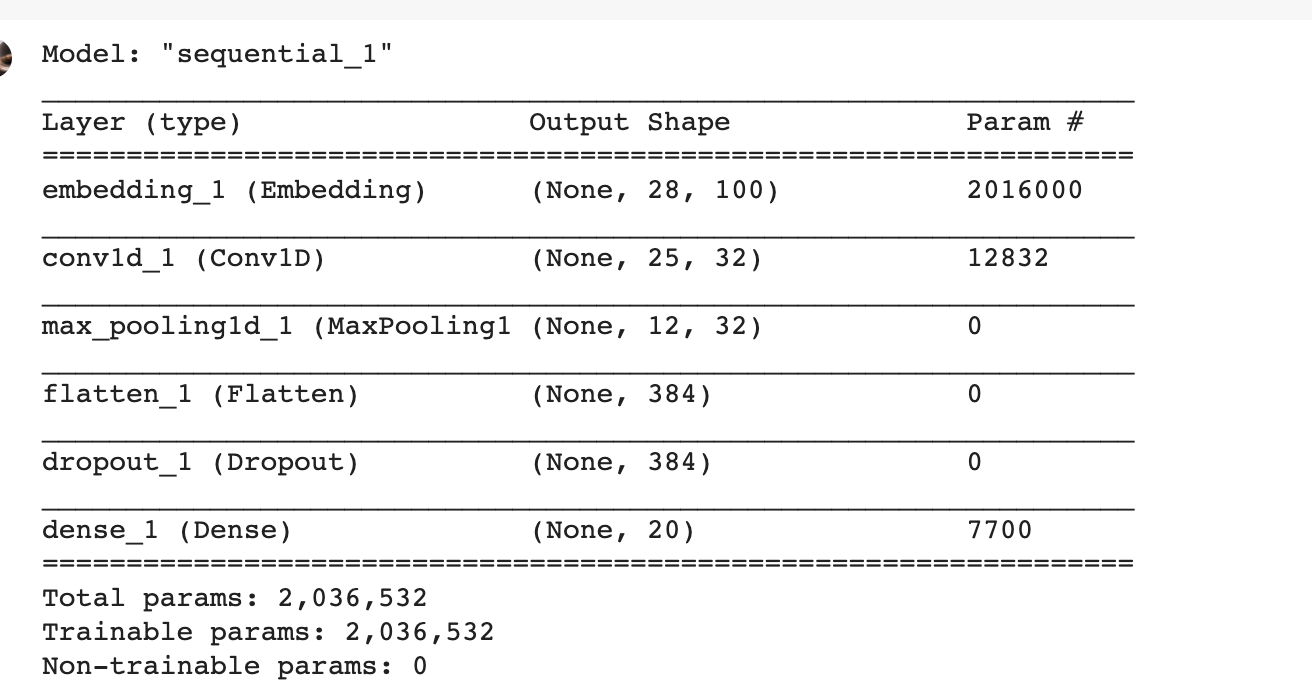
**Appendix 1:**



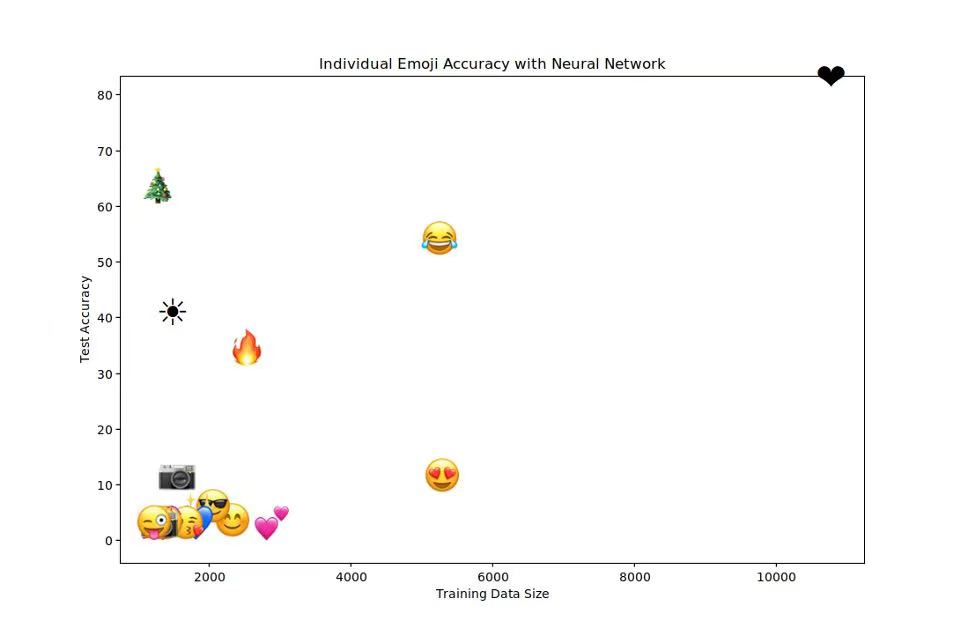
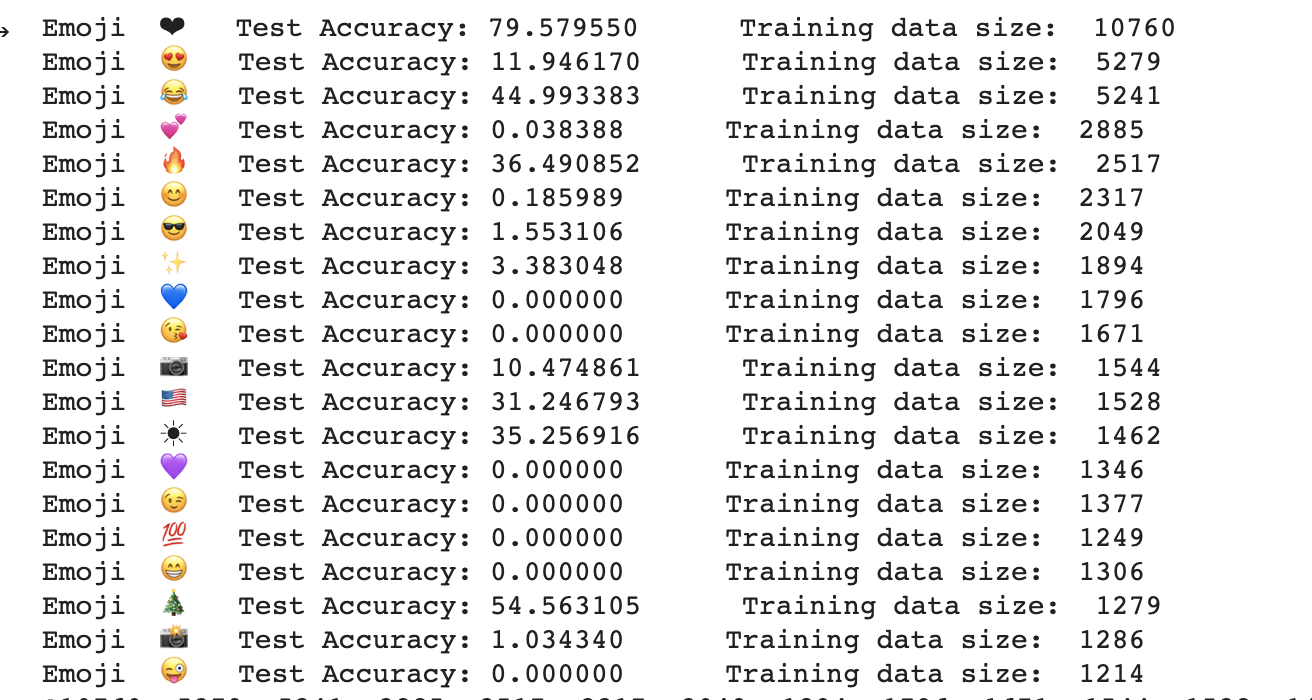


*Comparison of test accuracy of emojis on alpha = 0.70 and alpha = 4.50, using English (top) and Spanish (bottom) datasets.*

**Appendix 2:**



*CNN model, displaying the different layers.*

**Appendix 3:**

*Test accuracies of emojis as predicted by the Convolutional Neural Network.*