Capstone Project: Final Report
Twitter Sentiment Analysis with Deep Learning

By. Christopher Feliz

Table of Contents

Project Proposal + Problem Statement

Twitter is a platform where individuals post their thoughts and opinions as they see fit. But for some it can do more damage than good, such as having the ability to drop a company's stock price. Earlier in May, Tesla CEO Elon Musk sent out a tweet to his followers saying, “Tesla stock price is too high imo.” After this tweet was posted, Tesla's share price dropped more than 11% within the first couple of hours[1]. A negative tweet like that should be avoided at all cost for being detrimental to a company.

A few years ago a Twitter user tweeted at Wendy's asking for a year's worth of free nuggets in exchange for retweets. The Twitter user tweeted, "HELP ME PLEASE. A MAN NEEDS HIS NUGGS". Wendy's replied with a daunting '"18 million." This tweet went viral with a total of 3.4million retweets, taking third place of all time[2]. This tweet generated massive amounts of free marketing and good publicity for Wendys.

In this project I plan to build a model that can classify a tweet as containing positive, negative or neutral sentiments. This model can help companies understand the public's perception of them, both good and bad. As well as help brands who would like the benefits of free marketing from going viral on Twitter. The data will be collected from 'https://www.kaggle.com/c/tweet-sentiment-extraction', a data warehouse. The final deliverables for the project will be the code in a github repository alongside the final report and slide deck.

Data Collection and Wrangling

In the following section I will be explaining the preprocessing steps used for general machine learning. Shortly after I will be explaining the preprocessing steps needed for neural networks.

Humans are born with the ability to create and interpret words, machines on the other hand are not able to. The text must first go through various transformations before a machine is able to interpret it. Another challenge that is evident when dealing with internet text is the use of acronyms. Acronyms have become a part of our writing, especially in tweets due to their character limit. To improve this I scraped acronyms and their meanings from a website called "https://www.netlingo.com/acronyms.php", and made a dictionary. I then used a 'Lambda' function to iterate through each tweet and replace any acronyms with their full meaning. This will help add more data to our machine learning models.

[1] "Elon Musk tweet sends Tesla's stock plunging - The ...." 1 May. 2020, https://www.washingtonpost.com/technology/2020/05/01/musk-tesla-stock/. Accessed 11 Jul. 2020.
[2] "• Most retweeted posts of all time 2020 | Statista." 25 Feb. 2020, https://www.statista.com/statistics/699462/twitter-most-retweeted-posts-all-time/. Accessed 11 Jul. 2020.

My next step was cleaning and removing the noise from each tweet. For this I cleaned up my text data using regex, and removed all links, punctuation marks, and numbers. I also lower-cased all the data, to avoid having various meanings for the same word. After cleaning up the text, I tokenized each word and passed it through the Porter Stemmer. Stemming will help reduce each word to its root word, thus decreasing the number of words in our corpus. Decreasing the numbers of dimensions in a machine learning model is always preferred.

The final steps I took in transforming my tweets for machine learning was removing ‘stop words’, and numeric conversion. ‘Stop words’ are usually words that provide little or no value to a sentence. For example, words like ‘you’ , ‘I’, ‘they’, and ‘who'. To convert the tweets to numeric I used either CountVectorizer or TfidfVectorizer.

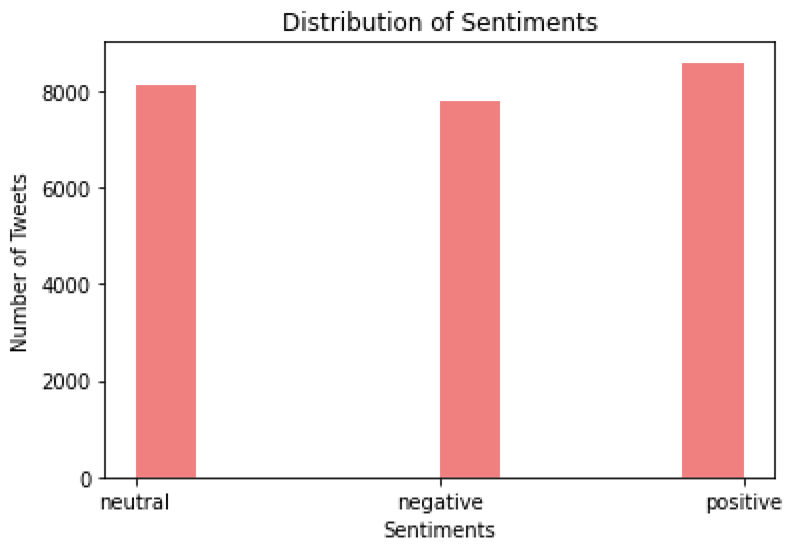
When preprocessing text for use in neural networks more information is always better. For these networks I did most of the preprocessing mentioned earlier, except stemming and 'stopword' removal. Text classification with deep learning differs from general machine learning, notably in preprocessing. In deep learning text data must first be transformed by an embedding layer. An embedding layer converts our text data to numeric, specifically a vector. In this project I created my own custom embedding layer using Keras. I first used the Tokenizer object in Keras to create a word-to-index dictionary, which maps each word to a special number.

Next, we had to deal with each tweet having different lengths. Data being passed into a neural network must have the same length, and this is where padding comes in. Padding is creating the same size vectors by either adding zeros to the beginning of a sentence or after to meet the set largest length. After converting our data into numeric values, it is finally ready to be used in machine learning.

Exploratory Data Analysis

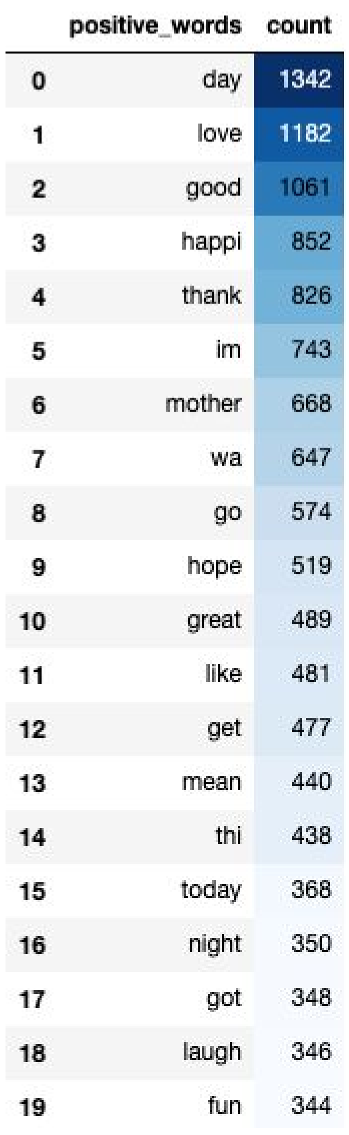
Bar plot of Target Values

In the chart below we can see that after downsampling our ‘neutral’ sentiment, most of the classes are within each other's range. Balancing our data is an important step because it will help avoid any added bias within our models.

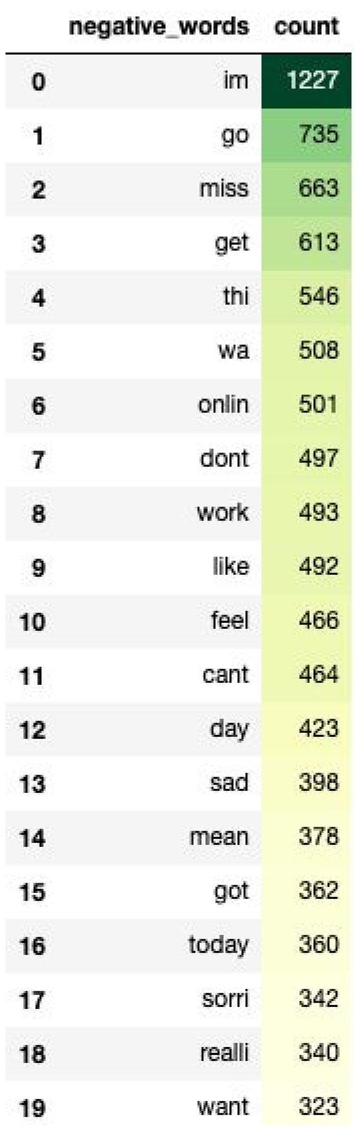


Most Common Words

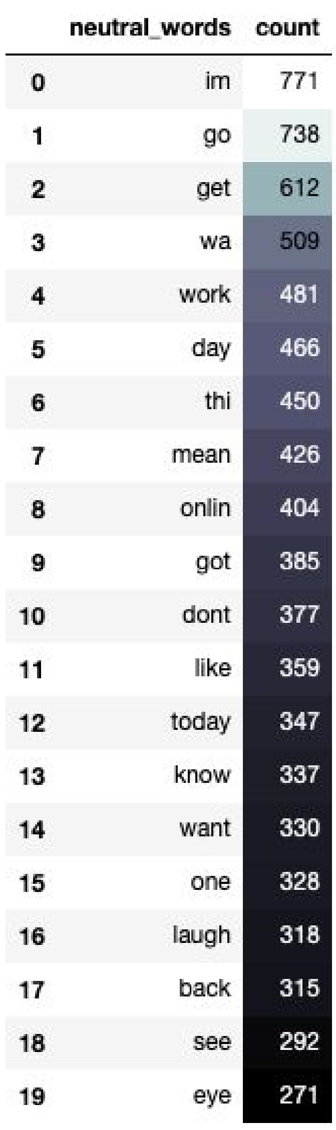
In the following tables we will be exploring the most common words for each sentiment.



	positive_words	count
0	day	1342
1	love	1182
2	good	1061
3	happi	852
4	thank	826
5	im	743
6	mother	668
7	wa	647
8	go	574
9	hope	519
10	great	489
11	like	481
12	get	477
13	mean	440
14	thi	438
15	today	368
16	night	350
17	got	348
18	laugh	346
19	fun	344



	negative_words	count
0	im	1227
1	go	735
2	miss	663
3	get	613
4	thi	546
5	wa	508
6	onlin	501
7	dont	497
8	work	493
9	like	492
10	feel	466
11	cant	464
12	day	423
13	sad	398
14	mean	378
15	got	362
16	today	360
17	sorri	342
18	realli	340
19	want	323

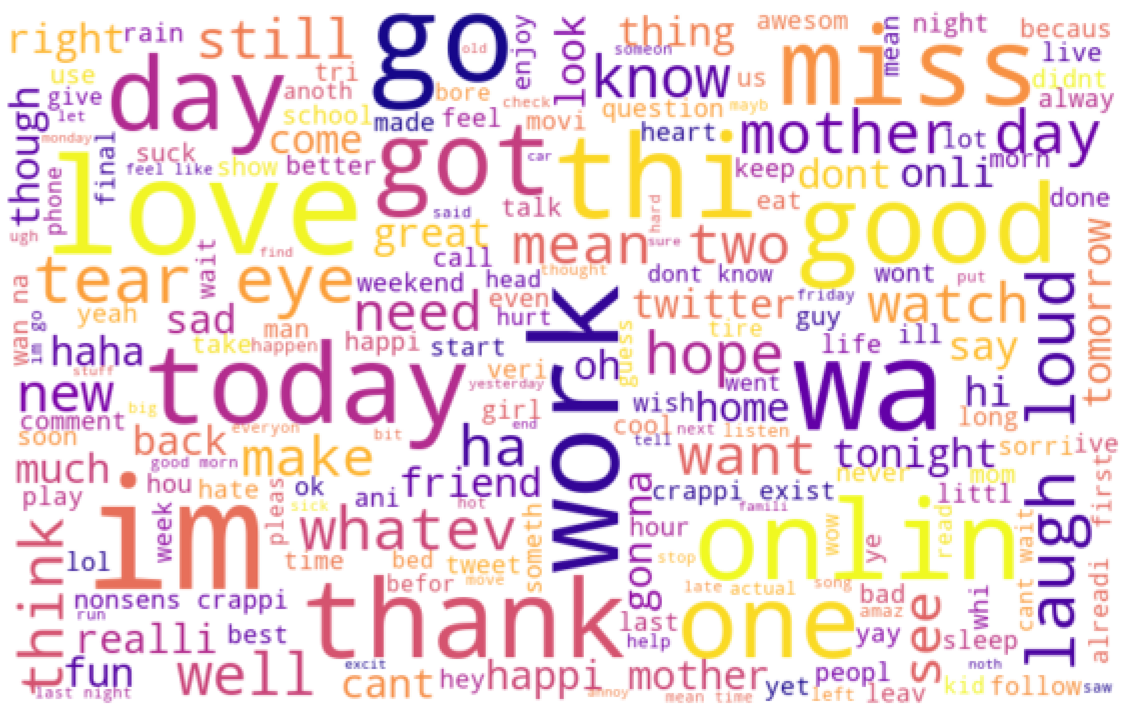


	neutral_words	count
0	im	771
1	go	738
2	get	612
3	wa	509
4	work	481
5	day	466
6	thi	450
7	mean	426
8	onlin	404
9	got	385
10	dont	377
11	like	359
12	today	347
13	know	337
14	want	330
15	one	328
16	laugh	318
17	back	315
18	see	292
19	eye	271

For the positive tweets we can see that most of the common words are love, good, happy, and great. For the negative tweets we can see that most of the common words are miss, cant, work, and mean. Finally, for the neutral tweets we can see that most of the common words are go, get, like, and know.

Word Clouds

Another way we can visualize the text in our corpus is using a word cloud.



Machine Learning: Preprocessing + Model Selection

In order to label a tweet's sentiment as either 'positive', 'negative', or 'neutral' we will use a multi-class classification algorithm. In this section I will be explaining the selection and performance of the general machine learning algorithms. In the following section will cover the neural networks used.

Before passing any text data into a machine learning model we must first transform it. To achieve this I used a 'Bag of Words' model known as CountVectorizer and the slightly different TFIDFVectorizer. The CountVectorizer converts our corpus into a dictionary with the word itself as the key, and its number of occurrences as the value. The TFIDFVectorizer focuses on the words that appear the least in relation to the number of documents. These are seen as significant words rather than a frequently occurring word.

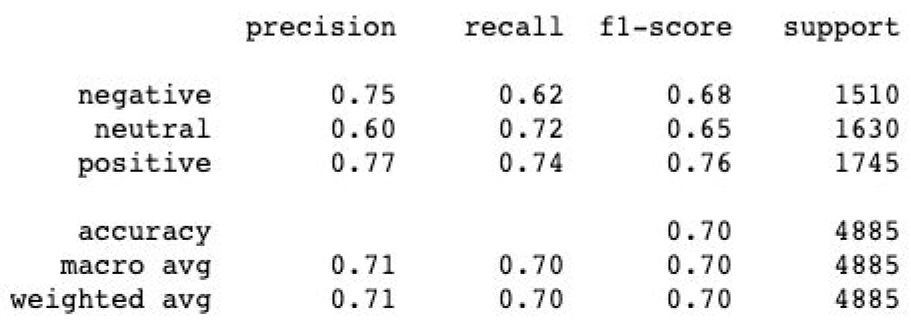
Both CountVectorizer and TFIDFVectorizer have adjustable parameters, the two I focused on are ‘ngram_range’ and ‘min_df’. The first ‘ngram_range’ refers to how big of a window size you would like to have in relation to co-occurring words. In this project I set this parameter to 1 to 3, which will take into account unigrams, bigrams, and trigrams. The next parameter, ‘min_df’ refers to the least number of times that a word should appear in your corpus for it to be a feature. I set this parameter to 5, which I believed worked best.

Machine Learning Models + Performance

In this section we will be exploring the best models and their performance. Please note that each model's performance was tested using both CountVectorizer and TFIDFVectorizer. We will be focusing on the models accuracy as well as F1 score for model selection.

The first classification model I built was using a DummyClassifier from scikit-learn. I used this model to get a baseline estimate for accuracy and F1 scores. The following models were then created, a Naive Bayes Classifier, Random Forest Classifier and a Support Vector Classifier. All the models mentioned previously had their hyperparameters tuned using grid search.

The best performing machine learning model was the Support Vector Classifier. The Support Vector Classifier achieved an accuracy of .70% and F1 score of .69%. The best hyperparameters found were ‘C’ set to 1, ‘gamma’ set to scale, and ‘kernel’ set to sigmoid. This model had similar measures between precision and recall throughout all 3 classes. After looking at the confusion matrix , it appears that the class it struggled most with was the ‘neutral’. This is interesting because from a human's viewpoint, determining whether someone's emotions are neutral is difficult.



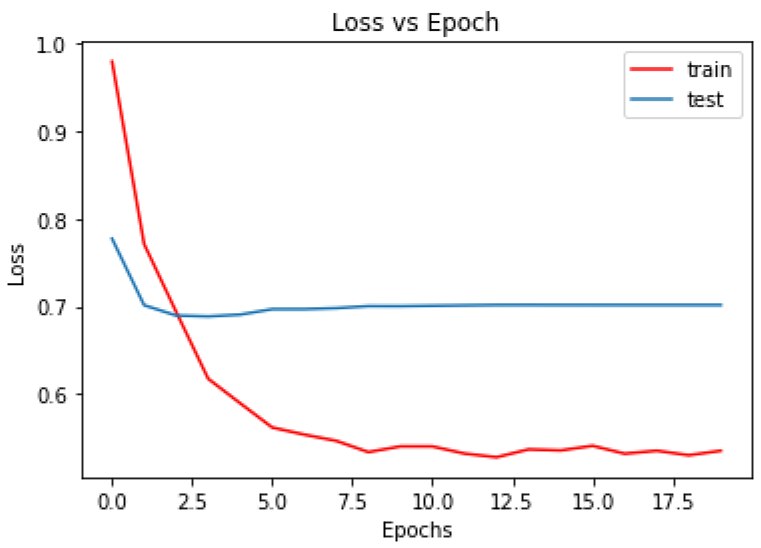
```
              precision    recall  f1-score   support

    negative       0.75      0.62      0.68      1510
     neutral       0.60      0.72      0.65      1630
    positive       0.77      0.74      0.76      1745

    accuracy                           0.70      4885
   macro avg       0.71      0.70      0.70      4885
weighted avg       0.71      0.70      0.70      4885
```

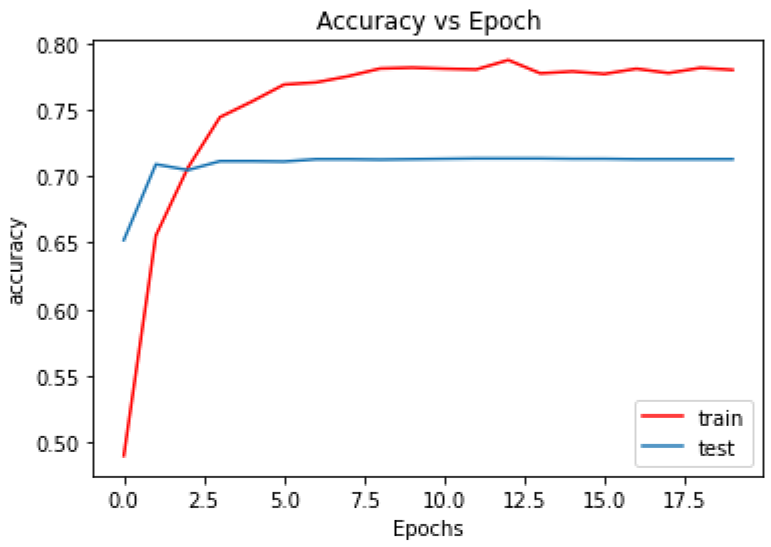
Deep Learning Models + Performance

In this section we will be exploring the best deep learning models as well as their performance. Prior to running our data set into a neural network it must again be transformed into its numeric form. For the transformation we will use a custom built embedding layer.

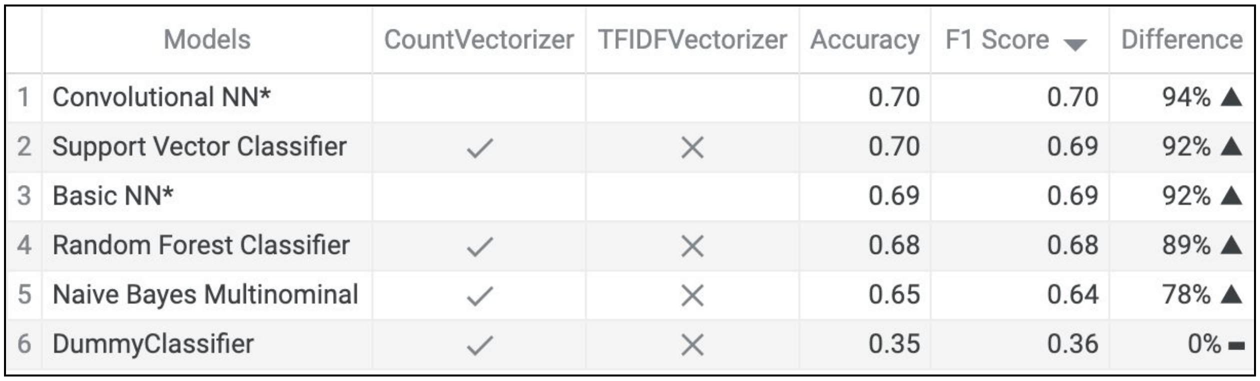
For the task of sentiment classification I will be using a basic neural network, recurrent neural network(LSTM) and convolutional neural network. The recurrent neural network is known to work well with sequential data because of its in-built memory system. Which helps the network take into account previous outputs. The convolutional neural network can be used in image and text classification. In this network we will see our documents as images, where each row of our matrix is a word vector.

The best performing neural network was the convolutional neural network. This network achieved an accuracy of .70 with an F1 score of .70. This model was prone to overfitting so I had to set a couple regularization parameters, such as dropout layers and a learning rate scheduler. This model used an 'adam' optimizer with a learning rate set to 0.04. We also used a one dimensional convolutional layer with 32 neurons. After looking at the confusion matrix, this model also tends to struggle with predicting a 'neutral' sentiment.


Loss vs Epoch
Loss
Epochs
train
test
1.0
0.9
0.8
0.7
0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Accuracy vs Epoch
accuracy
Epochs
train
test
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Machine Learning: Final Results



	Models	CountVectorizer	TFIDFVectorizer	Accuracy	F1 Score ▼	Difference
1	Convolutional NN*			0.70	0.70	94% ▲
2	Support Vector Classifier	✓	×	0.70	0.69	92% ▲
3	Basic NN*			0.69	0.69	92% ▲
4	Random Forest Classifier	✓	×	0.68	0.68	89% ▲
5	Naive Bayes Multinominal	✓	×	0.65	0.64	78% ▲
6	DummyClassifier	✓	×	0.35	0.36	0% ▬

Conclusion

After looking at the performances of the models it's no surprise that most of them struggled with predicting the ‘neutral’ class. It’s easier to predict ‘positive’ or ‘negative’ sentiments because they'll usually have a couple of words that lean towards one sentiment. This shouldn't be a big issue especially from a company's perspective. They would be more interested if people are having a positive or negative experience.

In regards to using CountVectorizer or TFIDF for data transformation there wasn't a significant difference in accuracy. The difference in model accuracy was in the range of about 1-2 % between models. From a general standpoint this happened to be the case with my dataset, with other datasets this difference could be better or worse.

Overall, I was able to get a 92% increase in the accuracy of our baseline model by using a support vector classifier and a convolutional neural network. For future iterations I plan on collecting more data, this will help in determining ‘neutral’ sentiments. We could also put in place pre-trained embedding layers such as word2vec or glove. These pre-trained embedding layers will help capture more of the relationships between words.