Project: Introduction to Machine Learning Applications

Section 1-3 Rough Draft

Due 11/8/2020

Vijay Yedidi

Professor Kuruzovich

**Executive Summary:**

Over the years Twitter has become a cornerstone of our lives. Over 330 million people use Twitter worldwide, and as we can see in this past US election cycle, the information posted on Twitter can be used to drastically sway public opinion. It’s hard to understand how these tweets can impact public opinion. Sometimes when these tweets go viral, they can drastically improve a person or company’s brand, or they can devastate the brands if the tweets are more negative in nature. In order to predict the impact viral tweets can have on the bottom line of a company, we need to thoroughly understand the connotation behind the tweets being put out, so if/when they go viral, we can create models that automatically predict whether they can positively or negatively impact the bottom line for the brands involved.

In order to do this, we need to understand the words or phrases Tweets contain that most significantly impact their sentiment, either positive, negative, or neutral, so we know what to look for when trying to analyze a Tweet and its impact on what it’s referring to. A machine learning algorithm can be employed to look at Tweets at a large scale to find the subtext that captures the connotation of the sentiment.

This problem was made into a competition by Kaggle, with Tweets and their corresponding “selected text”, or the phrases manually extracted by Figure Eight Data For Everyone [1], that most accurately signify the sentiment of the Tweet, which are to be used to train the model. The dataset also contains the sentiment of the Tweet, either positive, negative, or neutral. It is our job to train the model given the tweet and the sentiment to predict the subtext from the tweet that most accurately represents the sentiment.

Overall, 3 different methods were looked at in order to complete this model. The first solution involved no traditional machine learning methods; the data was split by sentiment and a weight was added to each individual word found in the tweet based on how often it was found in its respective sentiment. The weights of the words in each tweet were added up together

**Benchmarking of Other Solutions:**

The submission accuracy was calculated using the word level Jaccard Score[1]. This is a statistic that is used finding the similarity between two label sets. It’s found be dividing the size of the intersection by the size of the union of two label sets [2]. This is performed on the sentiments, so the more sentiments in common between the predicted and the actual sentiment, the higher Jaccard Score.

Zied Baklouti made a submission with a score of 0.52334 [3]. He used logistic regression to get this score. None of the hyperparameters were changed from the default variables, except for the regularization (C). This was set to 1500, which is meant to reduce the magnitude of the larger weight coefficients, which would reduce the noise picked up by the model. This score was relatively low compared to the top solution (0.736 being the highest), which is good to identify what happens when we try to train the model with limited hyperparameter tuning and almost no data cleaning. Some people make the argument that cleaning the tweets would make the training model more accurate, since we are removing useless words and punctuation that can add unnecessary noise to the final model (causing overfitting). But others claim that by cleaning the tweets up, we are thereby eliminating the meaning behind what people are saying and might reduce the effectiveness of the model.

Michal Taczynski submitted a solution with a score of 0.71603. He also didn’t clean the data before putting it in the training model. Michal used 10 splits, he set shuffle as True. This means the data is reshuffled before being split into batches. It is split into batches, and the model is trained using gradient descent via an AdamW optimizer via pytorch. This method of gradient descent takes into account the loss function of the previous step to optimize the learning rate, offering a 200% increase in model training speed [4]. He used a learning rate of 3 \* 10^-5, and betas of 0.9, 0.999 (non-negative weighting parameters). These are the standard recommended default values according to a Stanford University professor [5]. Unlike with traditional machine learning models, these tuning parameters shouldn’t impact the model outcome much at all, since they are constantly being corrected and optimized by the model itself with each iteration.

Nick Koprowicz submitted a solution with a score of 0.64664 [6]. His solution used only word counts with the CountVectorizer function in scikit learn to make his prediction. This basically tokenizes the words within the tweets, and creates dictionaries containing the words and their unique index. He then split the tweets into three subsets based on their sentiment. He then created an algorithm to assign a weight to each word inside the sentiment, based on how often they appeared in each sentiment. The algorithm’s logic can be seen in Figure 1. This model was interesting because it didn’t use any of the traditional machine learning algorithms many of the other authors used to solve this problem. This model was generated from scratch, based completely on the author’s own intuition. He realized that specific key words may be a lurking variable that could explain the sentiment of the tweet, and he utilized this theory to create his model. It generated a relatively high accuracy when created and proved that sometimes the elegant and simple solution can be more valuable than complex traditional machine learning models.

**Data Description and Initial Processing:**

Some authors defended that by cleaning the data, we are effectively removing parts of the speech that may affect the sentiment, while other authors state that without data cleaning, we will be overfitting the model, making it less usable for outside data. In this case, two models are going to be created, with and without the data cleaning to see which one performs better. The data will be cleaned and put into another column. It was cleaned by turning all the characters to lowercase, cleaning up website labels, and taking out punctuations/special characters like “-, /, :”. The “\*” was initially taken out, but it was realized that “\*” marks were used to signify cursing, which can impact the sentiment analysis models. A package called “nltk” was used to remove stop words, which are words like “the” or “as”, which are generic words used throughout all tweets that would not impact the outcome of the final model. When looking for null values, it was found that one row had null values for the tweet, with a neutral sentiment attached to it. This data point can be considered insignificant, especially in a training size of 27481, and because the corresponding sentiment was neutral, which is the least important of the sentiments we are trying to predict.

The data was also split into a positive, negative, and neutral tweets. The number of tweets for each of the sentiments was plotted in Figure 2. As can be seen in both the testing and training sets, the tweets with the neutral sentiment are much more common than with positive and negative. There seems to be marginally more positive tweets than negative tweets. In general, neutral tweets might not impact the bottom line of brands and companies. It’s going to be the positive and negative sentiments that are going to sway public opinions.

The next plot we want to create is with a breakdown of the word counts for the positive, negative, neutral, and total tweets. These results can be seen in Figure 3. These results were interesting; it was obvious that mostly positive words, like mostly “good”, “happy”, “love”, “thanks”, etc. were in the positive tweets. It was cool to see that many different variations of the word “mom” in the positive sentiment. Words like “miss”, “sad”, “sorry”, “bad”, and “hate”, which are negative in nature mostly show up in the negative tweets, which is to be expected.

There were words that didn’t really make much sense in the positive and negative sentiments. “Oh”, “got”, “like”, “would”, and similar words didn’t have an obvious reason to be in the positive sentiment. Words like “know”, “much”, “go”, and similar words didn’t have an obvious reason to be in the negative sentiment.

One interesting take away was the counts of the most popular words. The distribution shows that the most popular words in negative sentiments appear much less common than the most popular words in positive and neutral sentiments. The most popular word “miss” appeared 358 times in the negative tweets, while the most popular word in the positive sentiment, “good”, appeared 826 times, and the most popular word in the neutral sentiment, “get”, appeared 612 times. A further analysis, as shown in Figure 4, shows that the mean of the frequency distribution for unique words proves that the negative sentiments are repeated much less than any of the other sentiments (102.84). The positive frequency distribution has a slightly higher mean (162.64). The neutral and the total sentiment distribution mean was higher than the positive and the negative but can be mostly due to the neutral and total data sets having more tweets.

Another interesting thing is to see whether or not there is a correlation between the length of the tweets (character count) and the sentiment. The character count was done on the uncleaned data as well as the cleaned data. The results can be seen in Figure 5. It is interesting to see there is almost no difference in tweet lengths between the number of characters and the sentiment of the tweet. They are almost exactly the same between all the different sentiments.

Figure 6 shows the word distribution between the sentiments, and no discernable difference can be found between the sentiments. They all seem to be skewed right, probably because tweets are likely to be extremely short because it’s meant to be a platform for casual interaction, not for members to write paragraphs. It can be noted that by skewing the data, it becomes closer to a normal distribution as opposed to a bimodal distribution. This might be attributed to the fact that punctuation is used very often, probably more often than most words, and this is what produces a second peak at the higher end of the distribution.

**Sources:**

1. <https://www.figure-eight.com/data-for-everyone/>
2. <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.jaccard_score.html>
3. <https://www.kaggle.com/ziedbaklouti/bow-logistic-regression-sklearn>
4. <https://www.fast.ai/2018/07/02/adam-weight-decay/>].
5. http://cs231n.github.io/neural-networks-3/#ada
6. https://www.kaggle.com/nkoprowicz/a-simple-solution-using-only-word-counts

**Appendix I: Graphs and Tables**

**Text

Description automatically generated**

Figure 1: The algorithm for the weight distribution for words that Nick Koprowicz used to create his prediction.

Chart, bar chart

Description automatically generated

Figure 2: A bar graph to break down how many tweets are found by Sentiment. The graph on the left is the trained model, the right is the test model.

Figure 3: The distribution of word frequencies by Sentiment.

Chart, bar chart

Description automatically generated

Figure 4: A bar plot of the means of the word distribution by sentiment.

Chart, bar chart

Description automatically generated

Figure 5: Bar plots of the means of the number of characters for each tweet based on sentiment.

Chart, funnel chart

Description automatically generated

Figure 6: Histogram distribution for the word counts for each sentiment. The top row is preprocessed data, and the bottom row is processed.

**Appendix 2: Raw Code: Code can be found at https://github.com/yedidv/MachineLearningMidtermFinal**

import pandas as pd

def ReadCsv(data\_path):

    data = pd.read\_csv(data\_path)

    data.text = data.text.astype(str)

    if data\_path == 'train.csv':

        data.text = data.selected\_text.astype(str)

    return data

train = ReadCsv('train.csv')

train = ReadCsv('test.csv')

## We want to see if we have any null variables in the test or the train dataset.

print('\n train \n ')

print(train.isna().sum())

print('\n test \n ')

print(test.isna().sum())

## The train has one row with null variables. Unfortunately, because these are strings, these is no way we can assign a value to these null values without impacting our final model. So we are going to drop this one row.

train.dropna(inplace = True)

## Now we can see that there are no null variables.

print('\n train \n ')

print(train.isna().sum())

print('\n test \n ')

print(test.isna().sum())

## We want to clean the tweets up. We want to clean up website addresses, punctuation, and stop words.

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

import re

def RemoveStopWords(text):

    ## Remove the stop words using the nltk package.

    text\_tokens = word\_tokenize(text)

    tokens\_without\_sw = [word for word in text\_tokens if not word in stopwords.words()]

    return ' '.join(tokens\_without\_sw )

def CleanText(text):

    ## Clean up the tweets (punctuation, websites) using the re package

    text = str(text).lower()

    text = re.sub('https?://\S+|www\.\S+', '', text)

    text = re.sub('https:', '', text)

    text = re.sub("[-'.,)(#<!:/?]", '', text)

    text = RemoveStopWords(text)

    return text

train['clean\_text'] = train.text.apply(lambda x: CleanText(x))

test['clean\_text'] = test.text.apply(lambda x: CleanText(x) )

train.head()

## We want the length of the tweets for both the cleaned and preprocessed data.

train['number\_characters'] = train.text.str.len()

train['number\_clean\_characters'] = train.clean\_text.str.len()

train['number\_characters'] = test.text.str.len()

train['number\_clean\_characters'] = test.clean\_text.str.len()

train.head()

## We want to output the data to csv because processing the data took 30 minutes.

train.to\_csv('cleaned\_train.csv', index = True)

test.to\_csv('cleaned\_test.csv', index = True)

def GroupBySentiment(data):

    ## We want to split the data by sentiment.

    text\_sentiment\_count = data.groupby(['sentiment'], as\_index = False).text.count()

    positive\_text = data[data.sentiment == 'positive']

    negative\_text = data[data.sentiment == 'negative']

    neutral\_text = data[data.sentiment == 'neutral']

    return text\_sentiment\_count, positive\_text, negative\_text, neutral\_text

train\_text\_sentiment\_count, train\_positive\_text, train\_negative\_text, train\_neutral\_text = GroupBySentiment(train)

test\_text\_sentiment\_count, test\_positive\_text, test\_negative\_text, test\_neutral\_text = GroupBySentiment(test)

def PlotSentimentCount(data, axis, title):

## Tweet Counts by Sentiment

    sns.barplot(x = 'sentiment', y = 'text', data = data, ax = ax[axis])

    ax[axis].set\_xlabel('Sentiment')

    ax[axis].set\_ylabel('Text Count')

    ax[axis].set\_title(title)

import seaborn as sns

import matplotlib.pyplot as plt

sns.set()

fig, ax = plt.subplots(1, 2, constrained\_layout = True )

fig.suptitle('How Many Tweets are in Each Sentiment?')

PlotSentimentCount(train\_text\_sentiment\_count, 0, 'Train')

PlotSentimentCount(test\_text\_sentiment\_count, 1, 'Test')

mpy as np

import collections

def Words(data):

    ## Word Counts

    words = data.clean\_text.apply(lambda x: str(x).split())

    words = collections.Counter([item for sublist in words for item in sublist])

    words = pd.DataFrame(words, index = [0]).T

    words.columns = ['Count']

    words = words.sort\_values(by = 'Count', ascending = False)

    words = words[(words.index != '\*')]

    words = words[(words.index != "`")]

    return words.head(50)

train\_neutral\_words = Words(train\_neutral\_text)

train\_positive\_words = Words(train\_positive\_text)

train\_negative\_words = Words(train\_negative\_text)

train\_words = Words(train)

fig, ax = plt.subplots(4,1, figsize = (15, 40))

fig.suptitle('Word Counts')

def PopularWords(data, axis, title):

    ## Plot the Word Counts

    sns.barplot(x = data.Count, y = data.index, ax = ax[axis])

    ax[axis].set\_title(title)

PopularWords(train\_words, 0, 'Total')

PopularWords(train\_positive\_words, 1, 'Positive Sentiment')

PopularWords(train\_negative\_words, 2, 'Negative Sentiment')

PopularWords(train\_neutral\_words, 3, 'Neutral Sentiment')

fig.savefig('WordCounts.jpg')

## Word Distribution Descriptive Stats

words\_distribution\_description = train\_positive\_words.describe().T

words\_distribution\_description = words\_distribution\_description.append(train\_negative\_words.describe().T)

words\_distribution\_description = words\_distribution\_description.append(train\_neutral\_words.describe().T)

words\_distribution\_description = words\_distribution\_description.append(train\_words.describe().T)

words\_distribution\_description.index = ['Positive', 'Negative', 'Neutral', 'Train']

words\_distribution\_description['Sentiment'] = words\_distribution\_description.index

words\_distribution\_description

## Bar plot of the mean of the word count by sentiment.

sns.barplot(x = 'Sentiment', y = 'mean', data = words\_distribution\_description)

## Calculate the means of the number of characters of tweets based by sentiment

positive\_means = [train\_positive\_text.number\_characters.mean(), train\_positive\_text.number\_clean\_characters.mean()]

negative\_means = [train\_negative\_text.number\_characters.mean(), train\_negative\_text.number\_clean\_characters.mean()]

neutral\_means = [train\_neutral\_text.number\_characters.mean(), train\_neutral\_text.number\_clean\_characters.mean()]

train\_means = [train.number\_characters.mean(), train.number\_clean\_characters.mean()]

means = {'Positive Mean': positive\_means,

            'Negative Mean': negative\_means,

            'Neutral Mean': neutral\_means,

            'Total Mean': train\_means}

means = pd.DataFrame(means).transpose()

means.columns = ['NumberCharacters', 'NumberCleanCharacters']

means

fig, ax = plt.subplots(1,2, figsize = (15, 10) )

def PlotMeans(data, clean\_or\_dirty, axis):

    ## Means of the character count

    sns.barplot(x = data.index, y = data[clean\_or\_dirty], ax = ax[axis] ).set\_title(clean\_or\_dirty)

j = 0

for column in means.columns:

    PlotMeans(means, column, j)

    j+=1

train.number\_characters.describe()

## Distribution of the Word Counts for each sentiment

fig, ax = plt.subplots(2, 4, figsize = (15, 15))

plt.suptitle('Distribution of The Word Counts For Each Sentiment')

sns.distplot(train\_positive\_text.number\_characters, ax = ax[0,0]).set\_title('Positive Uncleaned')

sns.distplot(train\_negative\_text.number\_characters, ax = ax[0,1]).set\_title('Negative Uncleaned')

sns.distplot(train\_neutral\_text.number\_characters, ax = ax[0,2]).set\_title('Neutral Uncleaned')

sns.distplot(train.number\_characters, ax = ax[0,3]).set\_title('Train Uncleaned')

sns.distplot(train\_positive\_text.number\_clean\_characters, ax = ax[1,0]).set\_title('Positive Cleaned')

sns.distplot(train\_negative\_text.number\_clean\_characters, ax = ax[1,1]).set\_title('Negative Cleaned')

sns.distplot(train\_neutral\_text.number\_clean\_characters, ax = ax[1,2]).set\_title('Neutral Cleaned')

sns.distplot(train.number\_clean\_characters, ax = ax[1,3]).set\_title('Train Uncleaned')

**Chart, bar chart

Description automatically generated**

Word Count for Negative Tweets