Emojineering

Applied Natural Language Processing

Daniel Chen | Carlo Liquido | Hadrien Renold

1. Project Goals

1.1 Original intent

The ultimate goal of our project was to create an emoji tool that could assign the appropriate emoji to a given piece of text. We stripped all emojis from each document (ie, each tweet) and treated them as our target labels. If our predictions proved accurate enough, we would pursue further work into inserting emojis into larger corpuses. Below are the implementation steps we would tackle in order of success:

- Step 1: Binary Prediction
  - Given our training data, we would first predict whether or not it would be appropriate for a given tweet to have an emoji. Roughly four-fifths of our data contained tweets without emojis, meaning a score of 80% was our baseline. This preliminary prediction would shed light on how emojis are used, ie sentiment, complementary purposes, substitution, etc.

- Step 2: Categorization
  - In order to predict use of emoji, we first needed to categorize emojis at a more abstracted level. With more than 700 emojis in the entire bin of possible emojis, finding a method to accurately categorize these emojis was paramount to the success of our project.

- Step 3: Emoji Prediction by Cateogry
  - The final step will be to classify the category of emoji, where an emoji is appropriate. Our analysis from the second step will dictate our accuracy level. We plan to determine the granularity of our categories in conjunction with our training process. For example, 'grinning face' and 'neutral face' could be abstracted at a higher level of just 'face. A more granular level, however, would be to differentiate 'person-emotion' from 'person-physical-state'.

- Step 4: Predicting Number of Emoji
  - From our preliminary analysis, we noticed that many tweets contained more than one emoji in different variations. If we were able to accurately predict the most likely category of emoji, we planned to extend our project into assigning multi-category emojis where appropriate.

- Step 5: Placement of Emoji
  - The steps above fail to take into account where it is most relevant to insert an emoji. It assumes that some combination of emoji can

    be appended to the end of a document. This step would seek to address this issue and split the document into windows of text for relevant emoji use.

- Step 6: Generation of Emojis on Random Text
    - Our final step would be to generate emojis on an entire corpus rather than a tweet of 140 characters. It would look at corpuses with varying writing styles and syntactic structures. It would call upon the success of each of the aforementioned steps.

1.2 How far we got

We were able to fulfill steps one through three, i.e. predicting whether or not an emoji was appropriate, categorizing emojis into sensible buckets, and predicting the category of emoji based on our groupings. After a number of refinements to our pipeline and categorizations, we were not able to accurately predict emojis to justify moving on to further steps. The majority of our time was spent on cleaning and normalizing our data as well as exploratory analysis. We developed a robust set of features and ran our training set on a number of classifiers.

1.3 Future Work

Our findings inform the need for techniques that are more rigorous and unique to tweets. Because tweets are short in nature and specifically used as a conversational tool, we could do more analysis on how and why emojis are used. The features we created were suited for larger and more standardized pieces of text. Given this knowledge we have listed a number of potential steps to take for future work:

- Find a way to link tweets by conversation to create richer documents
- Conduct more rigorous sentiment analysis on how tweets are used
- Supplement our dataset with other social media resources
- Explore other methods of clustering and categorizing emojis which reflect real use
- Analyze Word2Vec outcomes, as this method afforded the best results

2. Description of Data

2.1 Overview of Data

Our dataset consisted of 1 million tweets streamed from the Twitter API over a 2-month period in 2013. All tweets originated in California. This dataset was originally collected for a class project in « Scalable Spatial Analytics » offered by the Civil Engineering department and attended by Carlo Liquido and Daniel Chen. The data contained an anonymized user ID, tweet ID, latitude and longitude of the tweet, tweet text content and a timestamp. The tweet content was not preprocessed and still contained identifiable information such as handles and urls.

2.2 Data Cleaning

Moreover, since the tweet content had not previously been preprocessed and tweets are inherently messy we spent a lot of our time in cleaning the data. The following are the steps we took towards cleaning our data in view of tokenization and prediction:

1. We replace @handles with the keyword “hdl”, which was not present in our dataset
2. We replaced urls with the keyword “url”. For both of these we decided to keep the information about whether there was a handle or an url present as this could be used as a feature for prediction.
3. We transformed emoticons[1] into emojis[2]. Indeed, most modern website and keyboards now automatically transform emoticons to emojis (for example Slack from which we inspired our processing[3]) while typing, so we can consider that they are semantically equivalent or at least very close. Moreover, this allows to create more data with emojis.
4. Many tweets contain retweeted content in quotations, or after the keyword RT. We decided to split this content, to be able to identify independently original content from the user, and retweeted content. Unfortunately, our dataset did not allow to reconstruct all the conversation chain, which could have helped us determine general trends in the conversation or change of topics, which might have been good predictors of emojis.
5. We split emojis from the text, as this is the label that we are trying to predict. We will talk in further sections of how we clustered those labels to create categories. We then assigned those categories to the data.
6. We removed all tweets that were not in English based on the character set being used.
7. Finally, we split hashtags into individual words as often hashtags are used to lump multiple words together within a sentence. This allows to keep the

[1] “a representation of a facial expression […] formed by various combinations of
[2]“a small digital image or icon used to express an idea, emotion, etc., in electronic communication.” http://dictionary.reference.com/browse/emoji
[3] https://slack.zendesk.com/hc/en-us/articles/202931348-Emoji-and-emoticons

content that is contained within the hashtag. This was done by first splitting on capitalized words within the sentence, followed by a dictionary lookup for lowercase hashtags.

2.3 Preprocessing

Following that we tokenized the clean dataset using a Twitter-aware tokenizer obtained from http://sentiment.christopherpotts.net/code-data/happyfuntokenizing.py. We also used character ngrams to compensate for the fact that Twitter spelling can be very bad and/or abbreviated. Character ngrams takes a sliding window of set size of words and returns all subsequence of the word of that window size.

Our final data preprocessing step was to remove all tweets shorter than 50 characters. This hard limit was set as a compromise between being inclusive of all tweets and having enough content to train and predict on. Moreover, from the data we noticed that there were multiple types of behaviors with regards to the use of emojis, one of which the emoji was the sole content of the response, or a comment of the retweeted content. In this scenario, since we do not have the history of the conversation we don't have any content to train or predict the emoji.

Prior to cleaning the data, we split the data into a training set containing 80% of the data, and a held out test set with 20% of the data. We developed the cleaning and preprocessing steps with the training data and only ran it on the test data at the very end.

In order to recreate the cleaning and preprocessing pipeline, this is the order in which notebooks have to be run:

1. data_partitioning.ipynb

For each of tweets_training.json and tweets_test.json:

2. data_cleaning.ipynb
3. data_preprocessing_assignManualCategories.ipynb
4. data_preprocessing_removeShortTweets.ipynb
5. modeling_pipeline.ipynb

3. Exploratory Work

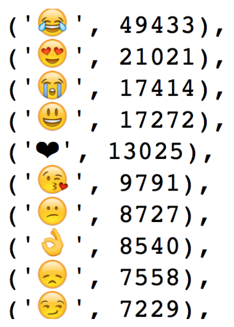
3.1 Frequency Distribution

Because there are more than 700 emojis in existence, we looked to understand how people used emojis within the domain of Twitter. We first examined the most commonly used emojis, which are displayed in Figure 1 in terms of raw count.

The word of the year, expectedly, was used much more frequently than any other emoji. The top ten most used contained mostly emojis that could be categorized as a ‘face’. Additionally, viewing the frequency distribution of emojis sentimentally, frequently used emojis were mostly composed of positive rather than negative emotions.

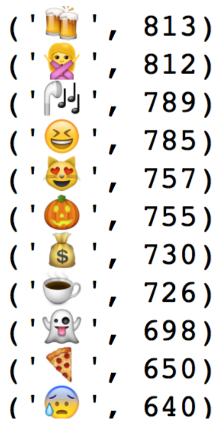
We also looked at the less commonly used emojis to further understand emoji usage behavior (Figure 2). There were a total of two human faces, a single female, one coffee, a beer icon, and a few other occasion purpose emojis. These Emojis have shown the possibility of people substituting words with emojis. After digging into our dataset more rigorously, we were able to confirm the existence of such substituting behavior. People would use the pizza icon without typing the words pizza. This behavior has shown the necessity of understanding the semantic use behind emojis.

From our tweets, we also came to understand that people frequently use more than one emoji in a given tweet. In the cases where people do use emojis, the average number of emojis used was 1.5 emojis. Out of the million tweets, 20% of the tweets contain emojis while 15% of the text contains emojis with faces. We also took a look at the common emoji bigrams to understand how people used emojis together (Figure 3). The result shows, again, the popularity of face emojis over object emojis. Another observation from bigrams was that people used emojis together in a certain pattern. The two Halloween emojis are popular because they are semantically connected, as well as the two celebration emojis.



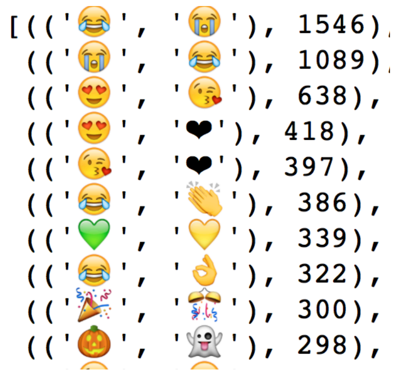
```
('😂', 49433),
('😍', 21021),
('😭', 17414),
('😃', 17272),
('❤', 13025),
('😘', 9791),
('😕', 8727),
('👌', 8540),
('😞', 7558),
('😏', 7229),
```

Figure 1



```
('🍻', 813)
('🙅', 812)
('🎧', 789)
('😆', 785)
('😻', 757)
('🎃', 755)
('💰', 730)
('☕', 726)
('👻', 698)
('🍕', 650)
('😰', 640)
```

Figure 2



```
[(('😂', '😭'), 1546)
 (('😭', '😂'), 1089)
 (('😍', '😘'), 638),
 (('😍', '❤'), 418),
 (('😘', '❤'), 397),
 (('😂', '👏'), 386),
 (('💚', '💛'), 339),
 (('😂', '👌'), 322),
 (('🎉', '🎊'), 300),
 (('🎃', '👻'), 298),
```

Figure 3

After examining bigrams, we used the collocations package in nltk to explore collocations of bigrams. We noticed that there were two main types of collocations of emojis. In the first observed scenario, people tended to use two identical emojis together, as seen from Figure 4. Some tweets tended to repeat the same emojis more than once.

The second type of Emoji collocation is to build up semantic relationship between the first Emoji with the second Emoji. As we can see in figure 5, French fries would be used with hamburgers and fried shrimp would be used with curry plate. Such usage of Emoji tells a slightly more sophisticated story than just a single one.

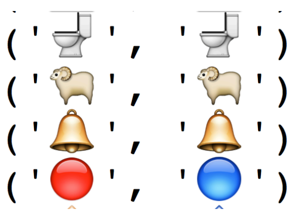


Figure 4

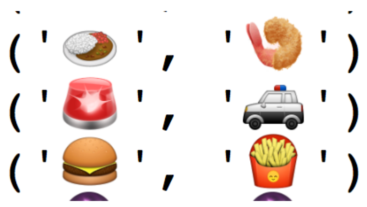
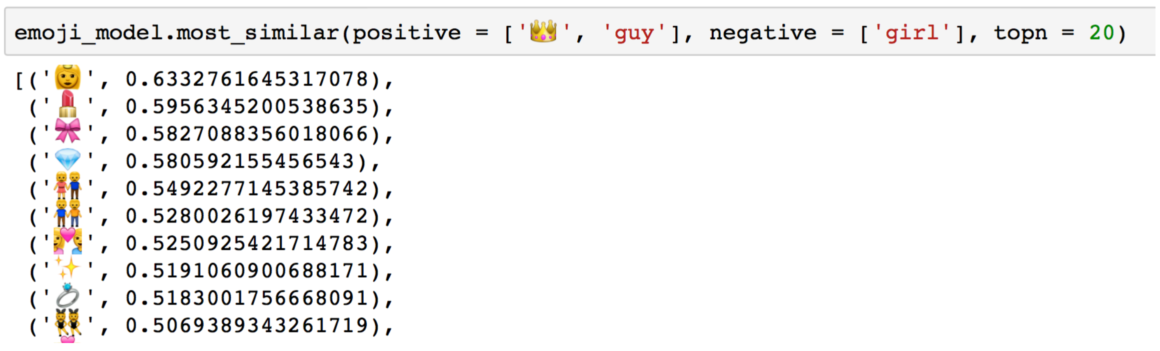


Figure 5

3.2 Word2Vec

Building upon semantic usage of emojis, we decided to use Word2Vec to further explore the semantic relationships between emojis. We have imported Word2Vec from genism package and trained the model with tokenized tweets. We adjusted the parameters and window size to optimize the outcome. The results were surprisingly informative. We were able to replicate the well-known example of the original word2Vec model (i.e., positive = [👑, guy], negative = ['girl'], resulting in 'princess'). Such result is stunning and we were surprised by the power of Word2Vec. If you take an even deeper look at the results, one would be surprised by the consistency of the output. The top 9 outputs are all related to the concept of queen and female. In order to test the model further, we tested the tool with an array of other icons in various combinations. By training our dataset on Word2Vec, we were able to infer semantic meaning behind emojis that would have been untenable otherwise.

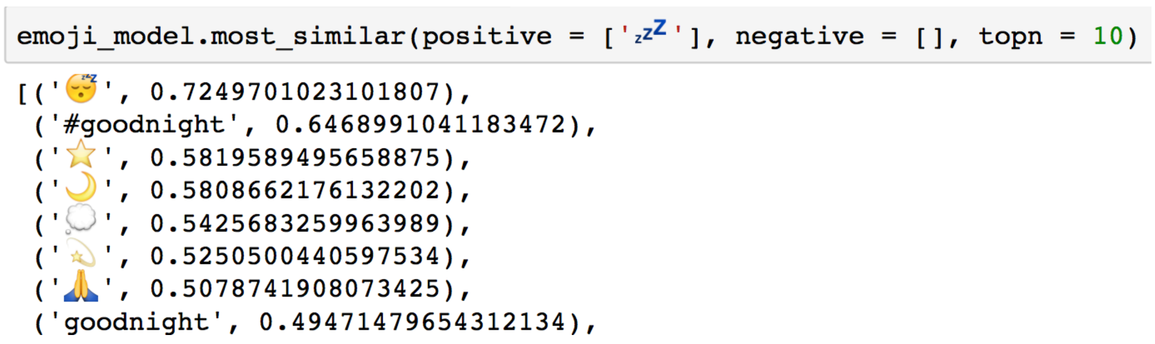


```
emoji_model.most_similar(positive = ['👑', 'guy'], negative = ['girl'], topn = 20)
```

```
[('👸', 0.6332761645317078),
 ('💄', 0.5956345200538635),
 ('🎀', 0.5827088356018066),
 ('💎', 0.580592155456543),
 ('👫', 0.5492277145385742),
 ('👫', 0.5280026197433472),
 ('💏', 0.5250925421714783),
 ('✨', 0.519106090068817 1),
 ('💍', 0.5183001756668091),
 ('👯', 0.5069389343261719),
```

Figure 6

Additionally, hashtags provided further insight into the use of emojis. Figure 9 shows "#goodnight", and "goodnight" as among the top ten output of the sleep emoji, 'ZZZ'. We were surprised by the detection of hashtags and upon further analysis found that hashtags were used much differently than regular text.



```
emoji_model.most_similar(positive = ['💤'], negative = [], topn = 10)
```

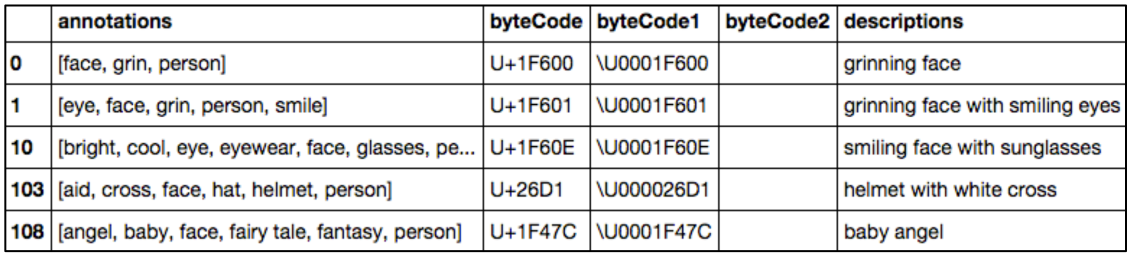
```
[('😴', 0.7249701023101807),
 ('#goodnight', 0.6468991041183472),
 ('⭐', 0.5819589495658875),
 ('🌙', 0.5808662176132202),
 ('💭', 0.5425683259963989),
 ('💫', 0.5250500440597534),
 ('🙏', 0.5078741908073425),
 ('goodnight', 0.4947147965431213 4),
```

Figure 7

4. Clustering & Categorization

4.1 K-means Clustering

For the first method of clustering emojis, we chose to use the K-means clustering technique. In order to create a set of feature vectors, we scraped the Unicode Consortium page for the set of keyword annotations associated with each emoji (as shown below in Table 1). As nearly 80% of all emojis contain the keyword 'face', we chose to subset our categorization by this element. Because k-means is sensitive to outliers and noise, only the top 50 keywords ordered by frequency were used. Each emoji was then classified with a binary score of 1 or 0 (that is, positive or negative respectively) for each of the keywords to make our feature vector.



	annotations	byteCode	byteCode1	byteCode2	descriptions
0	[face, grin, person]	U+1F600	\U0001F600		grinning face
1	[eye, face, grin, person, smile]	U+1F601	\U0001F601		grinning face with smiling eyes
10	[bright, cool, eye, eyewear, face, glasses, pe...	U+1F60E	\U0001F60E		smiling face with sunglasses
103	[aid, cross, face, hat, helmet, person]	U+26D1	\U000026D1		helmet with white cross
108	[angel, baby, face, fairy tale, fantasy, person]	U+1F47C	\U0001F47C		baby angel

Figure 8

The results of our clustering are shown in below in Table 2. After a number of iterations, a cluster of k = 5 produce the best outcome. Although the clusters make sense semantically, they are not distributed in a way that reflects sentiment. For example, ‘Cluster 2’ contains every emoji with the keyword ‘cat’. It does not discriminate on emotion keywords, such as happy, sad, or surprised. One can assume that emotions should hold greater value. Future work could involve weighting words differently (e.g., adjectives versus nouns). Moreover, the categories were not distributed equally based on frequency of use.

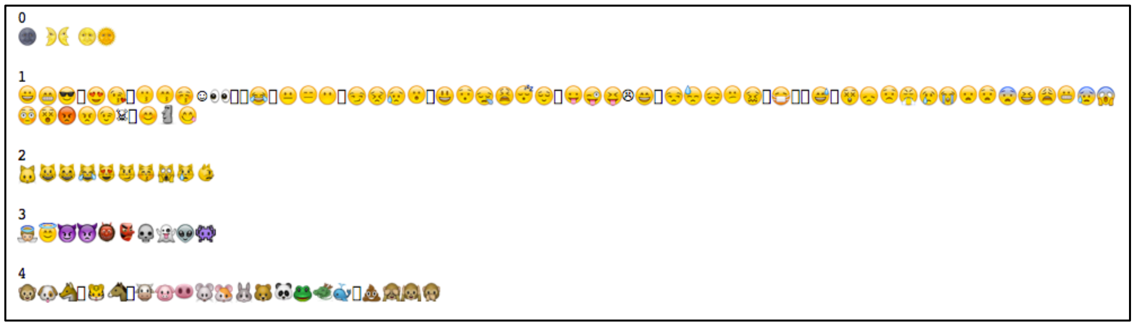


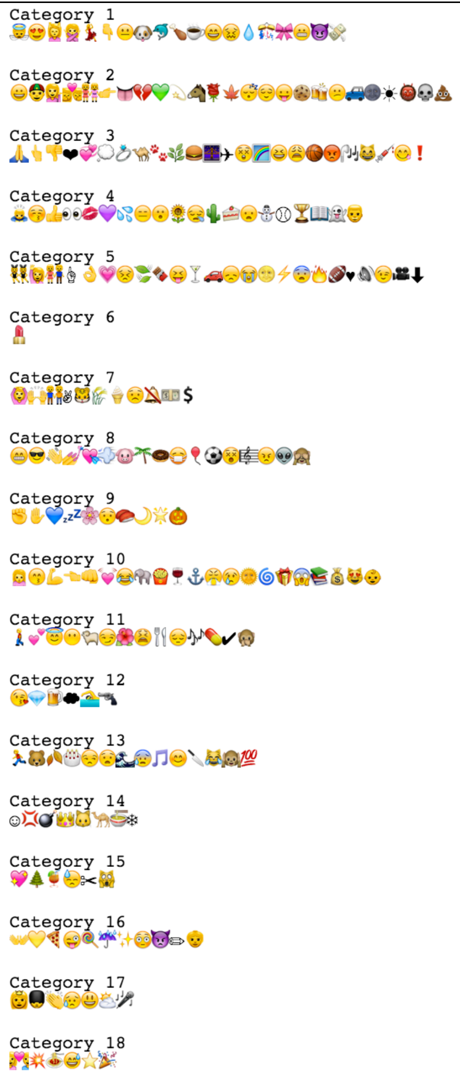
Figure 9

4.2 Word2Vec Hierarchical Clustering

After doing clustering on annotations, ie on how the emojis were designed to be used, we attempted to cluster on how emojis were actually being used. After having seen the power of word2vec and the encouraging results with emojis, we determined that the word2vec measure of similarity was a good candidate for clustering. Since word2vec gives a pairwise measure of similarity it seemed logical to attempt hierarchical clustering. We ran it with a complete linkage clustering method and only on the emojis present in the word2vec model. The word2vec model was trained on the whole clean training set with a window size of 10, thus attaching more importance to the semantics of the relationship than to the syntax. The feature vector size was set to 200, and we set the minimum count to 100, so we only look at common words and emojis in the dataset.

This enabled us to construct the following dendrogram and the respective categories below. From the dendrogram we were able to select a distance measure and the corresponding number of cluster (in this case 18) that correspond to the highest jump in distance.

It is hard to make sense of the categories created. Unfortunately, because of lack of time we didn't have the opportunity to see if these categories would have allowed to improve the prediction results than the manual categories defined or the clustering using kmeans. However, since our ultimate goal was to provide a tool for assisting humans in attaching emojis to tweets, we concluded that those categories would not make sense to the user. The same process was run subsetting to only faces emojis.



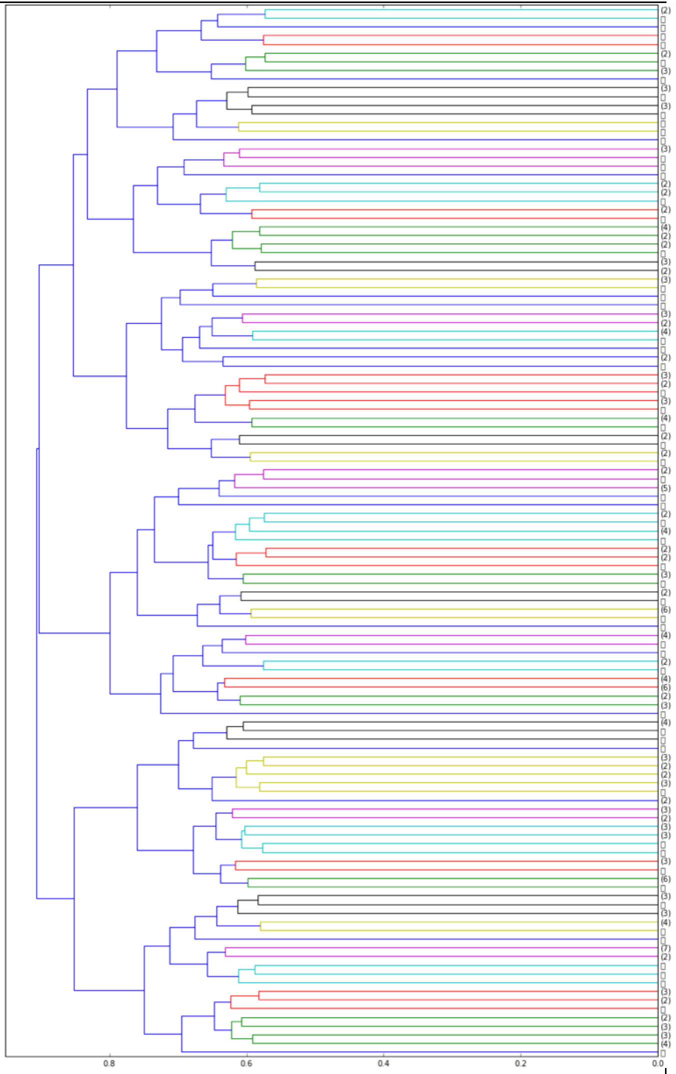


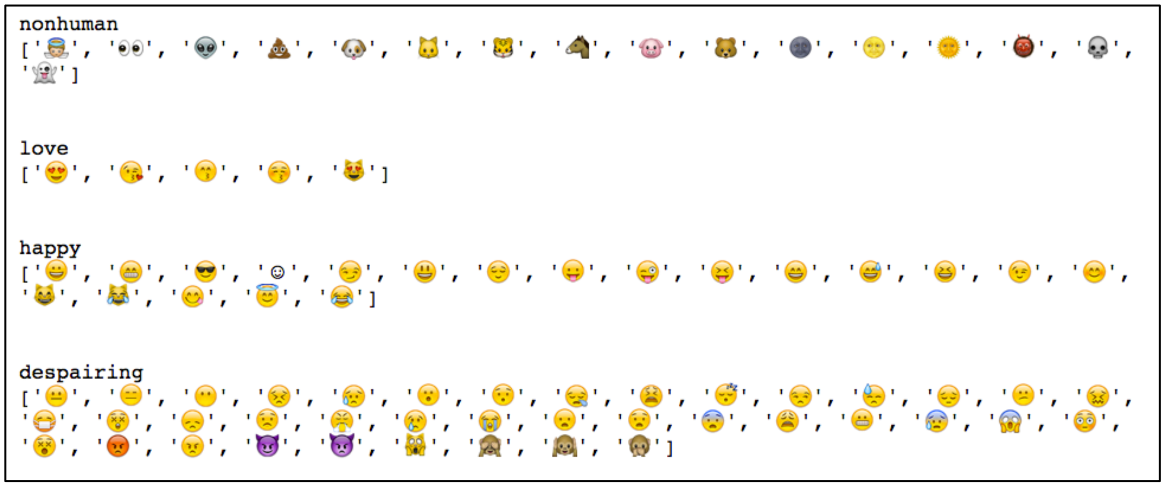
Figure 10	Figure 11

4.3 Manual Labelling

After failing to use supervised learning techniques to create meaningful and well-distributed categories, we decided to manually categorize emojis with the keyword ‘face’. Although manual labeling introduces a certain level of bias, the grouping reflected a more human…. Only face emojis that were identified as being used in the training set were used for labeling. We identified four meaningful groups: ‘non-human’, ‘love’, ‘happy’, and ‘despairing’. After labeling the data with these categories, we needed to resolve two issues:

1. Tweets with both Face emojis and Non-Face emojis
   There were a number of tweets which contained multiple emojis that were labeled as having the keyword ‘face’ (i.e. could be categorized as either ‘non-human’, ‘love’, ‘happy’, and ‘despairing’) and from non-face categories. The decision was made to remove all non-face emojis and subsequently categorize only based on the remaining face emojis.

2. Tweets with Face emojis from different categories
   There was also the situation of tweets with multiple emojis from different categories (e.g. a tweet that contained both a ‘happy’ face and a ‘despairing’ face). These tweets were removed from the training set when attempting to predict category of emoji.



```
nonhuman
['👼', '👀', '👽', '💩', '🐶', '🐱', '🐯', '🐴', '🐷', '🐻', '🌚', '🌝', '🌞', '👹', '💀',
'👻']


love
['😍', '😘', '😗', '😚', '😻']


happy
['😀', '😁', '😎', '☺', '😏', '😃', '😌', '😛', '😜', '😝', '😄', '😅', '😆', '😉', '😊',
'😸', '😹', '😋', '😇', '😂']


despairing
['😐', '😑', '😶', '😣', '😥', '😮', '😯', '😪', '😫', '😴', '😒', '😓', '😔', '😕', '😖',
'😷', '😲', '😞', '😟', '😤', '😢', '😭', '😦', '😧', '😨', '😩', '😬', '😰', '😱', '😳',
'😵', '😡', '😠', '😈', '👿', '🙀', '🙈', '🙉', '🙊']
```

Figure 12

5. Description of Algorithms

5.1 Features

To predict the Emojis, we came up with a list of features that we extracted from the data.

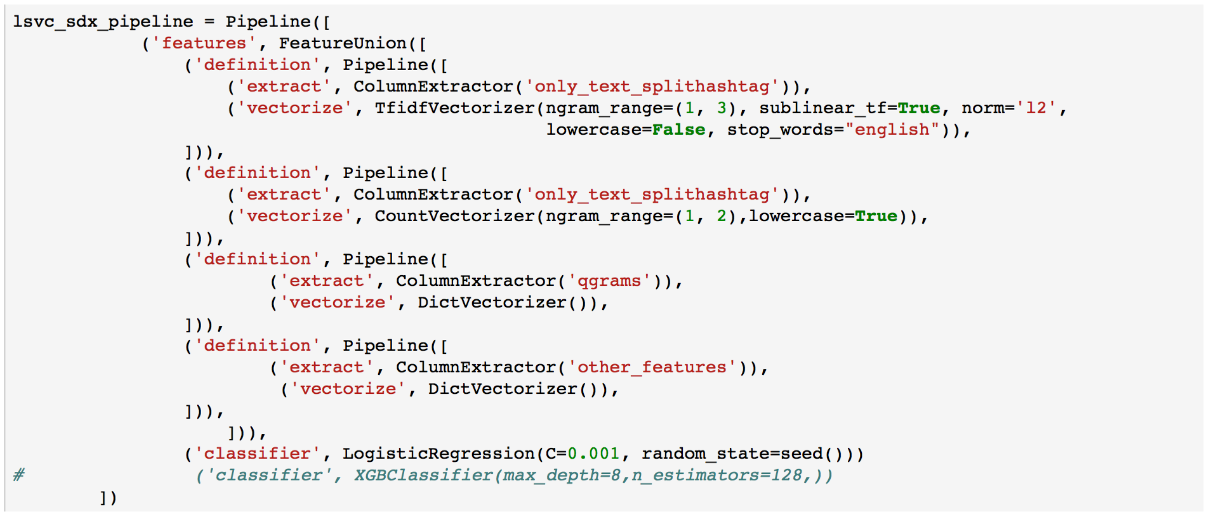
- Length of tweets
    - The number of characters in a given tweet.
- Number of punctuation
    - The number of punctuations in a given tweet.
- Number of hashtags
    - The number of hashtags in a given tweet.
- Sentiment score
    - Ranging from -1 to 1, we used the textblob package to score the sentiment of each tweet.
- IsReweets
    - Binary 0 and 1 variable that labels a tweet to decide whether it contains a retweet or not.
- Q-grams
    - From the package of abydos, we extracted q-grams of a tweet as the feature and use DictVectorizer to transform it
- Tfidf
    - Using TfidfVectorizer from sklearn we used Tfidf as a feature, setting an ngram range from 1 to 3, sublinear as true, lower case as false, and stop words of English.
- Count Vectorizer
    - Using sklearn's package, we used CountVectorizer to extract features while setting lowercase as false and an ngram range of 1 to 2.
- Synset
    - Extracting synets from Wordnet proved to take too much time
- POS tagging
    - Extracting POS from pos_tag in the nltk package, we eventually aborted due to the frequent grammatical errors native to twitter

5.1 Classifiers

We also set up pipelines to test different set of features and classifiers. Figure 10 shows a snippet of our pipeline code. We tried an array of classifiers and ranked them respectively on their performances.

- Logistic Regression
    - Consistently performed better than the other classifiers while also taking the least amount of time. Lower C value gave slightly better results than larger C values.
- Linear SVC

    - Performed the worst out of all four classifiers. Though it performed well in terms of time, the accuracy score was generally poor.
- XGB Classifier
    - Required large bandwidth to run. Performed slightly worse than logistic regression and was very inconsistent.
- Random Forest Regressor
    - Similar in results to XGB Classifier in terms of both time and in terms of accuracy and consistency.



```
lsvc_sdx_pipeline = Pipeline([
            ('features', FeatureUnion([
                ('definition', Pipeline([
                    ('extract', ColumnExtractor('only_text_splithashtag')),
                    ('vectorize', TfidfVectorizer(ngram_range=(1, 3), sublinear_tf=True, norm='l2',
                                                  lowercase=False, stop_words="english")),
                ])),
                ('definition', Pipeline([
                    ('extract', ColumnExtractor('only_text_splithashtag')),
                    ('vectorize', CountVectorizer(ngram_range=(1, 2),lowercase=True)),
                ])),
                ('definition', Pipeline([
                        ('extract', ColumnExtractor('qgrams')),
                        ('vectorize', DictVectorizer()),
                ])),
                ('definition', Pipeline([
                        ('extract', ColumnExtractor('other_features')),
                         ('vectorize', DictVectorizer()),
                ])),
                    ])),
                ('classifier', LogisticRegression(C=0.001, random_state=seed()))
#                ('classifier', XGBClassifier(max_depth=8,n_estimators=128,))
        ])
```

Figure 13

After creating features and applying them in variations to our classifiers, Logistic Regression proved to be the best. We had partitioned our data into a training set and a test set. Only until the final submission did we apply our model to the test set. Our baseline score for binary prediction was 80%. Our final score on the test was 81.2%.

6. Contribution of each Team Member

Each team member contributed significantly to different aspects of the Emojineering project. All three all of us contributed equally during brainstorming sessions. During the beginning stages of the project, Hadrien proposed a set of guidelines to follow in terms of workflow. This composed a combination of github, ipython notebook configuration, branch creation, and naming convention.

In the initial phrase of the project, Carlo and Hadrien acquired additional data by webscraping the ‘Unicode Consortium’ for annotations on each emoji. Hadrien then took the initiative of cleaning the data by replacing URLs, tweets handles, cleaning retweets, and converting emojis. In the meantime, Daniel worked on tokenization, emoji extraction, and cleaning non-English language. Carlo worked on further cleaning and splitting of hashtags. After cleaning the data, Daniel and Carlo worked on exploratory analysis of the training set. Carlo explored emoji distribution and the differences between emoji object usage and face usage, while Daniel worked on training the Word2Vec model and emoji collocation.

After data exploration, we moved on to categorization. Carlo applied K-means Clustering and created several ways of categorizing the emojis manually. Hadrien applied Word2Vec Hierarchical Clustering to add an additional option for categorization. After assessing both supervised-learning categorizations, our team decided to use the manually labeled categorizations that Carlo created. After categorization, Daniel actively worked on feature engineering and setting up the pipelines for prediction. Hadrien paired-programed with Daniel to assist on pipeline setup. In the final phrase, Carlo and Hadrien made further contributions to clean the data and relabel the categories for Daniel to run the predictions on. Carlo designed the presentation with Prezi and the poster with Adobe Illustrator. Hadrien and Daniel applied the final touches to the prediction and developed the Word2Vec demo.