IDS Project

Sentiment Analysis

Submitted by:

19UCS177 Saurav Somani
19UCS150 Aryan Dhuria
19UCS003 Shreyansh Jain
19UCS137 Shaurya Sethi

Submitted To : **Dr. Sudheer Sharma & Dr. Aloke Datta & Dr. Indradeep Mastan**

Code : https://github.com/Saurav-Somani/IDS_Project

Dataset : https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

Aim

To classify the given statements in the dataset in positive and negative sentiments using an appropriate ML classification algorithm.

Data Description

The dataset contains around 3000 instances of sentences from three different websites which are imdb,amazon and yelp. Each sentence contains 2 attributes, the statement and a value which is 0 for negative sentiment and 1 for positive sentiment.

Approach

We will follow a quite simple process. First of all reading the dataset in a format which we will use further in our whole project and separating the sentiments and statements. And then we have gone for analysis such as the number of statements in a particular category, if there are any missing values, relation between the length of the statement and the sentiment. Then text preprocessing and then splitting the dataset into training dataset and testing dataset and applying various ML classification algorithms to check each one's efficiency.

Importing libraries

We will import basic python libraries for text analysis, data visualisation and for modeling and learning. We have imported ‘pandas’, ‘numpy’ for data manipulation, nltk for natural language processing, ‘matplotlib’ for graphs and sklearn for modeling and learning.

```
import pandas as pd
import numpy as np
import string, re
import itertools
import nltk
import plotly.offline as py
import plotly.graph_objs as go
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud,STOPWORDS
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

- `from keras.models import Sequential`
- `from keras.callbacks import EarlyStopping`

Text preprocessing & data analysis

Now we will read the dataset by downloading it. We will use the reviews from two sites one is amazon and the other is imdb. We will concatenate the two inputs so that we can use the whole as a single file.

Code:

```
imdb = pd.read_table('imdb_labelled.txt',names=['Statement', 'Sentiment'])
amazon = pd.read_table('amazon_labelled.txt',names=['Statement', 'Sentiment'])

text = pd.concat([amazon,imdb])
text
```

Output:

	Statement	Sentiment
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value.	1
2	Great for the jawbone.	1
3	Tied to charger for conversations lasting more...	0
4	The mic is great.	1
...	...	...
743	I just got bored watching Jessice Lange take h...	0
744	Unfortunately, any virtue in this film's produ...	0
745	In a word, it is embarrassing.	0
746	Exceptionally bad!	0
747	All in all its an insult to one's intelligence...	0

1748 rows × 2 columns

The text contains 1748 rows or reviews and it has 2 columns one for review and the other for sentiment associated with each review.

Now checking for missing values. For this we will use the dropna() inbuilt function of pandas library. If the number of rows after passing the text through the function is the same as before then there are no null values or missing values.

Code and output for the same is:

```
print(text.shape)
text = text.dropna()
print(text.shape)
```

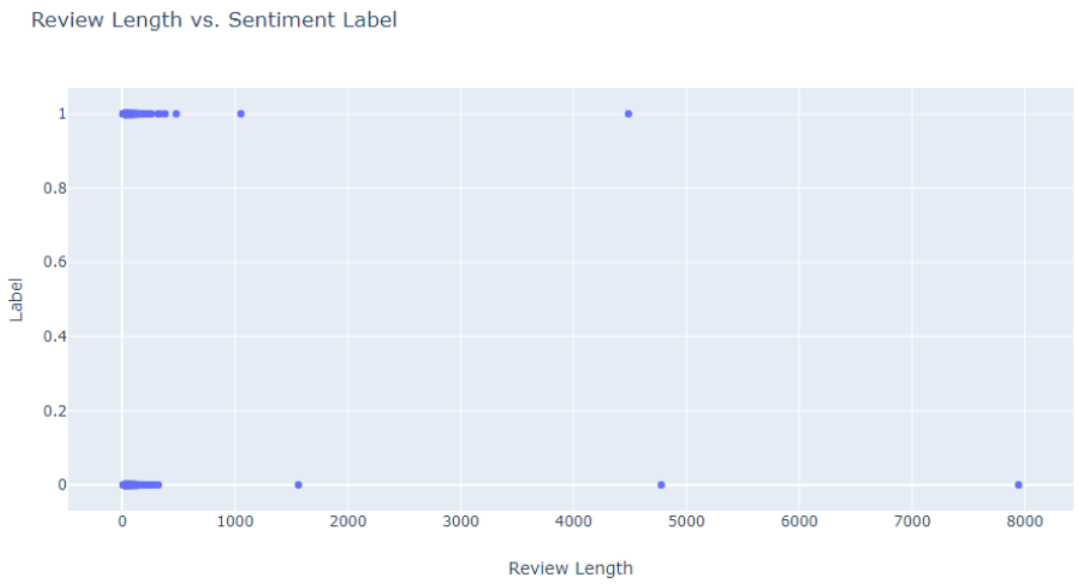
```
(1748, 2)
(1748, 2)
```

Hence there are no missing values in the dataset.
Finding the relation between length of the review and the sentiment label

For reviewing the relationship between review length and sentiment label.
Code:

```
text=df
text['senLen'] = text['Sentence'].apply(lambda x: len(x))
data = text.sort_values(by='senLen')
plot = go.Scatter(x = data['senLen'], y = data['Sentiment'], mode='markers')
lyt = go.Layout(title="Review Length vs. Sentiment Label", xaxis=dict(title='Review Length'),yaxis=dict(title='Label'))
fig = go.Figure(data=[plot], layout=lyt)
py.iplot(fig)
```

Output:



Now, moving on to the parts of removing all the unnecessary punctuation and converting everything to lowercase and numbers from the dataset as they will be of

no use in our further process of classification. For this we will be using the 'nltk' inbuilt library.
Code :

```
def remove_punctuation(sentences):
    comp = re.compile("[%s\d]" % re.escape(string.punctuation))
    return " ".join(comp.sub(" ", str(sentences)).split()).lower()

text['Statement'] = text['Statement'].apply(remove_punctuation)
```

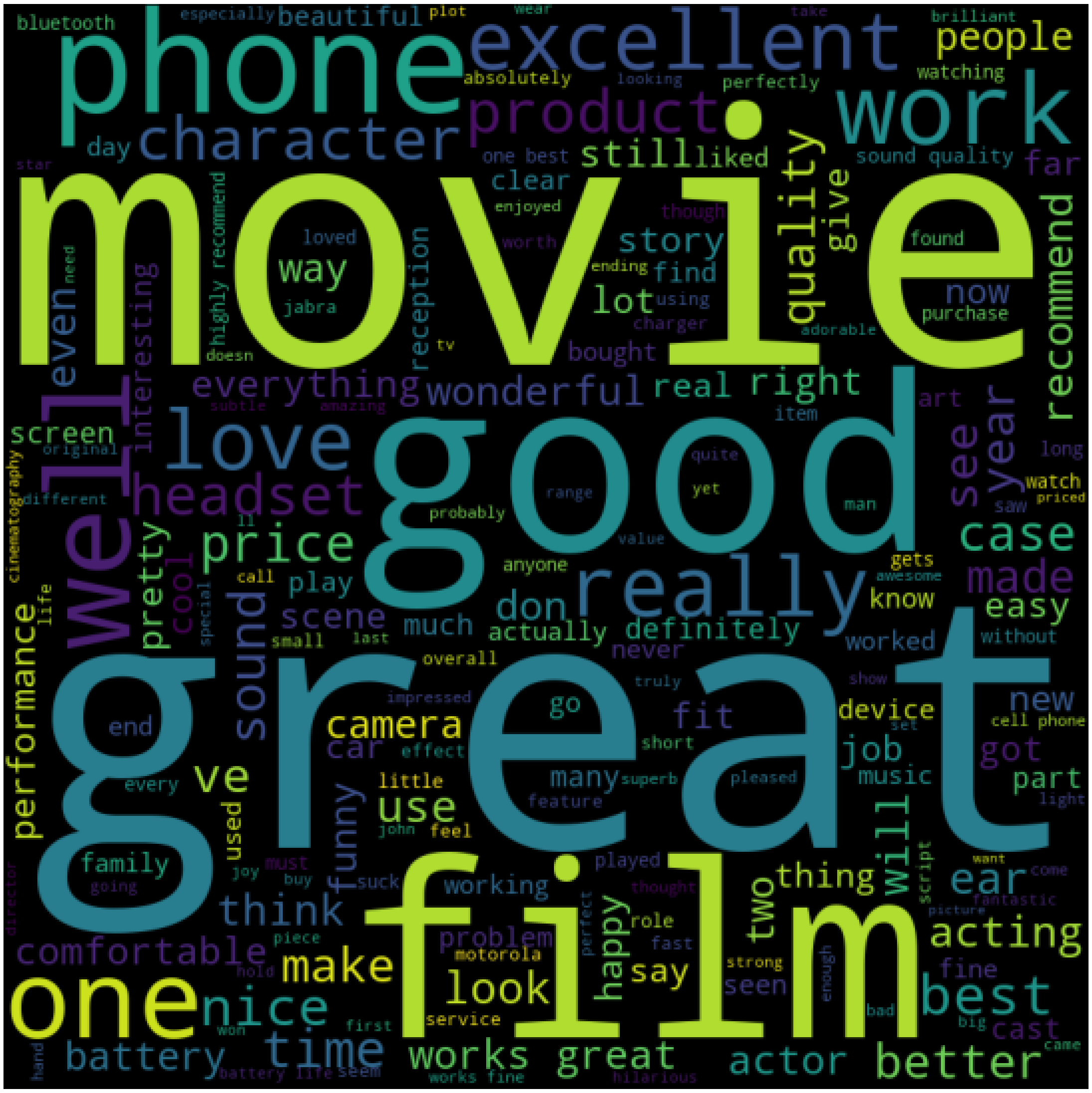
After this now let us see which are the major words which are occurring in the sentiment. For this we will use word cloud. For making the word cloud we have to divide the statements in the 2 categories, one which has sentiment value as 1 while the other will have the sentiment value as 0. We have formed the word cloud removing all the stop words like of,etc. For this we have used the 'WordCloud' module.

For positive sentiment:

Code:

```
text_negative = text[ text['Sentiment'] == 0]
text_negative = text_negative['Statement']

text_positive = text[ text['Sentiment'] == 1]
text_positive = text_positive['Statement']

wordcloud_positive = WordCloud(stopwords=STOPWORDS,
                      background_color='black',
                      width=800,
                      height=800,
                      min_font_size=10).generate(" ".join(text_positive))
plt.figure(figsize=(10, 10),facecolor=None)
plt.imshow(wordcloud_positive)
plt.axis('off')
plt.show()
```

Output:

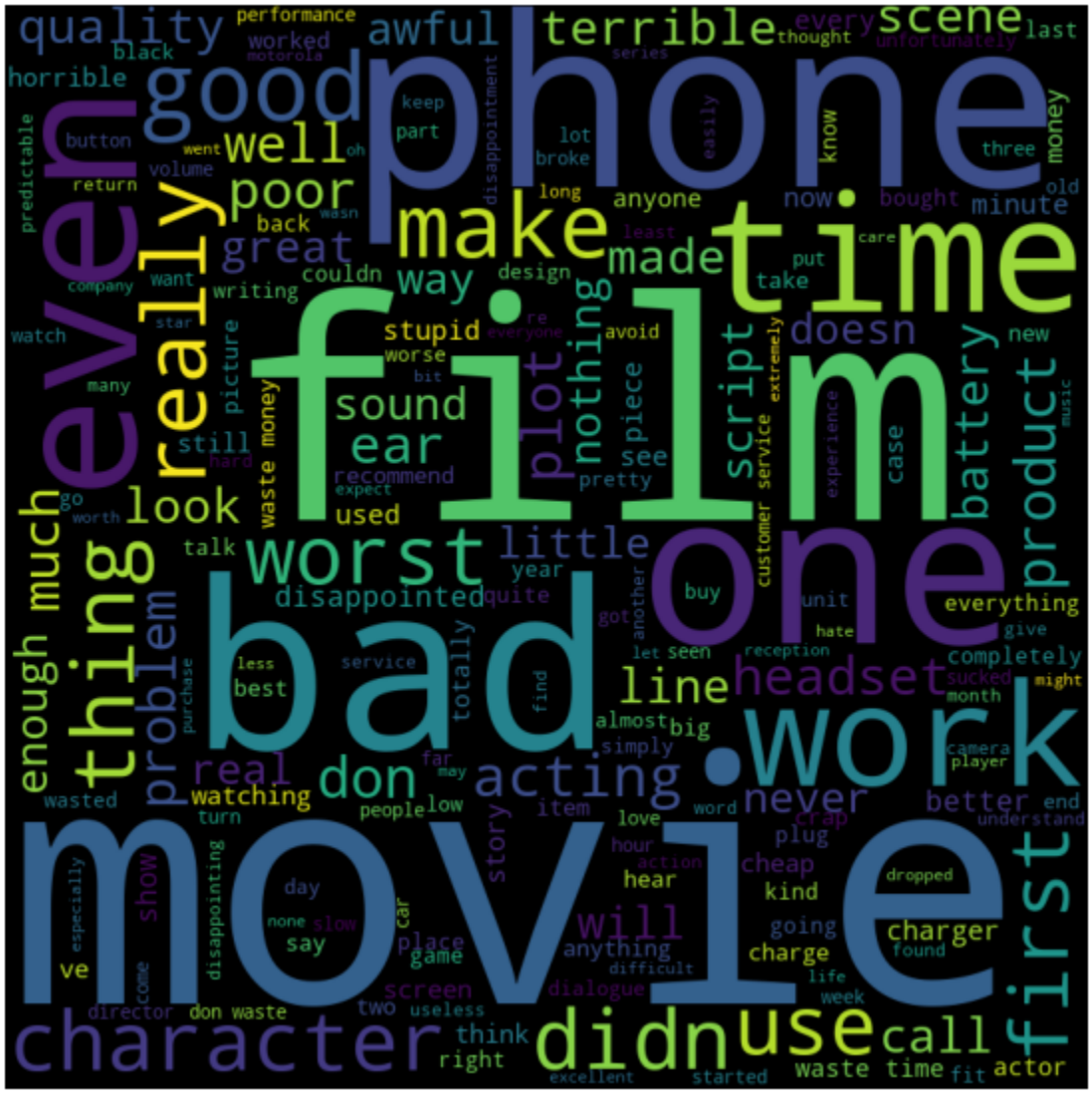

movie
great
good
film
phone
excellent
work
product
one
well
love
headset
character
really
price
nice
time
quality
recommend
story
way
even
sound
battery
comfortable
think
make
look
works great
actor
better
best
acting
ear
thing
two
will
happy
camera
use
ve
funny
performance
family
everything
wonderful
real
right
screen
interesting
pretty
cool
scene
much
don
actually
definitely
case
made
easy
know
worked
new
device
job
got
music
part
fine
cast
see
year
people
bluetooth
beautiful
especially
plot
brilliant
watching
absolutely
perfectly
looking
day
still
liked
sound quality
one best
far
clear
enjoyed
found
now
purchase
give
loved
jabra
lot
using
charger
find
bought
reception
highly recommend
art
item
long
watch
priced
saw
man
case
gets
awesome
without
show
cell phone
light
come
want
fantastic
picture
script
played
working
thought
role
fast
strong
seen
problem
motorola
say
service
first
won
hand
battery life
seem
works fine
hilarious
bad
big
came
piece
hold
going
joy
buy
suck
must
john
feel
little
many
superb
pleased
feature
effect
car
every
end
impressed
go
fit
short
truly
overall
never
small
last
special
call
play
anyone
probably
value
range
quiet
yet
original
different
cinematography
life
director
need
doesn
tv
ending
worth
though
star
subtle
amazing
adorable
enough
perfect
used
set

For negative sentiment:

Code :

```
wordcloud_negative = WordCloud(stopwords=STOPWORDS,
                      background_color='black',
                      width=800,
                      height=800,
                      min_font_size=10).generate(" ".join(text_negative))
plt.figure(figsize=(10, 10),facecolor=None)
plt.imshow(wordcloud_negative)
plt.axis('off')
plt.show()
```

Output:



Inference from word clouds

From the word clouds of the two categories of the statements it is quite clear that their sentiment is strongly associated with the words used in the statement. For positive one, words like good,great,excellent are common. While in negative one, words like bad,worst,didnt are common. So the whole classification will revolve around these words.

Counts of the number of statement in each category
Positive : 886
Negative : 862

Train Our Model

For training our model we will first split our dataset into two parts one for training our model and the other for testing our model. We will use 90% of our dataset for training purposes and the rest for 10% to test it.

As this is a very simple binary classification,we will use naive bayes or SVC(Support vector) classification.Here naive bayes or SVC is best suited because of the following properties of dataset.

1. The reviews are independent of each other.

2. Our task is just to differentiate between two types of sentiment labels.

We will use TF-IDF vectorization for our model. We have tried CountVectorizer, but the accuracy we get through it is less than that of what we get through TF-IDF vectorization.

Accuracy scores for various combinations are:

1. GaussianNB with CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import GaussianNB
vectorizer = CountVectorizer(min_df=2)
train_term = vectorizer.fit_transform(X_train)
test_term = vectorizer.transform(X_test)

model = GaussianNB()
model.fit(train_term.toarray(), y_train)
predictions_train = model.predict(train_term.toarray())
predictions_test = model.predict(test_term.toarray())
print('Train Accuracy:', accuracy_score(y_train, predictions_train))
print('Test Accuracy:', accuracy_score(y_test, predictions_test))
```

```
Train Accuracy: 0.8887476160203432
Test Accuracy: 0.6857142857142857
```

2. GaussianNB with TF-IDF

```
from sklearn.naive_bayes import GaussianNB
vectorizer = TfidfVectorizer(min_df=2)
train_term = vectorizer.fit_transform(X_train)
test_term = vectorizer.transform(X_test)

model = GaussianNB()
model.fit(train_term.toarray(), y_train)
predictions_train = model.predict(train_term.toarray())
predictions_test = model.predict(test_term.toarray())
print('Train Accuracy:', accuracy_score(y_train, predictions_train))
print('Test Accuracy:', accuracy_score(y_test, predictions_test))
```

```
Train Accuracy: 0.9389701207883026
Test Accuracy: 0.7371428571428571
```

3. MultinomialNB with CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=2)
train_term = vectorizer.fit_transform(X_train)
test_term = vectorizer.transform(X_test)

model = MultinomialNB()
model.fit(train_term, y_train)
predictions_train = model.predict(train_term)
predictions_test = model.predict(test_term)
print('Train Accuracy:', accuracy_score(y_train, predictions_train))
print('Test Accuracy:', accuracy_score(y_test, predictions_test))
```

```
Train Accuracy: 0.9230769230769231
Test Accuracy: 0.8114285714285714
```

4. MultinomialNB with TF-IDF

```
vectorizer = TfidfVectorizer(min_df=2)
train_term = vectorizer.fit_transform(X_train)
test_term = vectorizer.transform(X_test)

model = MultinomialNB()
model.fit(train_term, y_train)
predictions_train = model.predict(train_term)
predictions_test = model.predict(test_term)
print('Train Accuracy:', accuracy_score(y_train, predictions_train))
print('Test Accuracy:', accuracy_score(y_test, predictions_test))
```

```
Train Accuracy: 0.9313413858868405
Test Accuracy: 0.8285714285714286
```

5. SVC with CountVectorizer

```
vectorizer = CountVectorizer(min_df=2)
train_term = vectorizer.fit_transform(X_train)
test_term = vectorizer.transform(X_test)

model = svm.SVC()
model.fit(train_term, y_train)
predictions_train = model.predict(train_term)
predictions_test = model.predict(test_term)
print('Train Accuracy:', accuracy_score(y_train, predictions_train))
print('Test Accuracy:', accuracy_score(y_test, predictions_test))
```

```
Train Accuracy: 0.8970120788302607
Test Accuracy: 0.8114285714285714
```

6. SVC with TF-IDF

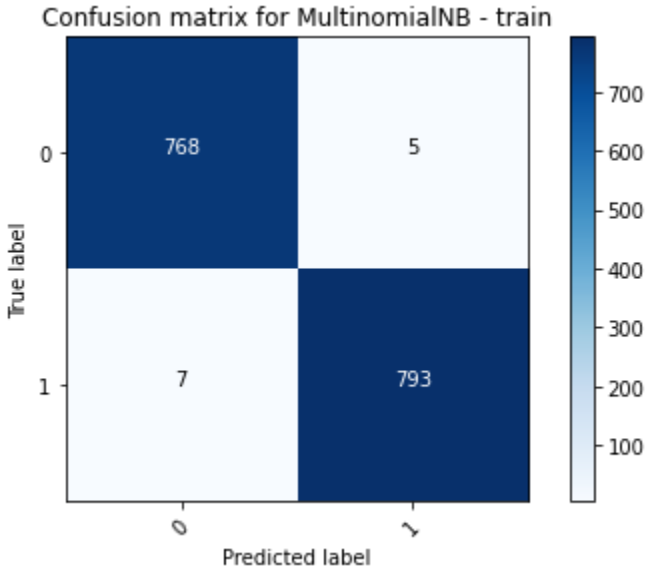
```
from sklearn import svm
vectorizer = TfidfVectorizer(min_df=2)
train_term = vectorizer.fit_transform(X_train)
test_term = vectorizer.transform(X_test)

model = svm.SVC()
model.fit(train_term, y_train)
predictions_train = model.predict(train_term)
predictions_test = model.predict(test_term)
print('Train Accuracy:', accuracy_score(y_train, predictions_train))
print('Test Accuracy:', accuracy_score(y_test, predictions_test))
```

```
Train Accuracy: 0.9910998092816274
Test Accuracy: 0.8342857142857143
```

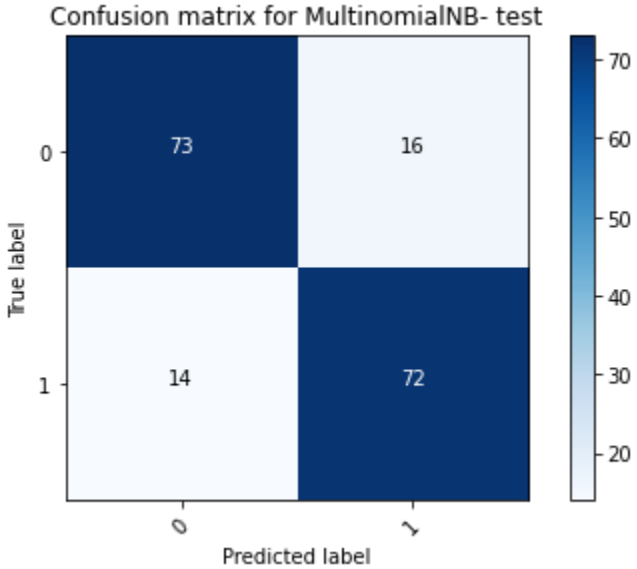
Results & Inferences

We will be drawing a confusion matrix for better understanding of the model.

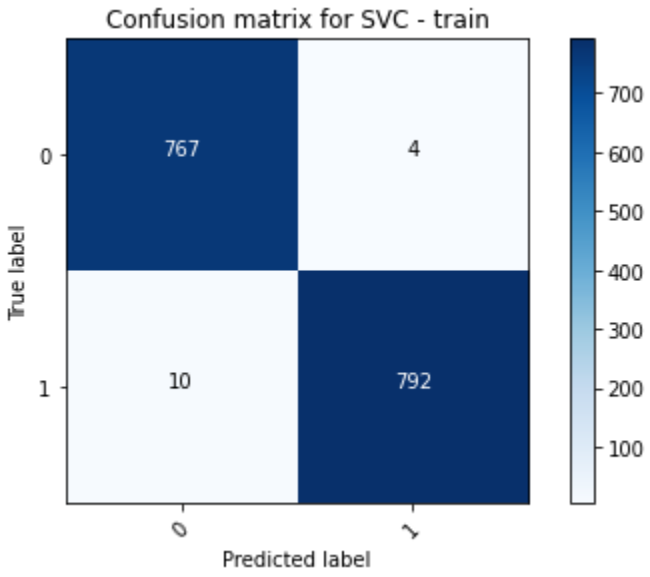
The confusion matrix for training data(MultinomialNB)



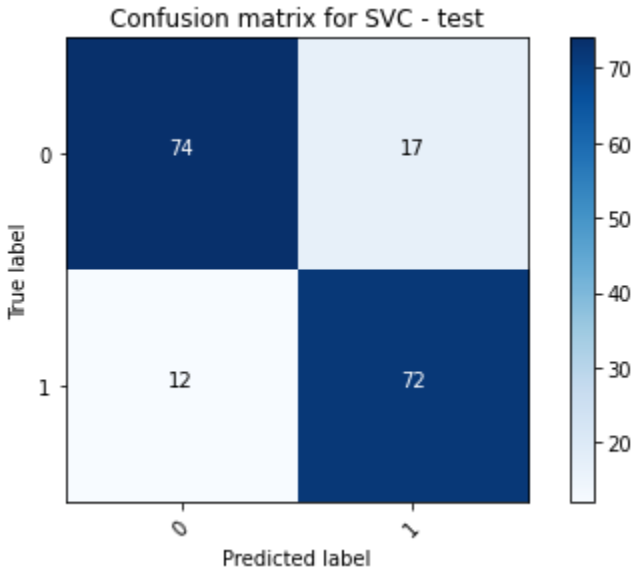
The confusion matrix for test data(MultinomialNB)



The confusion matrix for training data(SVC)



The confusion matrix for test data(SVC)



Precision,recall and f-values for various models are as follows

For Test data

Models	Precision	Recall	f-value
GaussianNB with CountVectorizer	0.764	0.734	0.724
GaussianNB with Tf-idf	0.722	0.720	0.719
MultinomialNB with CountVectorizer	0.817	0.817	0.817
MultinomialNB with Tf-idf	0.835	0.834	0.834
SVC with CountVectorizer	0.768	0.764	0.764
SVC with Tf-idf	0.828	0.828	0.828

Some observations are:

1. POS tagging filtering impairs the performance significantly (to just 0.6 accuracy score).

2. SnowballStemmer doesn't downgrade the performance, while PorterStemmer reduced the accuracy to 0.8472 from 0.8509.

3. Removing stopwords reduced the accuracy to 0.8.

4. Trying bigrams and 3-grams impairs the performance significantly.

Conclusions

From the whole analysis,modelling and testing it is clear that MultinomialNB and SVC with TF-IDF out performs all the other models. Hence they are best suited for sentiment analysis dataset.