Emoji Predictor: Survey of Classification Problem Approaches and Algorithms

Alexandra Gamez,
Texas A&M University
Department of Computer Science & Engineering

Abstract

This paper seeks to examine different, commonly used classification algorithms applied to predicting emojis in tweets. Emoji prediction can be posed as a classification problem and sentiment analysis problem. Using the emoji labels as a type of sentiment label and the tweets as documents. The documents will be stored as feature vectors using a "bag-of-words" model approach. The document set will be trained and tested using a custom Naïve Bayes algorithm along with optimized classification algorithms from the Scikit-learn Machine Learning library. Namely, optimized Naïve Based, Stochastic Gradient Descent, Support Vector Machines, Logistic Regression, K Nearest Neighbors, Decision Tree, Neural Networks, and Random Forest algorithm.

The problem is taken from the CodaLab competition for SemEval 2018, SemEval-2018 Task2, Multilingual Emoji Prediction The original consists of a training set of 500k English and 100k Spanish documents. The emojis in question are a list of 20 most commonly used emojis in USA tweets and 20 most commonly used emojis in Spain. The problem in this project is meant is modified to only examine English language tweets with a training set of 50k documents.

1 Motivation

Since the beginning of the semester, it has been a goal to create a sentiment of any scale, as this is a basic but important task in Data Science and Natural Language Processing. The final project proved to be an excellent opportunity to apply the knowledge gained in class to finally complete this task in a larger scale. Sentiment analysis is a classification problem. The problem can be proposed in many ways. Classification is one of the main problems in Machine Learning and exploring different algorithms as well as attempting to dive into the subject is a worthwhile pursuit.

2 Process

SemEval provides many tools for the completion of the project. This includes an emoji extractor. Given a tweet will return two files, one with the tweet content and one with the label for the tweet. An emoji mapping file. Shows which labels belong to which emoji. A twitter crawler. The program is simple enough that a simple command will get the crawler started and the tweets will be retrieved.

2.1 Tweet Retrieval

The provided twitter crawler is called, and the tweets are extracted automatically. After the tweets are retrieved the emoji extractor is called and the files for labels and words are created.

2.2 Scikit Machine Learning Classifiers

A big part of this project was learning how to use the Scikit Learn machine learning tools. These tools are used in industry and is a requested skill for Data Science positions.
Scikit provided a straight forward text classification example. The example code was very helpful in the implementation of the classifiers. The Scikit classifiers used are as follow: Multinomial Naïve Bayes, Stochastic Gradient Descent Classifier, Support Vector Machines, Logistic Regression, K Nearest Neighbors, Decision Tree, and Neural Networks.

2.3 My Naïve Bayes

Since the beginning, the goal was to implement a machine learning algorithm chosen was the Naïve Bayes algorithm.

$$P(X) = \frac{P(X|C) * P(C)}{P(X)}$$

X is a document and C is a class. P(C|X) is the probability of a class given a document which is what is being looked for. P(X|C) is the probability of a document given a class, a generative model. P(C) is he prior probability of a class. P(X) is the probability of a document. However, this denominator is left out because when comparing different classes, both will have the same denominator and thus is not important when it comes to calculation.

In the code, there is a class named 'train' that holds all the important information from a document. It is important to note that CountVectorizer from Scikit-learn (open source machine learning tool) is used to create the term document matrix in TF-IDF form which should make the classifier robust against common stop words and make rare words important. A term document matrix was meant to be created however it proved to be to slow and difficult to implement efficiently. It is imported from sklearn.feature_extracting.text from the sklearn python package. The class holds two CountVectorizers (which is a *m* x *n* matrix) where *m* is the number of documents and *n* is the number of word features. One for the tweet content (word features) and one for the labels (emoji classifiers). The vocabularies for both the tweets (word vocabulary) and the labels (emoji label vocabulary) are saved. Also, the sums of the x and y axis are extracted from the matrices of both. For the tweets, the sum of the columns represents total counts of each feature word. The sum of the rows represents the number of words in a document. For the labels, the sum of columns is the number of documents that use a certain emoji.

Term Document Matrix

$$\begin{array}{c|cccccc|c}
 & W1 & W2 & W3 & . & . & WN & \\
D1 & [\ 1 & 1 & 1 & & & 0\ ] & 3 \\
D2 & [\ 0 & 0 & 1 & & & 0\ ] & 1 \\
D3 & [\ 1 & 0 & 1 & & & 1\ ] & 3 \\
. & & & & & & & \\
. & & & & & & & \\
DN & [\ 0 & 0 & 0 & & & 1\ ] & 1 \\
 & [\ 2 & 1 & 3 & & & 1\ ] &
\end{array}$$

Feature counts

# of features in a doc

Doc Label Matrix

$$\begin{array}{c|cccccc}
 & L1 & L2 & L3 & . & . & LN \\
D1 & [\ 1 & 0 & 0 & & & 0\ ] \\
D2 & [\ 0 & 1 & 0 & & & 0\ ] \\
D3 & [\ 1 & 0 & 0 & & & 1\ ] \\
. & & & & & & \\
. & & & & & & \\
DN & [\ 0 & 0 & 0 & & & 1\ ] \\
 & [\ 2 & 1 & 3 & & & 1\ ]
\end{array}$$

Class counts

The figures above represent the matrices stored in the 'train' class.

The other class, 'classifyDoc' is where the Naïve Bayes calculations occur. P(X|C) and P(C) can easily be extracted from the matrices as well as the sum. The important functions are *'doc_class_prob'* which gets P(X|C), probability of a bag-of-words given a class. Also, *'label_prob'*, which returns P(C).

Functions:

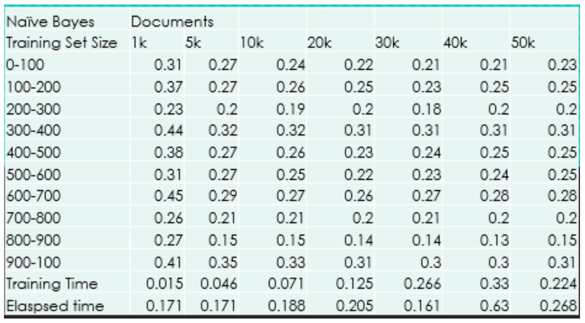
- *'doc_class_prob'* - gets P(X|C), probability of a tweet (feature vector) belonging to a class C.
- *'label_prob' – return P(C)*
- *'word_class_prob'* – returns probability of a word belonging to a certain class. Used by *'doc_class_prob'* on each word in the document.
- *'countc'* – return total count of a specific class (# of times emoji appears)

- *'countwc'* – returns the number of time a word appears in class.
- *'predict'* – returns a label (emoji) given a document (tweet).

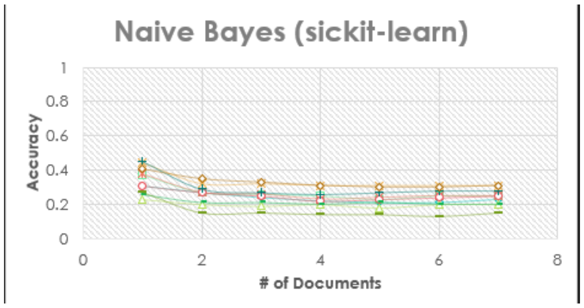
3 Algorithm Analysis

The algorithms are tested on accuracy versus each other. They are all given the same 10 test vectors for the same document sets. The document sets range from 1k to 50k.

3.1 Naïve Bayes (Scikit)

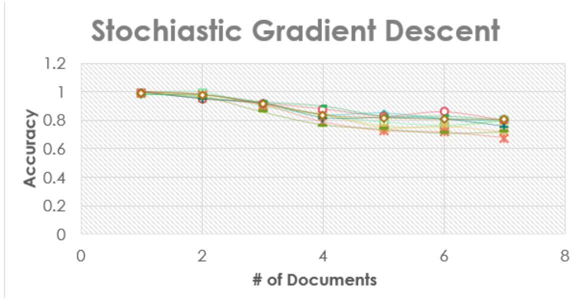


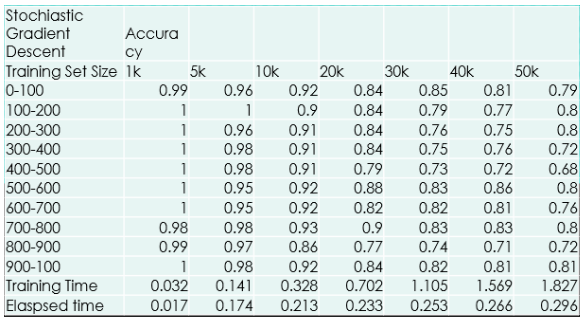
Naïve Bayes	Documents						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	0.31	0.27	0.24	0.22	0.21	0.21	0.23
100-200	0.37	0.27	0.26	0.25	0.23	0.25	0.25
200-300	0.23	0.2	0.19	0.2	0.18	0.2	0.2
300-400	0.44	0.32	0.32	0.31	0.31	0.31	0.31
400-500	0.38	0.27	0.26	0.23	0.24	0.25	0.25
500-600	0.31	0.27	0.25	0.22	0.23	0.24	0.25
600-700	0.45	0.29	0.27	0.26	0.27	0.28	0.28
700-800	0.26	0.21	0.21	0.2	0.21	0.2	0.2
800-900	0.27	0.15	0.15	0.14	0.14	0.13	0.15
900-100	0.41	0.35	0.33	0.31	0.3	0.3	0.31
Training Time	0.015	0.046	0.071	0.125	0.266	0.33	0.224
Elaspsed time	0.171	0.171	0.188	0.205	0.161	0.63	0.268



The figures above show the accuracy of the Naïve Bayes algorithm (Scikit). The accuracy only ever reaches a maximum of 41% and then decreases as more samples are added. It is evident that Naïve Bayes does not perform well for this problem. However, both training and testing where very fast. All under 0.268 seconds for both training and testing for 1k to 50k documents.

3.2 Stochastic Gradient Descent Classifier

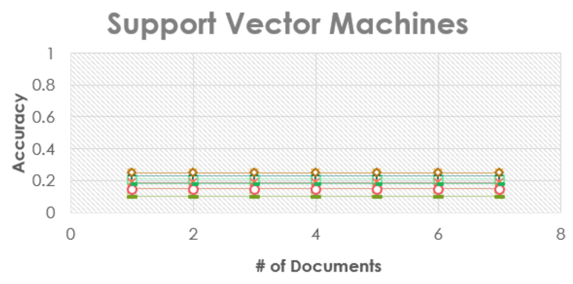




Stochiastic Gradient Descent	Accuracy						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	0.99	0.96	0.92	0.84	0.85	0.81	0.79
100-200	1	1	0.9	0.84	0.79	0.77	0.8
200-300	1	0.96	0.91	0.84	0.76	0.75	0.8
300-400	1	0.98	0.91	0.84	0.75	0.76	0.72
400-500	1	0.98	0.91	0.79	0.73	0.72	0.68
500-600	1	0.95	0.92	0.88	0.83	0.86	0.8
600-700	1	0.95	0.92	0.82	0.82	0.81	0.76
700-800	0.98	0.98	0.93	0.9	0.83	0.83	0.8
800-900	0.99	0.97	0.86	0.77	0.74	0.71	0.72
900-100	1	0.98	0.92	0.84	0.82	0.81	0.81
Training Time	0.032	0.141	0.328	0.702	1.105	1.569	1.827
Elaspsed time	0.017	0.174	0.213	0.233	0.253	0.266	0.296

Above is the analysis for the Stochastic Gradient Descent Classifier. The maximum accuracy is 100%. It decreases as more documents are added to the training set. Both the training and classifying periods are very fast. Although, the accuracy is at 0.7 at 50k documents and seems as if it might continue decreasing as document numbers increase.

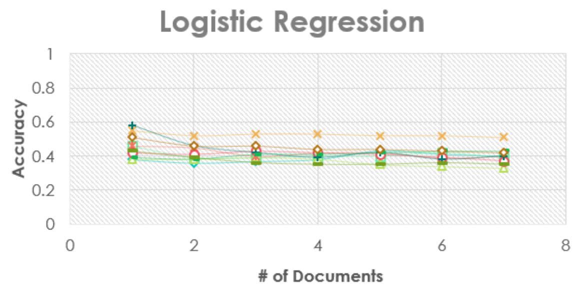
3.3 Support Vector Machines



Vector Machines	Accura cy						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	0.19	0.19	0.19	0.19	0.19	0.19	0.19
100-200	0.21	0.21	0.21	0.21	0.21	0.21	0.21
200-300	0.15	0.15	0.15	0.15	0.15	0.15	0.15
300-400	0.25	0.25	0.25	0.25	0.25	0.25	0.25
400-500	0.19	0.19	0.19	0.19	0.19	0.19	0.19
500-600	0.15	0.15	0.15	0.15	0.15	0.15	0.15
600-700	0.23	0.23	0.23	0.23	0.23	0.23	0.23
700-800	0.18	0.18	0.18	0.18	0.18	0.18	0.18
800-900	0.1	0.1	0.1	0.1	0.1	0.1	0.1
900-100	0.25	0.25	0.25	0.25	0.25	0.25	0.25
Training Time	0.674	14.68	58.409	253.598	572.56	1041.821	1344.291
Elapsed time	0.673	2.44	5.304	9.736	18	19.182	21.89

The accuracy for Support Vector Machines was surprisingly very low. The training time was slow. About 20 minutes for 50k documents with it set to increase significantly as the document count increased. The accuracy did not exceed 25%.

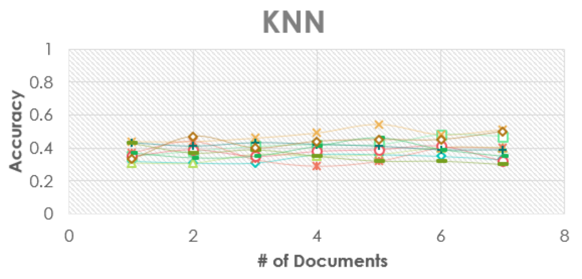
3.4 Logistic Regression



Logistic Regression	Accuracy						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	0.38	0.36	0.37	0.38	0.43	0.41	0.4
100-200	0.46	0.39	0.39	0.4	0.4	0.42	0.39
200-300	0.38	0.38	0.39	0.38	0.35	0.34	0.33
300-400	0.55	0.52	0.53	0.53	0.52	0.52	0.51
400-500	0.46	0.45	0.4	0.42	0.42	0.39	0.4
500-600	0.42	0.41	0.43	0.42	0.41	0.39	0.37
600-700	0.58	0.46	0.42	0.39	0.42	0.38	0.4
700-800	0.39	0.38	0.41	0.41	0.42	0.43	0.43
800-900	0.43	0.39	0.36	0.35	0.35	0.36	0.35
900-100	0.51	0.46	0.46	0.44	0.44	0.43	0.42
Training Time	0.212	1.12	2.4	6.08	13.339	18.722	20.227
Elapsed time	0.085	0.078	0.078	0.094	0.309	0.107	0.082

Logistic Regression had very fast training, 20 seconds for 50k documents. This of course would increase as the document set increases, but it is not nearly as slow as other algorithms that take several minutes for the same number of documents. The accuracy, on the other hand, is not very good. It starts at a maximum of 60% and then decreases as the document set increase.

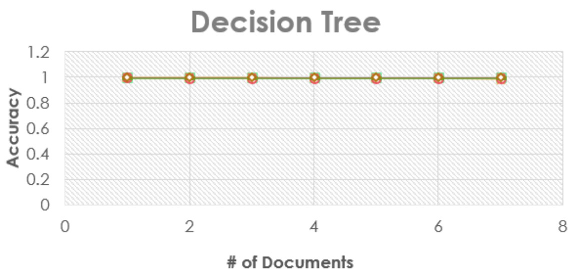
3.5 K Nearest Neighbors



K Nearest Neighbors	Accura cy						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	0.32	0.31	0.31	0.36	0.36	0.35	0.33
100-200	0.35	0.38	0.41	0.42	0.42	0.48	0.47
200-300	0.31	0.31	0.36	0.35	0.36	0.4	0.4
300-400	0.44	0.44	0.46	0.49	0.54	0.48	0.51
400-500	0.37	0.44	0.33	0.29	0.32	0.4	0.4
500-600	0.35	0.39	0.35	0.38	0.39	0.41	0.32
600-700	0.43	0.41	0.43	0.42	0.41	0.39	0.39
700-800	0.37	0.34	0.35	0.41	0.46	0.39	0.35
800-900	0.43	0.37	0.39	0.35	0.32	0.32	0.3
900-100	0.33	0.47	0.4	0.44	0.45	0.45	0.5
Training Time	0	0	0	0.016	0.019	0.031	0.031
Elapsed time	0.155	0.625	1.078	1.983	3.31	4.045	4.59

KNN has very fast training times. Some so fast that they appeared as zeros in the timing when captured. However, the accuracy is very low and does not show any signs of increasing as the document sets increase.

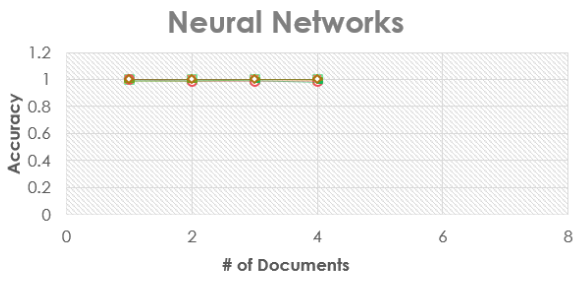
3.6 Decision Tree



Decision Tree	Accura cy						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	1	1	1	1	1	1	1
100-200	1	1	1	1	1	1	1
200-300	1	1	1	1	1	1	1
300-400	1	1	1	0.99	0.99	0.99	0.98
400-500	1	1	1	1	1	1	1
500-600	1	0.99	0.99	0.99	0.99	0.99	0.99
600-700	1	1	1	1	1	1	1
700-800	0.99	0.99	0.99	0.99	0.99	0.99	0.99
800-900	0.99	0.99	0.99	0.99	0.99	0.99	0.99
900-100	1	1	1	1	1	1	1
Training Time	0.806	7.974	23.993	56.48	99.828	159.71	198.07
Elapsed time	0.075	0.092	0.154	0.078	0.102	0.094	0.0899

Decision Tree was one of the best performing algorithms. This algorithm has very fast training speeds and extremely accurate results. It showed signs that the accuracy might decrease after a few more document sets are added however it is unlikely that it the accuracy would have changed for the worse. Out of the tested algorithms this was a top performer both in speed and accuracy.

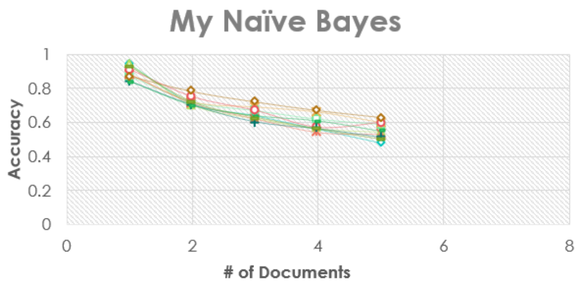
3.7 Neural Networks



Neural Networks	Accuracy						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	1	1	1	1			
100-200	1	1	1	1			
200-300	1	1	1	1			
300-400	1	1	1	0.99			
400-500	1	1	1	1			
500-600	1	0.99	0.99	0.99			
600-700	1	1	1	1			
700-800	0.99	0.99	0.99	0.98			
800-900	0.99	0.99	1	1			
900-100	1	1	1	1			
Training Time	56.315	435.316	1506.819	10349.03			
Elapsed time	0.026	0.105	0.25	0.15			

Neural Networks was also a top performer. The accuracy was extremely good. 99% on average. However, the training time was extremely slow. It took approximately 2.77 hours to train 20k documents. The rest of the document sets were not captured do to timing constraints. It is evident that the time would have only increased and would have to take several hours if a 100k document set was to be trained. This is a worthwhile algorithm since the accuracy is so good.

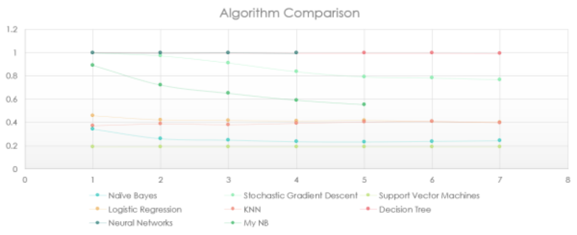
3.8 My Naïve Bayes



MyNB	Accuracy						
Training Set Size	1k	5k	10k	20k	30k	40k	50k
0-100	0.94	0.71	0.64	0.56	0.48		
100-200	0.87	0.72	0.67	0.62	0.59		
200-300	0.94	0.7	0.62	0.56	0.54		
300-400	0.88	0.72	0.69	0.66	0.6		
400-500	0.88	0.71	0.62	0.54	0.53		
500-600	0.91	0.75	0.67	0.57	0.6		
600-700	0.84	0.7	0.6	0.56	0.51		
700-800	0.84	0.7	0.64	0.61	0.55		
800-900	0.93	0.72	0.63	0.56	0.5		
900-100	0.87	0.78	0.72	0.67	0.63		
Training Time	0.16	0.944	2.713	14.99	24.331		
Elaspsed time	132.059	877.13	2418.075	3427.908	41635.77		

The Naïve Bayes algorithm that was created had good initial performance but then dropped off as more documents where added. It also takes very long. It is not the training that causes this, it is the classifying computations. The training is relatively fast, 20 seconds for 30k documents. But it took 10 hours to classify just 1k documents. Approximately 1 hour per 100 tweets. The computation time was too long to continue this under the time constraints of this assignment. The accuracy started at high 90% and then dropped to 50% at 30k documents. It seemed that this is where the accuracy levels off.

4 Conclusion



Although this paper is about tweets and prediction of their emojis, this is mostly a survey of different classification algorithms and their performances. Some better at predicting emoji than others. Top performer include Decision Trees and Neural Networks and the rest where not up to the task.

5 Next Steps

A submission for the SemEval competition could is a definite possibility. As well as different measures for test validation. These include cross validation and a confusion matrix. To not just measure accuracy but also recall and other important measures used to determine the efficiency of machine learning algorithms of a specific system.