Twitter Sentiment Analysis

IST_736 Final Project Report

17th June 2022

Suihin Wong, Morgan Gere, Niranjan Juvekar

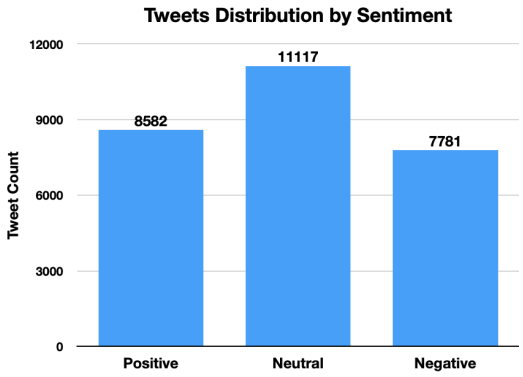
Introduction

Twitter is a social media network platform where people can communicate with each other in messages like tweets. The goal of the report is to create a high end machine learning model that predicts positive, neutral or negative sentiment for tweets for a company. There are a variety of uses of the model such as helping companies to see how the market reacts to their product, so the product team can have a better understanding on how to improve the products. In this report, we will cover exploratory data analysis to understand the data, pre-processing the raw data, and build the model with different methods to compare the performance with an online sentiment analysis tool.

Exploratory Data Analysis:

Data Set

The data was obtained from kaggle from this link:
https://www.kaggle.com/datasets/yasserh/twitter-tweets-sentiment-dataset
The dataset has 27,481 tweets labeled according to their sentiment.



The data is unbalanced with significantly more number of neutral tweets than the positive and negative.

Word Clouds

To have more insight about the tweets, we created three word clouds, for all tweets, positive tweets, and negative tweets. The imported data was divided using .loc to identify the tweets that had what labels. The tweets were then joined together into a string. All of the three word clouds showed the most top frequency words with 2000 max. Moreover, it would provide a deeper understanding on how we can pre-process the data and what the tweets are about in a high

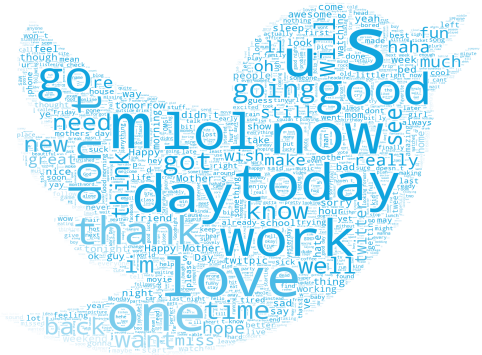
level view. To create the word clouds, we used twitter logo to shape the word cloud and removed the stopword with a default list of english words from wordcloud library. An example of the code is provided below.

```
#collecting all positive labeled tweets
positive_df = df.loc[df['sentiment'] == 'positive']
# print(positive_df)
```

```
# placing all positive and negative tweets into a string
positiveText = ' '.join(positive_df["text"])
negativeText = ' '.join(negative_df["text"])
allText      = ' '.join(df["text"])
```

```
#creating a word cloud in the shape of a green twitter bird
mask = np.array(Image.open(r'C:\Users\Morga\programsMG\TextMining\TwitterGreen.png'))
mask_colors = ImageColorGenerator(mask)
wc = WordCloud(stopwords=STOPWORDS,
               mask=mask, background_color="white",
               max_words=2000, max_font_size=256,
               random_state=42, width=mask.shape[1],
               height=mask.shape[0], color_func=mask_colors)
cloud = wc.generate(positiveText)
plt.title('Wordcloud of Positive Tweets')
# plt.savefig('WordcloudPositiveTweets.png')
cloud.to_file('WordcloudPositiveTweets.png')
plt.imshow(wc, interpolation="bilinear")
plt.axis('off')
plt.show()
```

Word Cloud[1] (all tweets)



There are many words present that are ambiguous to sentiment; they will appear in all the word clouds. The character “m” comes from the word I’m and “s” comes from an apostrophe ‘s’ from words such as “It’s”, “that’s”, “what’s”, “let’s” etc. Some words such as “u” will need to be fixed in preprocessing and converted into “you”. The word clouds also shows internet slang.

Word Cloud[1] (positive tweets only)

This word cloud showed the word frequency count for positive tweets before preprocessing. This word cloud provided more information on what the top frequency words showed in positive tweets. From this positive tweets word cloud, it included some positive words such as love, thank, happy, nice, great, and good. It also included some unknown letters like s, m and u which was discussed previously..

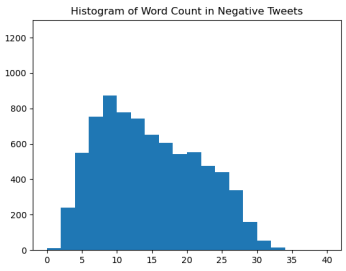
Word Cloud[1] (negative tweets only)

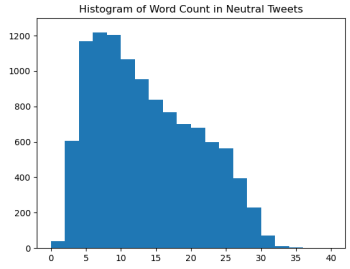
This word cloud showed the word frequency count for negative tweets before pre-processing. The negative tweets word cloud included words such as miss, work, sad, sorry and hate. It also included some words like now, today, day and one which can't tell if it is a negative word or not by just one single word. However, all of the word cloud gave a better understanding of the raw data and how we can implement the pre-processing.

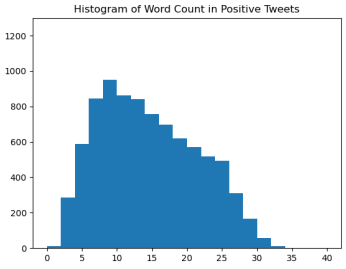
We also wanted to see if there is a pattern in terms of word lengths of positive, negative and neutral tweets.

```
plt.clf()
plt.hist(positive_df['WordCount'], bins=20, range=(0, 40))
plt.ylim(0, 1300)
plt.title('Histogram of Word Count in Positive Tweets')
plt.savefig("positive_df_WordCount.png")

plt.clf()
plt.hist(negative_df['WordCount'], bins=20, range=(0, 40))
plt.ylim(0, 1300)
plt.title('Histogram of Word Count in Negative Tweets')
plt.savefig("negative_df_WordCount.png")

plt.clf()
plt.hist(neutral_df['WordCount'], bins=20, range=(0, 40))
plt.ylim(0, 1300)
plt.title('Histogram of Word Count in Neutral Tweets')
plt.savefig("neutral_df_WordCount.png")
```

Following are the word frequencies of the negative neutral and positive tweets. X axis represents bins (number of words in each tweet) whereas Y axis represents number of tweets in the respective bin:







As you can see from these graphs, the distribution of number of words per tweet for each category is similar. The neutral tweets tend to be longer than the positive and negative.

Method

Input

The preprocessed csv file was uploaded using pandas directly into a data frame.

```
import pandas as pd
# Importing the data into a pandas data frame.
df = pd.read_csv(r'C:\Users\Morga\programsMG\TextMining\Tweets.csv')
```

Sentiment

To understand how well the created models are performing three accuracies were obtained. The first was the raw agreement where if all predictions were placed in one category how well would the model predict. The highest number of one category is neutral, found to be 11,117 this made up ~40.45%. Since there are 3 possible categories the baseline is ~33.33%.

To have a higher standard to hold the model created to an out of the box sentiment analyzer was obtained through NLTK. Vader gives a positive, negative, neutral, and compound score. The compound score was used to divide the tweets into positive, negative and neutral tweets. The compound score ranges from -1 to 1 so any tweet that had received a compound score less than or equal to -0.4 was predicted as negative, any tweet that scored above or equal to 0.4 was predicted as positive, and the remaining tweets were predicted as neutral. The predictions obtained were placed into an accuracy score against the provided gold standard labels. This accuracy was found to be ~63.90%.

```
# Finding the raw agreement vs baseline
df['sentiment'].value_counts(normalize=True)
```

```
neutral     0.404549
positive    0.312300
negative    0.283151
Name: sentiment, dtype: float64
```

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

#selecting sentiment analyzer
sid = SentimentIntensityAnalyzer()
vader_predict=[]

#Dividing the tweets according to compond score
for tweet in M:
    if (sid.polarity_scores((tweet))['compound']) >= 0.4:
        vader_predict.append('positive')

    elif (sid.polarity_scores((tweet))['compound']) <= -0.4:
        vader_predict.append('negative')

    else:
        vader_predict.append('neutral')
# finding the accuracy of vader vs gold standard
from sklearn.metrics import accuracy_score
print(accuracy_score(y, vader_predict))
```

```
0.6390465793304221
```

Preprocessing

The data was checked for null values. One text was found to be empty but still contained a neutral sentiment. This was discarded using its index number.

```
#checking null values
print('sentiment null:',df['sentiment'].isnull().sum())
print('text null:',df['text'].isnull().sum())
```

```
sentiment null: 0
text null: 1
```

```
#Finding the null index number
x= df['text'].isnull()
i=0
for item in x:
    if item is False:
        i=i+1
    if item is True:
        break
print(i)
```

```
314
```

```
# Dropping the null value
df.drop([314], inplace = True)
```

The data was split into text and label (sentiment) and placed into lists.

```
#seperating the label
#placing the data into lists
y=df['sentiment'].values
M=df['text'].values
```

The tweets were manually inspected to have an idea of what preprocessing was required to be done. There were many words with multiple repetition of characters that amounted to improper spelling. An example would be “whyyyyyy”. To reduce the multiple repeats, a regular expression was used to replace the word with multiple repeats, with the same word containing only two repeats of the character. So to “whyy” for the given example.

```
import re
#replacing words with repeated characters
X1=[]
for item in M:
    X1.append(re.sub(r'([a-zA-Z0-9_<>-])\1+', r'\1\1', item))
```

There were many words that were slang or misspelled. These were replaced in the string using .replace. This was done to make the actual token count be a real representation of the number of times a word was used. This greatly condensed the total vocabulary.

```
# using replace to change slang and misspellings
tempX=[]
for item in X1:
    item1=item.lower()
    item1=item1.replace('2day ','today ')
    item2=item1.replace('2moro ','tomorrow ')
    item3=item2.replace('2morrow ','tomorrow ')
    item4=item3.replace('2night ','tonight ')
    item5=item4.replace('2nite ','tonight ')
    item6=item5.replace('b-day ','birthday ')
    item7=item6.replace('b4 ','before ')
    item8=item7.replace('bb ','be back ')
    item9=item8.replace('bbl ','be back later ')
    item10=item9.replace('bc ','because ')
    item11=item10.replace('bday ','birthday ')
    item12=item11.replace('belive ','believe ')
    item13=item12.replace('bf ','boyfriend ')
    item14=item13.replace('bff ','best freind forever ')
    item15=item14.replace('brb ','be right back ')
    item16=item15.replace('bros ','bro ')
    item17=item16.replace('bs ','bullshit ')
    item18=item17.replace('btw ','by the way ')
    item19=item18.replace('dat ','that ')
    item20=item19.replace('doc ','doctor ')
    item21=item20.replace('docs ','doctor ')
    item22=item21.replace('hott ','hot ')
    item23=item22.replace('fb ','facebook ')
    item24=item23.replace('jk ','just kidding ')
    item25=item24.replace('jst ','just ')
    item26=item25.replace(' ng ',' nice game ')
    item27=item26.replace('nt ','nice try ')
    item28=item27.replace('ok ','okay ')
    item29=item28.replace('okayy ','okay ')
    item30=item29.replace('omgg ','omg ')
    item31=item30.replace('ppl ','people ')
    item32=item31.replace('tonite ','tonight ')
    item33=item32.replace(' u ',' you ')
    item34=item33.replace('u2 ','you too ')
    item35=item34.replace('ugg ','ugh ')
    item36=item35.replace('uggh ','ugh ')
    item37=item36.replace('uh ','ugh ')
    item38=item37.replace('uhh ','ugh ')
    item39=item38.replace('umm ','um ')
    item40=item39.replace('ur ','your ')
    item41=item40.replace('waah ','waa ')
    item42=item41.replace('wah ','waa ')
    item43=item42.replace('waay ','waa ')
    item44=item43.replace('xoxo ','xo ')
    item45=item44.replace('xx ','xo ')
    item46=item45.replace(' y ',' why ')
    item47=item46.replace('whyy ','why ')
    item48=item47.replace('yaay ','ya ')
    item49=item48.replace('yah ','ya ')
    item50=item49.replace('juss ','just ')
    item51=item50.replace('whassqoodd ','whats good')
    item52=item51.replace('soo ','so ')
    tempX.append(item52)
```

The data set obtained had most hashtags removed. There were found to be a few remaining that were added to a list. Other specific words found in the data that need to be removed were also placed in this list along with unwanted punctuation. This list will be applied during vectorization.

```
#selecting the stop words
stopwords =['!','#','$','%','&',"'",'*','(',')','+',',','-','.','/',':',';','<','=','>'
            ,'?','@','[','\\',']','^','_','`','{','|','}','~','¿','#bgt','#fail','#fb'
            ,'#ff','#followfriday','#hhrs','#judday','#sanctuarysunday','#starwarsday'
            ,'#twpp','>','_','_127','__','_b','_benson','_c','_carter','_d','_guy','_henrie'
            ,'_j','_m','_marie','_skies','_x','_xo']
```

Vectorization

The stop list created during preprocessing was joined to the skearn's stop words list using .union.

```
#Adding the created stopwords to sklearn stopword list.
from sklearn.feature_extraction import text
stop_words = text.ENGLISH_STOP_WORDS.union(stopwords)
```

A tweet tokenizer and the porter stemmer are obtained from NLTK and two functions were created that allowed each to run inside of sklearns vectorization.

```
# import TweetTokenizer() method from nltk
from nltk.tokenize import TweetTokenizer
tk = TweetTokenizer()
```

```
# create a function for the tweet tokenizer from NLTK
def tok(text):
    tt = TweetTokenizer()
    return tt.tokenize(text)
```

```
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
# selecting porter stemmer and placing it in a function
# for use in sklearn's vectorizor
stemmer = PorterStemmer()
analyzer = TfidfVectorizer().build_analyzer()

def stemmed_words(doc):
    return (stemmer.stem(w) for w in analyzer(doc))
```

To test the vectorization options, eight different vectors were created. Each was then applied to a Support vector machine (svm) and a Multinomial Naive Bayes (mnb) model. A 10-fold cross validation was used to obtain Accuracy and the F1-Score for each label (postive, negative, and neutral). These were then compared to find the best vectorization option for each algorithm.

All vectorizations used term frequency inverse document frequency (tf-idf), minimum document frequency of 5 and the stop words list created during preprocessing. The first two vectors were created to compare the regular sklearn tokenizer vs. the NLTK tweet tokenizer. The 3rd vectorization used the sklearn tokenizer with additional stop words from sklearn and the 4th did the same with the NLTK tweet tokenizer. The remaining four vectorization used the NLTK tweet tokenizer. The 5th vector is comprised of the added sklearn stop words and the porter stemmer and the 6th is porter steming without additional stop words. The 7th vectorization is adding bigrams without the porter stemmer and no additional stop words. Finally the 8th vector is adding bigrams with stemming and no additional stop words. The text obtained after preprocessing was fit and transformed using these different options.

```
# creating multiple vecorization options for testing using tfidf vectorizer, set minimum document frequency to 5
# and using stopwords or stopwords with sklearns stopwords
unigram_tfidf_vectorizer = TfidfVectorizer(encoding='latin-1', use_idf=True, min_df=5, stop_words=stop_words)
unigram_tfidf_vectorizer_Tweet_toke = TfidfVectorizer(encoding='latin-1', use_idf=True, min_df=5
                                                      ,stop_words=stop_words,tokenizer=tok)
unigram_tfidf_vectorizer_no_stop = TfidfVectorizer(encoding='latin-1', use_idf=True, min_df=5, stop_words=stopwords)
unigram_tfidf_vectorizer_Tweet_toke_no_stop = TfidfVectorizer(encoding='latin-1', use_idf=True
                                                              , min_df=5,stop_words=stopwords,tokenizer=tok)
unigram_tfidf_vectorizer_Tweet_toke_no_stop_stem = TfidfVectorizer(encoding='latin-1', use_idf=True
                                                                   , min_df=5,stop_words=stopwords
                                                                   ,tokenizer=tok,analyzer=stemmed_words)
unigram_tfidf_vectorizer_Tweet_toke_stem = TfidfVectorizer(encoding='latin-1', use_idf=True, min_df=5
                                                           ,stop_words=stopwords,tokenizer=tok,analyzer=stemmed_words)
ngram_tfidf_vectorizer_Tweet_toke_no_stop_ngram = TfidfVectorizer(encoding='latin-1', ngram_range=(1,2)
                                                                  , use_idf=True, min_df=5
                                                                  ,stop_words=stopwords,tokenizer=tok)
ngram_tfidf_vectorizer_Tweet_toke_stem_ngram = TfidfVectorizer(encoding='latin-1', ngram_range=(1,2)
                                                               , use_idf=True, min_df=5,stop_words=stopwords
                                                               ,tokenizer=tok,analyzer=stemmed_words)
```

```
# fitting and transforming the training data using the vectorizer
X_train_vec = unigram_tfidf_vectorizer.fit_transform(X)
```

```
# fitting and transforming the training data using the vectorizer
X_train_tweet = unigram_tfidf_vectorizer_Tweet_toke.fit_transform(X)
```

```
# fitting and transforming the training data using the vectorizer
X_train_vec_no_stop = unigram_tfidf_vectorizer_no_stop.fit_transform(X)
```

```
# fitting and transforming the training data using the vectorizer
X_train_tweet_nostop = unigram_tfidf_vectorizer_Tweet_toke_no_stop.fit_transform(X)
```

```
# fitting and transforming the training data using the vectorizer
X_train_tweet_no_stop_stem = unigram_tfidf_vectorizer_Tweet_toke_no_stop_stem.fit_transform(X)
```

```
# fitting and transforming the training data using the vectorizer
X_train_tweet_stem = unigram_tfidf_vectorizer_Tweet_toke_stem.fit_transform(X)
```

```
# fitting and transforming the training data using the vectorizer
X_train_tweet_nostop_ngram = ngram_tfidf_vectorizer_Tweet_toke_no_stop_ngram.fit_transform(X)
```

```
# fitting and transforming the training data using the vectorizer
X_train_tweet_stem_ngram = ngram_tfidf_vectorizer_Tweet_toke_stem_ngram.fit_transform(X)
```

SVM vs MNB

Eight models for each algorithm (mnb and svm) were initialized and each was run using a different training set created from the different vectorizations preformed. The accuracy and each F1-Score (positive, negative and neutral) were placed into separate dictionaries and graphed to visualize the differences in the results. Only one example of the code is shown below the rest are attached separately within a jupyter notebook (IST736_Final_Project)

```
# import the the models
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
# initialize the models
svm1 = LinearSVC(C=1)
svm2 = LinearSVC(C=1)
svm3 = LinearSVC(C=1)
svm4 = LinearSVC(C=1)
svm5 = LinearSVC(C=1)
svm6 = LinearSVC(C=1)
svm7 = LinearSVC(C=1)
svm  = LinearSVC(C=1)
mnb1 = MultinomialNB()
mnb2 = MultinomialNB()
mnb3 = MultinomialNB()
mnb4 = MultinomialNB()
mnb5 = MultinomialNB()
mnb6 = MultinomialNB()
mnb7 = MultinomialNB()
mnb  = MultinomialNB()
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
import numpy as np
```

```
# use the training data to train the model
svm_model_vec = svm1.fit(X_train_vec,y)
#cross validation score
svm_cv_scores_vec = cross_val_score(svm1, X_train_vec, y, cv=10)
svm_cv_neu_f_scores_vec = cross_val_score(svm1,X_train_vec, y,scoring=make_scorer(f1_score, average='weighted'
                                                                                  , labels=['neutral']),cv=10)
svm_cv_neg_f_scores_vec = cross_val_score(svm1,X_train_vec, y,scoring=make_scorer(f1_score, average='weighted'
                                                                                  , labels=['negative']),cv=10)
svm_cv_pos_f_scores_vec = cross_val_score(svm1,X_train_vec, y,scoring=make_scorer(f1_score, average='weighted'
                                                                                  , labels=['positive']),cv=10)
#finding the overall average metrics.
svm_cv_score_vec = round(np.mean(svm_cv_scores_vec),5)
svm_cv_neu_f_score_vec = round(np.mean(svm_cv_neu_f_scores_vec),5)
svm_cv_neg_f_score_vec = round(np.mean(svm_cv_neg_f_scores_vec),5)
svm_cv_pos_f_score_vec = round(np.mean(svm_cv_pos_f_scores_vec),5)
print('accuracy:',svm_cv_score_vec)
print('neutral f score:',svm_cv_neu_f_score_vec)
print('negative f score:',svm_cv_neg_f_score_vec)
print('positive f score:',svm_cv_pos_f_score_vec)
```

```
accuracy: 0.67911
neutral f score: 0.66086
negative f score: 0.64743
positive f score: 0.73202
```

```
# creating a dictionary of all the accuracys
svmmnbacc = {'svm_cv_score_vec':svm_cv_score_vec
             ,'svm_cv_score_tweet':svm_cv_score_tweet
             ,'svm_cv_score_vec_no_stop':svm_cv_score_vec_no_stop
             ,'svm_cv_score_tweet_no_stop':svm_cv_score_tweet_no_stop
             ,'svm_cv_score_tweet_no_stop_stem':svm_cv_score_tweet_no_stop_stem
             ,'svm_cv_score_tweet_stem':svm_cv_score_tweet_stem
             ,'svm_cv_score_tweet_nostop_ngram':svm_cv_score_tweet_nostop_ngram
             ,'svm_cv_score_tweet_stem_ngram':svm_cv_score_tweet_stem_ngram
             ,'mnb_cv_score_vec':mnb_cv_score_vec
             ,'mnb_cv_score_tweet':mnb_cv_score_tweet
             ,'mnb_cv_score_vec_no_stop':mnb_cv_score_vec_no_stop
             ,'mnb_cv_score_tweet_no_stop':mnb_cv_score_tweet_no_stop
             ,'mnb_cv_score_tweet_no_stop_stem':mnb_cv_score_tweet_no_stop_stem
             ,'mnb_cv_score_tweet_stem':mnb_cv_score_tweet_stem
             ,'mnb_cv_score_tweet_nostop_ngram':mnb_cv_score_tweet_nostop_ngram
             ,'mnb_cv_score_tweet_stem_ngram':mnb_cv_score_tweet_stem_ngram

}
```

```
import matplotlib.pyplot as plt

# function to add value labels
def addlabels(y):
    for i in range(len(y)):
        plt.text(i,y[i],y[i],fontweight = 'bold')


values = list(svmmnbacc.values())
names = list(svmmnbacc.keys())
colors = ['red','red','red','red','red','red','red','red'
          ,'blue','blue','blue','blue','blue','blue','blue','blue'
]
plt.figure(figsize=(60, 10))

plt.subplot(121)
plt.bar(names
        ,values
        ,color= colors
       )

plt.xticks(rotation= 45)
plt.ylabel('Percent')
plt.xlabel('Accuracy')
plt.title('SVM vs MNB accuracy')

addlabels(values)
plt.show()
```

Feature Set Engineering

For the feature engineering part, different feature sets were created to test out the performance of the models and see which features had the biggest impact on the model in terms of performance.

Vectorization and data frame creation

The previously best vectorization options for each model were used to create training data and that training vectors were placed into a pandas data frame. Both algorithms were used to find the 10-fold cross validation accuracy without any feature sets being added, and placed into a dictionary. To compare the slight changes in the output of the models only accuracy was used during this section.

```
# import the LinearSVC module
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
from sklearn.feature_extraction.text import TfidfVectorizer
# Best vectorization options for SVM
unigram_tfidf_vectorizer_Tweet_toke_no_stop_stem = TfidfVectorizer(encoding='latin-1'
                                                                   , use_idf=True
                                                                   , min_df=5
                                                                   ,tokenizer=tok
                                                                   ,analyzer=stemmed_words)
# Best vectorization options for MNB
ngram_tfidf_vectorizer_Tweet_toke_no_stop_ngram = TfidfVectorizer(encoding='latin-1'
                                                                  , ngram_range=(1,2)
                                                                  , use_idf=True
                                                                  , min_df=5
                                                                  , tokenizer=tok)
#vectorizing
vecs = unigram_tfidf_vectorizer_Tweet_toke_no_stop_stem.fit_transform(X)
# adding the vectors created into a data frame
vecsdf=pd.DataFrame(vecs.toarray(),
            columns=unigram_tfidf_vectorizer_Tweet_toke_no_stop_stem.get_feature_names_out())
#vectorizing
vecs2 = ngram_tfidf_vectorizer_Tweet_toke_no_stop_ngram.fit_transform(X)
# adding the vectors created into a data frame
vecs2df=pd.DataFrame(vecs2.toarray(),
            columns=ngram_tfidf_vectorizer_Tweet_toke_no_stop_ngram.get_feature_names_out())
```

```
#creating empty dict to add the accuracys into
svm_fs_accurracy={}
mnb_fs_accurracy={}
```

```
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
svm8 = LinearSVC(C=1)
mnb8= MultinomialNB()
import numpy as np
#train model
#svm_model_http = svm8.fit(vecs2df,y)
#cross validation score
svm_cv_scores = cross_val_score(svm8, vecsdf, y, cv=10)
mnb_cv_scores = cross_val_score(mnb8, vecsdf, y, cv=10)
#finding the overall average accuracy.
svm_cv_score = np.mean(svm_cv_scores)
mnb_cv_score = np.mean(mnb_cv_scores)
svm_fs_accurracy['Original svm Accuracy']=svm_cv_score
mnb_fs_accurracy['Original mnb Accuracy']=mnb_cv_score
```

The new features that had been engineered were added as a column. This data frame was then used to create a new model and using 10-fold cross validation the accuracy was obtained and placed into the same dictionary. The feature columns were then removed and new features were added and the process repeated for each feature set created. The dictionary of accuracies was then graphed for visual comparison. These were then used to find the best combination of features to be added in order to obtain the highest accuracy. The process of the model creation is the same as the above example and can be found in an attached juptyer notebook (IST736_Final_Project_New_Feature_Sets). Below is an explanation of how the features were engineered.

Http Count

Using the preprocessed untokenized text a loop was created that took each tweet and then took each item in the tweet that started with 'http' and added 1 to a count. This count was added to a list for each tweet.

```
http_count = []
for item in toktext:
    for word in item:
        count=0
        if word.startswith('http'):
            count = count+1
        else:
            continue
    http_count.append(count)
```

Emoticons Count

`Using the preprocessed untokenized text a loop was created that took each tweet and then each item in the tweet that started with an emiticon and added 1 to count. This count was added to a list for each tweet. This was repeated for each type of emoticon. Only one is shown below.

```
frown_count = []
for item in Xtemp:
    for token in item:
        count=0
        if token.startswith(':/'):
            count = count+1
        else:
            continue
    frown_count.append(count)
```

Sentences Count

The untokenized text was tokenized using the NLTK sentence tokenizer and a loop was used to take each tweet and count each token in it and place that count into a list.

```
from nltk import sent_tokenize
sentences=[]
for tweet in tempX:
    sentences.append(sent_tokenize(tweet))
```

```
sentences_count =[]
for tweet in sentences:
    count =0
    for sent in tweet:
        count=count+1
    sentences_count.append(count)
```

Word/Token Count

The untokenized text was tokenized using the NLTK tweet tokenizer and a loop was used to take each tweet and count each token in it and place that count into a list.

```
words=[]
for tweet in Xtemp:
    words.append(tweet)
```

```
word_count =[]
for tweet in words:
    count =0
    for word in tweet:
        count=count+1
    word_count.append(count)
```

Negation Count

The negation count presented in class for feature set engineering was used to create a count for each token that is not, no, never, or ends in “less”.

```
import re
def has_negation(post):
    pattern_neg_1 = re.compile(r'\b(not|no|never)\b')
    pattern_neg_2 = re.compile(r'\b([a-z]+less)\b')
    if pattern_neg_1.search(post.lower()) or pattern_neg_2.search(post.lower()):
        return 1
    else:
        return 0
```

```
neg_count=[]
for item in X:
    neg_count.append(has_negation(item))
```

But this requires more fine tuning because while this catches words such as “shameless”, “hopeless”, “useless”, it also catches words such as “wireless”, “priceless”, “nonetheless”, “harmless” which clearly was not intended to be caught as negative words.

Combinations

The highest two feature sets were added to the vector and a new model was created. (negation and word count).

Graph

```
import matplotlib.pyplot as plt

# function to add value labels
def addlabels(y):
    for i in range(len(y)):
        plt.text(i,y[i],y[i],fontweight = 'bold')


values = list(mnb_fs_accurracy.values())
names = list(mnb_fs_accurracy.keys())
colors = ['red','red','red','red','red','red','red','red'

]
plt.figure(figsize=(60, 10))

plt.subplot(121)
plt.bar(names
        ,values
        ,color= colors
       )

plt.xticks(rotation= 45)
plt.ylabel('Percent')
plt.xlabel('Feature Set')
plt.title('MNB Feature Set Accuracy')

addlabels(values)
plt.show()
```

Gridsearchcv Hyperparameter Tuning

The SVM model with the best vectorization options along with the best combination of engineered feature sets were passed into gridsearchcv to tune the hyperparameters. This was done on a 3-fold cross validation because of the strain of computing power. The results were placed into a pandas data frame. (note they are only commented out as to make sure they did not get re-run)

```
from sklearn.svm import SVC
#initialize the model
svc = SVC()
```

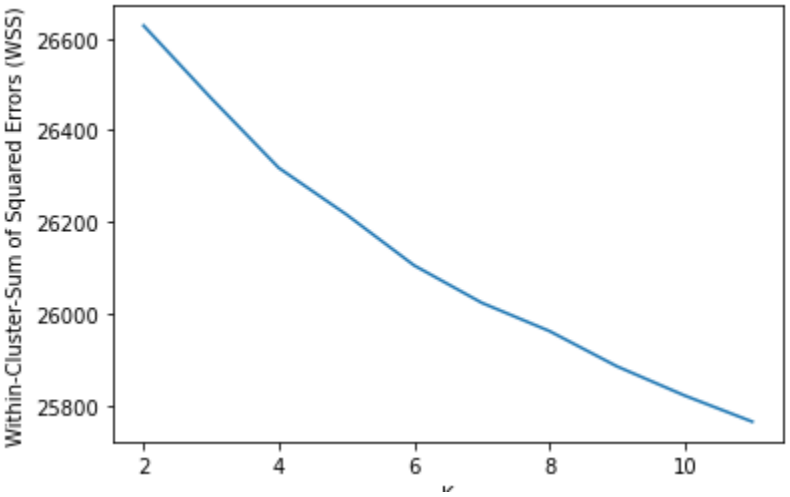
```
#from sklearn.model_selection import GridSearchCV
#param_grid = {'C': [0.1, 1, 10, 100],'kernel': ['linear']}
#grid_svm_ngram = GridSearchCV(svc, param_grid, refit = True, verbose = 3, cv=3)
#grid_svm_ngram.fit(vecsdf, y)
#grid_svm_ngram.cv_results_
```

K-means Clustering

The number of clusters needed to be identified, the elbow method was used. Multiple k means were run capturing the WSS (within cluster sum of squares) information and storing it into a list. Each time the k-means was run the k value (number of clusters) is increased, in this case from 2 to 12 number of clusters. This captured information is graphed, and based off the change in slope, certain k values are created into models, to try and gain information from the data.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
import sklearn.cluster as cluster
import sklearn.metrics as metrics
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
K=range(2,12)

wss = []

for k in K:
    kmeans=cluster.KMeans(n_clusters=k)
    kmeans=kmeans.fit(vecs)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)
```

```
plt.xlabel('K')
plt.ylabel('Within-Cluster-Sum of Squared Errors (WSS)')
plt.plot(K,wss)
```



This yielded bad information as the slope doesn't change with any great degree, multiple kmeans were run to find the division of the clusters. The only clustering with this data that yielded actual clusters was found to be 2 clusters. All other models of more clusters had high

amount of overlapping clusters. These clusters do not show negative, neutral and positive tweets. With some inspection information was still gained and will be discussed in the conclusion.

The kmeans was run using 2 clusters and k-means++ which allows for better starting locations of the centroids.

```
# setting the kmeans to use k-means++ and the number of clusters to 4
kmeans = cluster.KMeans(n_clusters=2 ,init="k-means++")
kmeans = kmeans.fit(vecs)
```

PCA (Principal component analysis) is a way to select/reduce the number of dimensions that data has so it can be graphed accordingly. In this case 2 was selected because a 2d graph is desired. The first PCA created was the x and the second PCA created was the y.

```
from sklearn.decomposition import PCA

#creating teh cluster prediction to graph
clusters = kmeans.predict(vecs)

# the PCA is trained off the dense version of tfidf
pca = PCA(n_components=2)

two_dim = pca.fit_transform(vecs.todense())

# saving the PCA values to plot
scatter_x = two_dim[:, 0] # first principle component
scatter_y = two_dim[:, 1] # second principle component
```

```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('ggplot')

fig, ax = plt.subplots()
fig.set_size_inches(20,10)

# Selecting the color for each cluster
cmap = {0: 'green', 1: 'blue'
        #,2: 'red', 3: 'yellow', 4: 'purple', 5: 'orange'
       }

#Producing a scatter plot were each cluster is the group and selecting the color
# also creating legend with the color to the grouping
for group in np.unique(clusters):
    ix = np.where(clusters == group)
    ax.scatter(scatter_x[ix], scatter_y[ix], c=cmap[group], label=group)

ax.legend()
plt.xlabel("PCA 0")
plt.ylabel("PCA 1")
plt.show()
```

Final Model

All of the best options were selected (vectorization, features sets, and hyperparameters) a 5-fold cross validation was implemented to find the accuracy and F1-scores for each neutral, negative, and positive. 5-fold was used because of the computing power required and time.

```
from sklearn.svm import SVC
svc = SVC(C=1, kernel='linear')
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
import numpy as np
svm_model_final = svc.fit(vecsdf,y)
```

```
svm_cv_scores_final = cross_val_score(svc, vecsdf, y, cv=5)
```

```
svm_cv_neu_f_scores_final = cross_val_score(svc,vecsdf, y,scoring=make_scorer(f1_score, average='weighted'
                                                                               , labels=['neutral']),cv=5)
```

```
svm_cv_neg_f_scores_final = cross_val_score(svc,vecsdf, y,scoring=make_scorer(f1_score, average='weighted'
                                                                               , labels=['negative']),cv=5)
```

```
svm_cv_pos_f_scores_final = cross_val_score(svc,vecsdf, y,scoring=make_scorer(f1_score, average='weighted'
                                                                               , labels=['positive']),cv=5)
```

```
final_model_dict['svm_cv_scores_final']=svm_cv_scores_final
final_model_dict['svm_cv_neg_f_scores_final']=svm_cv_neg_f_scores_final
final_model_dict['svm_cv_neu_f_scores_final']=svm_cv_neu_f_scores_final
final_model_dict['svm_cv_pos_f_scores_final']=svm_cv_pos_f_scores_final
```

```
import matplotlib.pyplot as plt

# function to add value labels
def addlabels(y):
    for i in range(len(y)):
        plt.text(i,y[i],y[i],fontweight = 'bold')


values = list(final_model_dict.values())
names = list(final_model_dict.keys())
colors = ['red','red','red','red'

]
plt.figure(figsize=(60, 10))

plt.subplot(121)
plt.bar(names
        ,values
        ,color= colors
       )

plt.xticks(rotation= 45)
plt.ylabel('Percent')
plt.xlabel('Feature Set')
plt.title('SVM Final Model Accuracy and F1-Scores')

addlabels(values)
plt.show()
```

Finally, some of the indicative features for predicting were looked at and are displayed.

```
def f_importances(coef, names, top=-1):
    imp = coef
    imp, names = zip(*sorted(list(zip(imp, names))))

    # Show all features
    if top == -1:
        top = len(names)

    plt.barh(range(top), imp[::-1][0:top], align='center')
    plt.yticks(range(top), names[::-1][0:top])
    plt.show()
```

```
f_importances(abs(svm_model_final.coef_[0]), feature_names, top=10)
```

Results

Sentiment

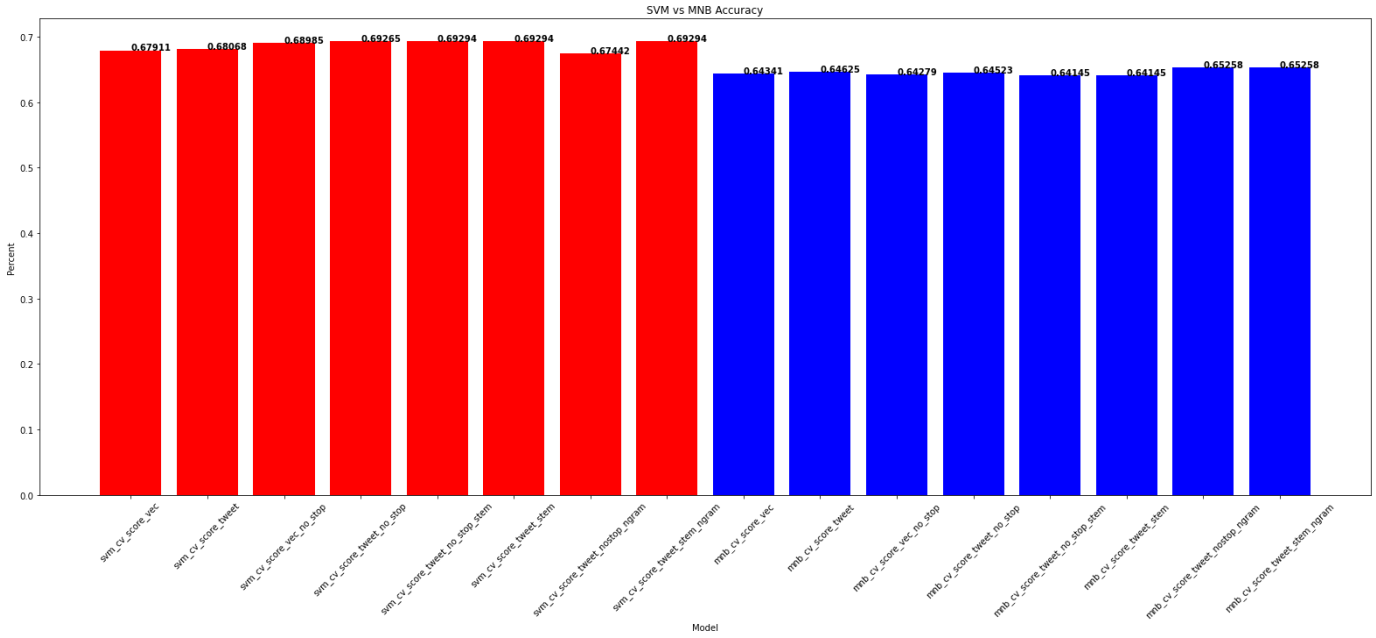
The accuracy's to compare the finish model against to determine how well the model created was performing.

Raw agreement ~40.45% accuracy

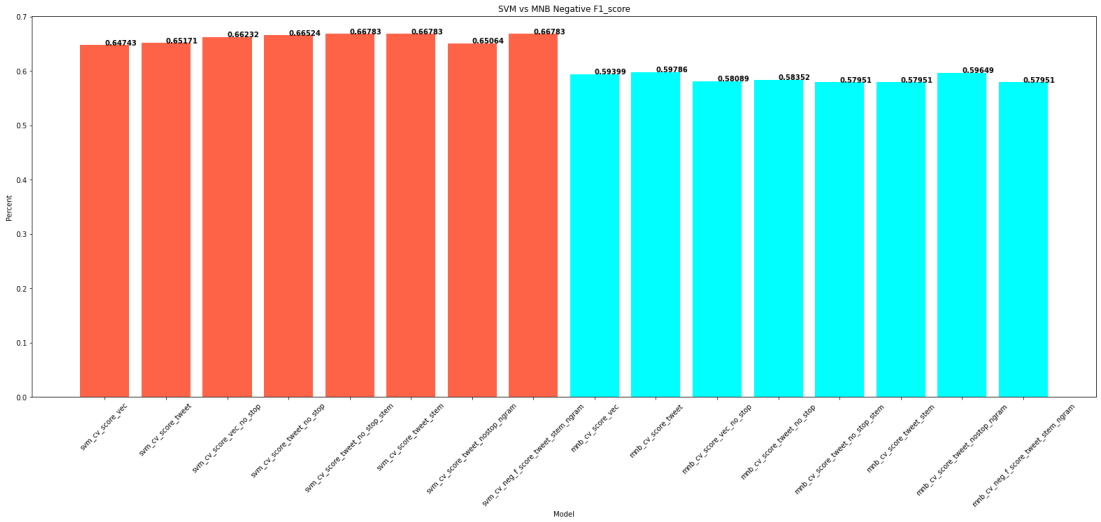
Baseline is ~33.33%. accuracy

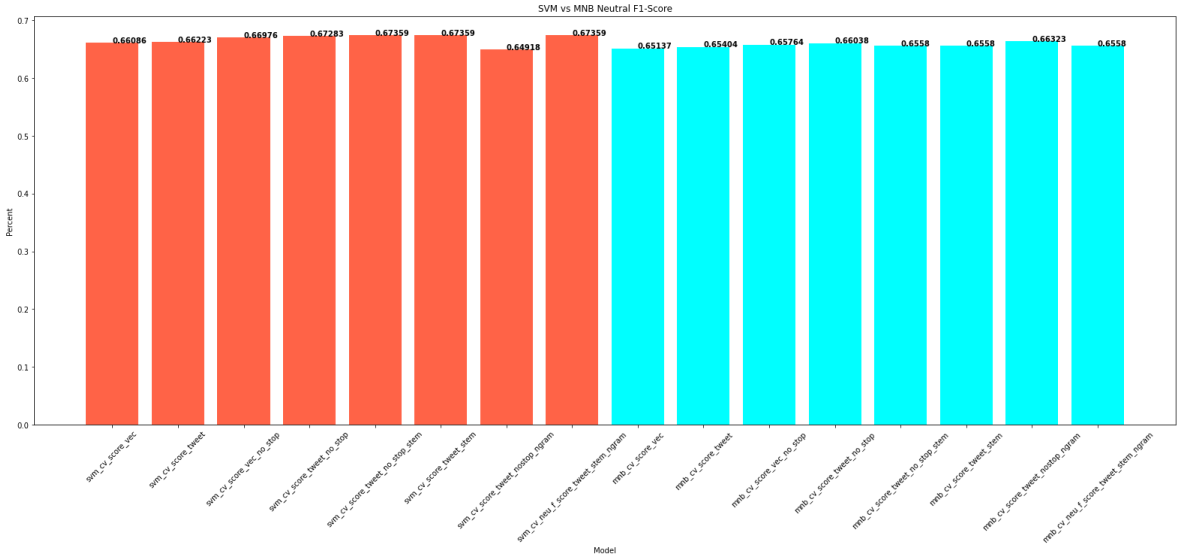
sentiment analyzer ~63.90% accuracy

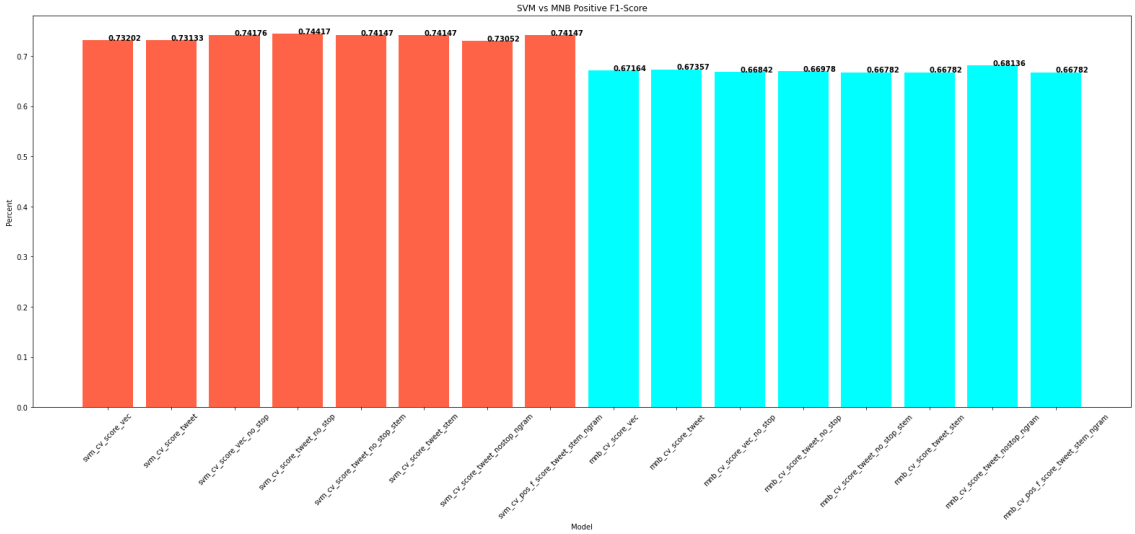
Vectorization Comparison of SVM vs MNB Accuracy



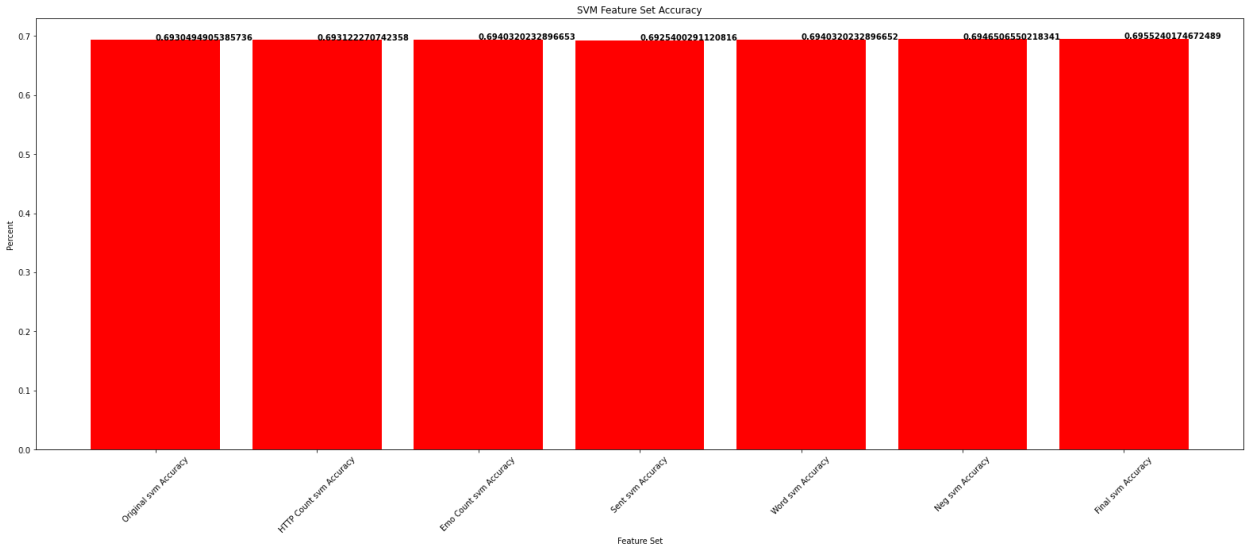
Vectorization Comparison of SVM vs MNB F1-Scores



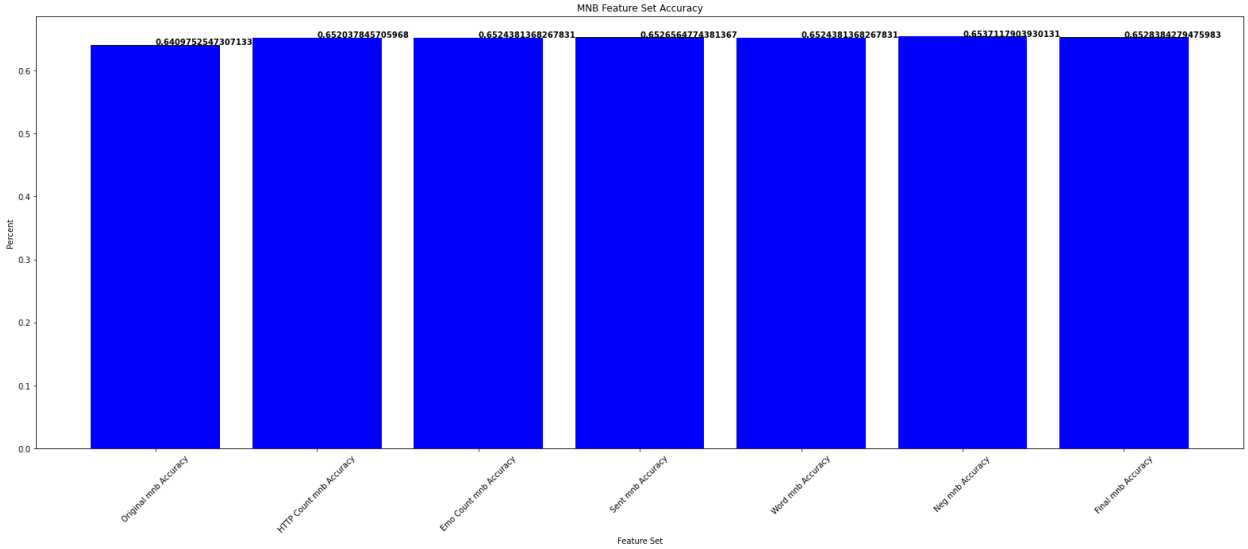




Feature Set Engineering SVM



Feature Set Engineering MNB



As you can see from the above graphs, there is around 1 percentage point improvement between the original accuracy vs accuracy of the model after applying the feature set.

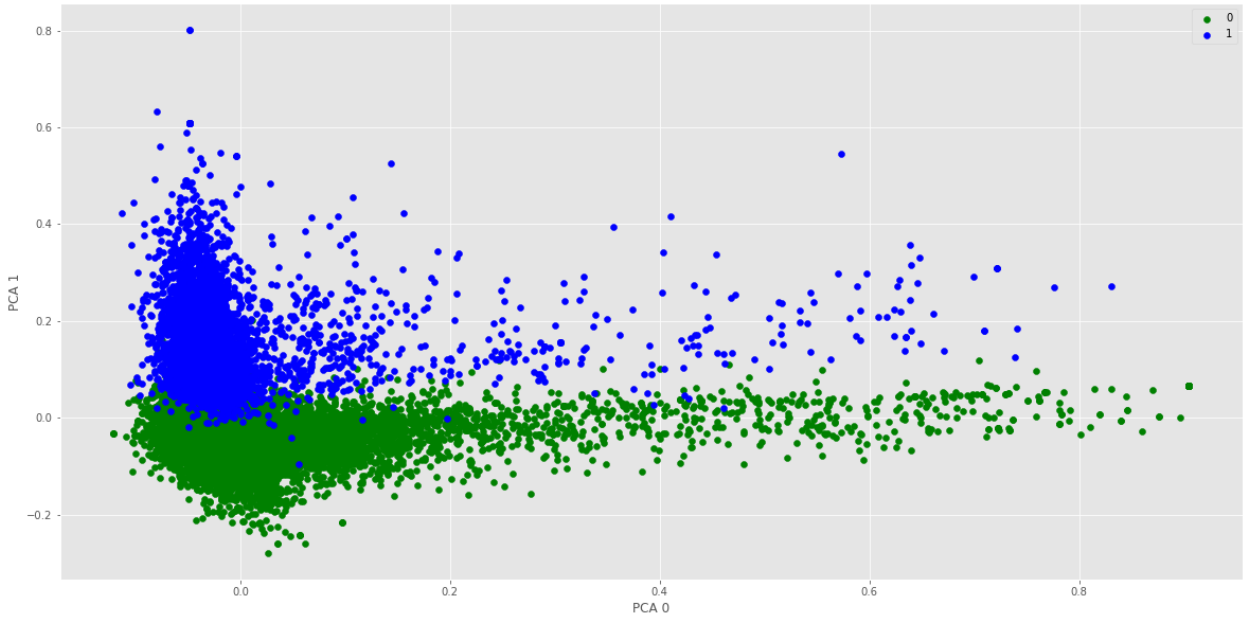
Gridsearchcv Hyperparameter Tuning

```
#svm_ngram_df
```

e_time	std_score_time	param_C	param_kernel	params	split0_test_score	split1_test_score	split2_test_score	mean_test_score	std_test_score	rank_test_score
665642	8.807955	0.1	linear	{'C': 0.1, 'kernel': 'linear'}	0.630459	0.629258	0.634389	0.631368	0.002191	4
851850	0.669213	1	linear	{'C': 1, 'kernel': 'linear'}	0.708406	0.712009	0.713319	0.711245	0.002077	1
752024	0.153277	10	linear	{'C': 10, 'kernel': 'linear'}	0.677729	0.680022	0.673581	0.677111	0.002666	2
092519	0.238366	100	linear	{'C': 100, 'kernel': 'linear'}	0.646070	0.651310	0.645524	0.647635	0.002608	3

C = 1 was shown to have the highest accuracy.

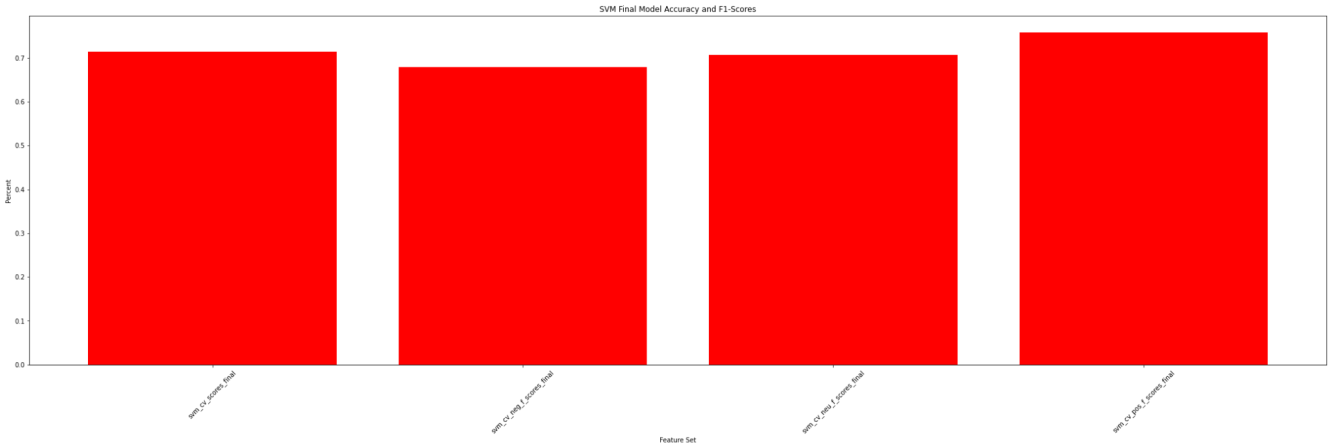
Kmeans Clustering



Final Model

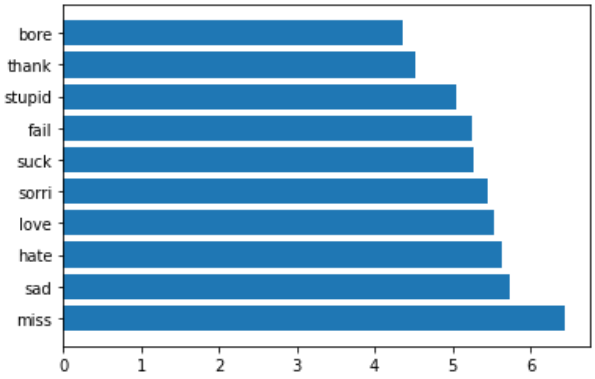
```
print('final accuracy:',svm_cv_score_final)
print('final neutral f score:',svm_cv_neu_f_score_final )
print('final negative f score:', svm_cv_neg_f_score_final)
print('final positive f score:',svm_cv_pos_f_score_final )
```

```
final accuracy: 0.7146652110625908
final neutral f score: 0.70677
final negative f score: 0.67853
final positive f score: 0.75833
```



This is the result for the final model with 5 fold cross validation. Due to the limitation of computing power, we couldn't be able to perform 10 fold cross validation of the model to compare the performance. However, due to the size of our dataset, 5 fold cross validation can generate a significant result. The final accuracy is around 71%, F1 score for neutral is around 71%, for negative is around 68% and for positive is around 76%. Moreover, the dataset is unbalanced so the accuracy score is not accurate but the F1 score is close to the accuracy score.

Feature importances



From the above graph, it showed the top 10 feature importance of the final model. Most of the words are negative such as stupid, fail, suck, hate, bore and sad. Some of the words are positive such as love and thank.

Conclusion

Vectorization Comparison of SVM vs MNB

Looking at the Accuracy overall SVM outperforms MNB. While looking deeper at the vectorization options of the SVM models, all models where porter stemming was performed have the same accuracy. If the F1-Scores are examined, they show that any vector with SVM and stemming are performing the same level as well. This allowed for the use of unigram vectors with no additional stop words removed to be used for the remainder of the tests. (this was selected as it was the simplest model).

While looking at MNB the accuracy is only around 5% behind, so feature set engineering was still done in case something dramatically increased the accuracy. With MNB, both ngram (unigram plus bigram) models performed at the same accuracy which was the highest accuracy, while looking at the F1-Scores we see the model with no porter stemming outperforms the porter stemmed model, this vectorization was used for the feature sets.

Feature Set Engineering

It was found that many of the feature sets increase the accuracy of the models. While this may only be by slight amounts from the first model's original accuracy without any feature sets to the SVM model with the combination of negation and word count the accuracy increases from ~69.30% to ~69.55%. This is a 0.25% increase which accounts for ~ 70 more correct classifications.

Lessons Learnt in the Process:

The models out of the box are considering everything that is in the data already breaking them down into word tokens. So, all we may be doing in pre-processing is to either break it down further or to replace certain things such as emoticons and abbreviations. What might end up happening is that the data may collapse a bit on itself. For example, if we replace a ":-)" or an actual smiley with word "smile", a column in the vector for that smiley is reduced and that data would collapse into the word "smile" in the vector if it already exists or it replaces the column header from the actual smiley to the word smile - effectively either staying in place or slightly reducing the data going to the model.

Model accuracy may go up or down depending upon the type of data and whether data collapsing is desired or not by the given model. In some cases, feature engineering may change the results in both positive as well as negative directions.

K-means Clustering

Since K-means clustering is unsupervised the clusters created could only be inspected and how they are being clustered may be revealed. These tweets are not being clustered based on

sentiment. Upon looking at the text of the tweets in each cluster it becomes apparent they are based on the content. One cluster is tweets that are focused on the individual who is writing the tweets while the other cluster is tweets that mention another person.

An examples of tweets in the first cluster is as shown below:

as much as i love to be hopeful, i reckon the chances are minimal =P i`m never gonna get my cake and stuff
I really really like the song Love Story by Taylor Swift
My Sharpie is running DANGERously low on ink
i want to go to music tonight but i lost my voice.

Some examples of tweets in the second cluster are:

'you can ride one, you can catch one, but its not summer til you pop open one' ?
Hey, you change your twitter account, and you didn`t even tell me...
i miss you bby wish you were going tomorrow to make me do good.
Hi how are you doing ??? *just joined twitter...*
Happy Mothers day to all you Mums out there

Final Model

The overall accuracy was raised from ~69.29% to ~71.47. This indicates that around 607 more tweets were classified correctly. The F1-Scores show that the model does a fairly good job of correctly identifying the tweets without overfitting.

Indicative words

The words found to separate the vectors are mostly negative. This is an interesting find and specific for this data. It would be interesting to use this data on unrelated tweets that have a gold standard.

Out-of-the-box Sentiment Comparison

The model created greatly outperforms the markers placed for comparison. Beating the raw agreement and baseline by ~30-40%. Also compared to the out of the box sentiment analyzer of NLTK (Vader) was around 7.5% better. This was very significant if a company was looking to get a good idea about how their products or marketing campaign is doing.

Overall

It is hard to run models with so much data. It takes time and there are many ways to tune them. The results tend to have good results. In the future more regular expressions would have been used. Weeks could have been spent creating regular expressions to manipulate tweets to remove slang and misspellings. The shelf life of a project such as this is also short. Internet slang and how words are used change yearly or shorter. This can be a big asset to any company, especially large companies with lots of products. They can identify what works and what doesn't in terms of marketing, product development, and company image