Homework 7: Sentiment Analysis of Movie Reviews using Naive Bayes & Support Vector Machines

Student: Patrick Walsh
Professor: Dr. Ami Gates
School: Syracuse University
Date: 3/7/2024

INTRODUCTION

In today's digital age, where opinions are readily shared and influential, understanding customer sentiment has become paramount for businesses striving to stay ahead in the competitive market landscape. Sentiment analysis, a technique that involves extracting insights from text to determine the emotional tone behind it, offers invaluable insights into customer perceptions and preferences. One area where sentiment analysis holds significant potential is in the movie industry.

Movie reviews, whether posted on social media, review platforms, or blogs, are a treasure trove of opinions that reflect audience reactions to films. Analyzing these reviews can provide movie studios, producers, and distributors with critical insights into audience sentiment, helping them gauge the success of their productions, identify areas for improvement, and make informed decisions regarding marketing and distribution strategies.

In this report, we delve into the realm of sentiment analysis of movie reviews using advanced machine learning techniques. By harnessing the power of Naive Bayes and Support Vector Machines (SVM), we aim to decipher the sentiment embedded within movie reviews and shed light on the business value derived from such analysis. Through our exploration, we seek to uncover the most impactful words and features that drive audience perceptions, ultimately empowering businesses to make data-driven decisions that resonate with their target audience and drive success in the dynamic world of cinema.

ANALYSIS

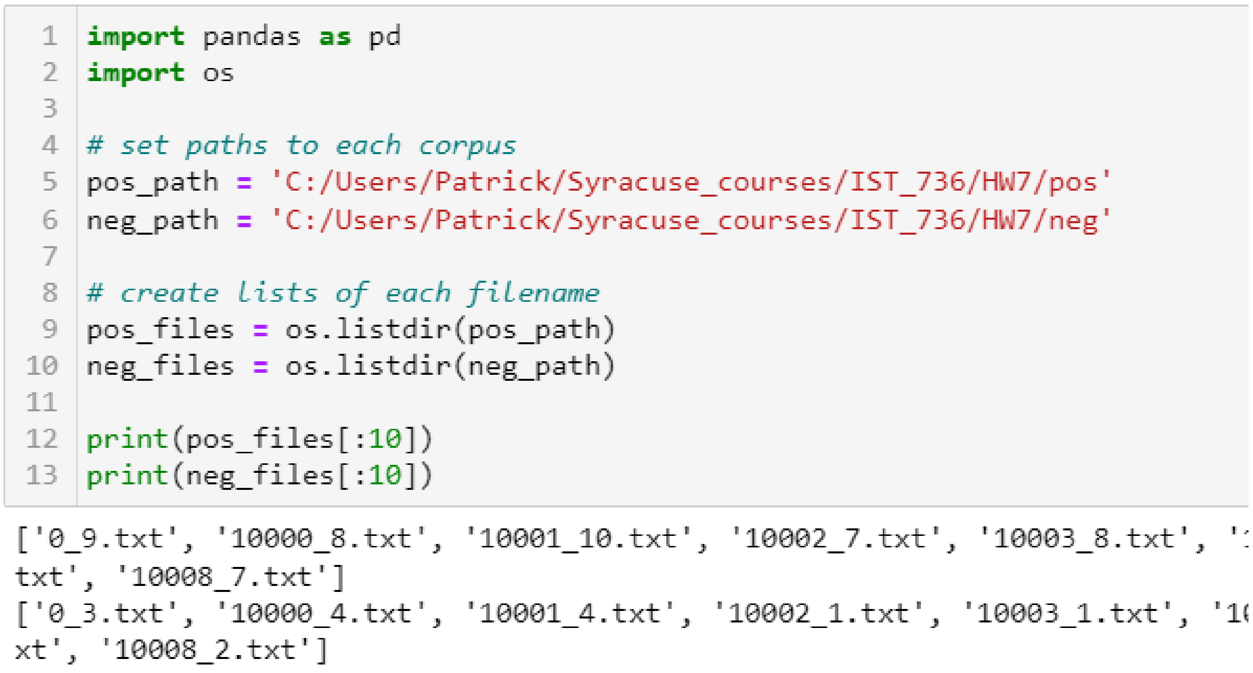
Dataset

The Movie Reviews dataset consists of two corpora categorized into a positive review corpus and a negative review corpus. Each corpus contains 12,500 documents that contain movie reviews that are either positive or negative.

The corpora can be found here:

https://drive.google.com/drive/folders/19oUCruzbXo0OS89OctTaKmvoM3OtaBl2
https://drive.google.com/drive/folders/14RRnbHC3Xwc5S3dZW6ysDMWn4p2td8lL

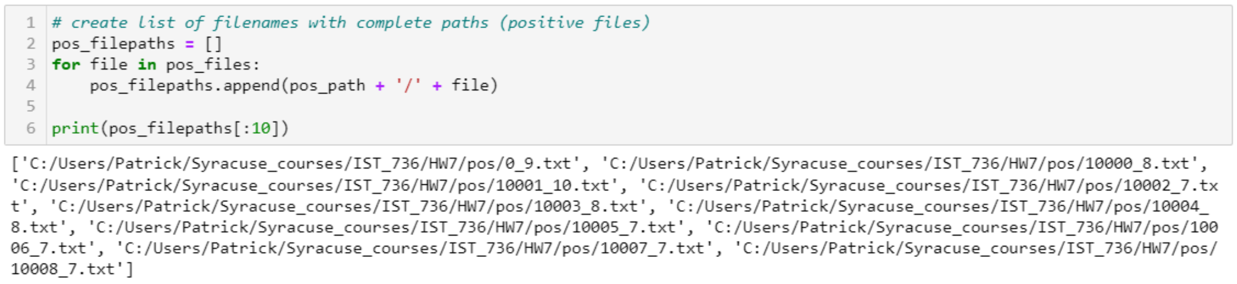
First, we will load the two corpora into os and display the first 10 filenames of each corpus, as seen below.



```python
import pandas as pd
import os

# set paths to each corpus
pos_path = 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos'
neg_path = 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg'

# create lists of each filename
pos_files = os.listdir(pos_path)
neg_files = os.listdir(neg_path)

print(pos_files[:10])
print(neg_files[:10])
```

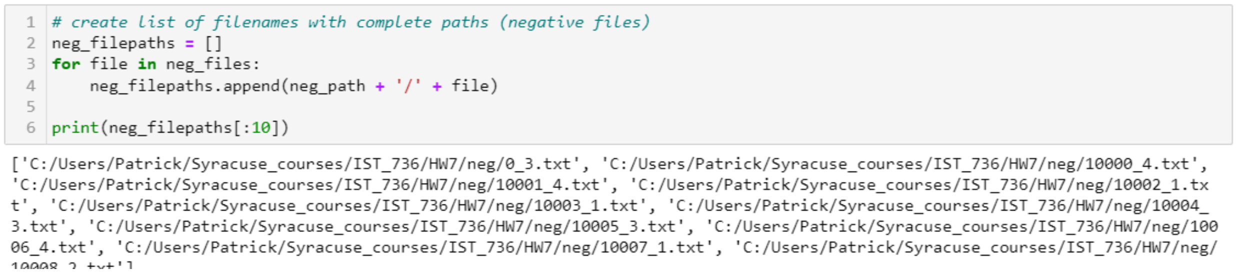
```
['0_9.txt', '10000_8.txt', '10001_10.txt', '10002_7.txt', '10003_8.txt', '
txt', '10008_7.txt']
['0_3.txt', '10000_4.txt', '10001_4.txt', '10002_1.txt', '10003_1.txt', '10
xt', '10008_2.txt']
```

After each corpus is loaded into os as a list of filenames, they need to be formatted to contain the full filepath for each document. This step is necessary before the corpora can be loaded into a dataframe using CountVectorizer() from the SciKit-Learn package. Once the corpora have contain the full filepath information, we will examine the first 10 files of each corpus to confirm this step was completed successfully.



```
# create list of filenames with complete paths (positive files)
pos_filepaths = []
for file in pos_files:
    pos_filepaths.append(pos_path + '/' + file)

print(pos_filepaths[:10])
```

```
['C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/0_9.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/10000_8.txt',
'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/10001_10.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/10002_7.tx
t', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/10003_8.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/10004_
8.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/10005_7.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/100
06_7.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/10007_7.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/pos/
10008_7.txt']
```



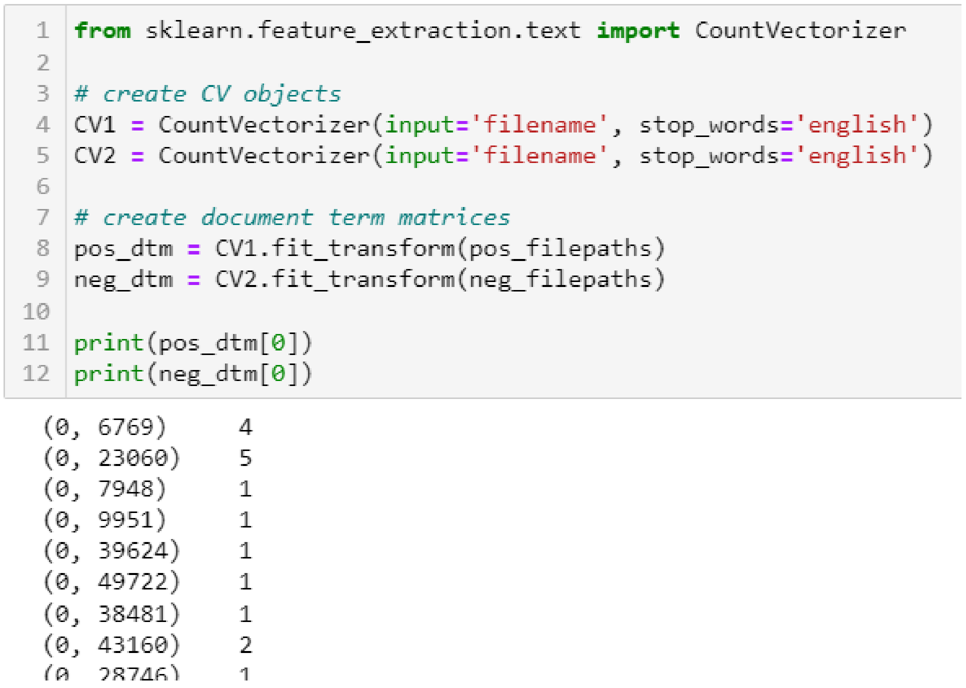
```
# create list of filenames with complete paths (negative files)
neg_filepaths = []
for file in neg_files:
    neg_filepaths.append(neg_path + '/' + file)

print(neg_filepaths[:10])
```

```
['C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/0_3.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/10000_4.txt',
'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/10001_4.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/10002_1.tx
t', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/10003_1.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/10004_
3.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/10005_3.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/100
06_4.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/10007_1.txt', 'C:/Users/Patrick/Syracuse_courses/IST_736/HW7/neg/
10008_2.txt']
```

Data Preparation

Next, the corpora must be tokenized and vectorized so that they will be in a numeric format for further processing and training with machine learning models. We import CountVectorizer() from SciKit-Learn and instantiate the CountVectorizer() object. This step tokenizes the documents and creates document term matrices (DTM) which will then be converted to vectorized dataframes. This step is what transforms the unstructured text data into a structured, numeric format for processing in supervised modeling. The screenshot below captures the tokenization and creation of DTMs for the corpora.

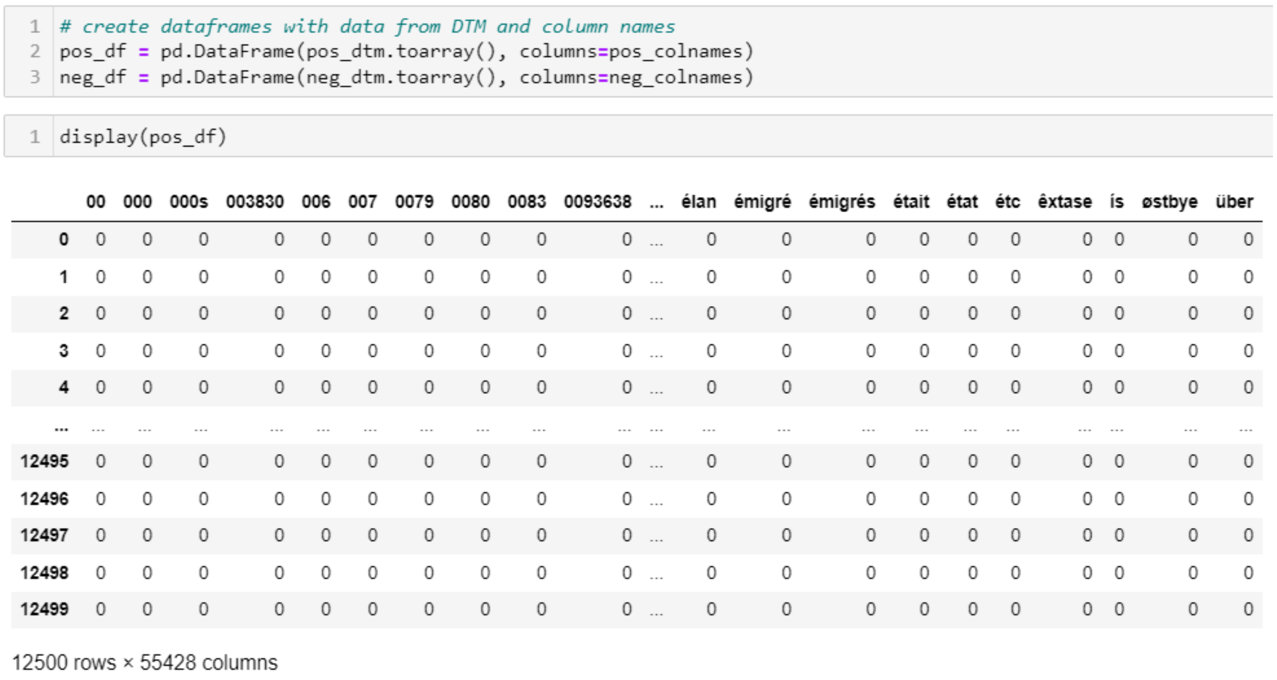
Tokenization & Vectorization ¶



```python
from sklearn.feature_extraction.text import CountVectorizer

# create CV objects
CV1 = CountVectorizer(input='filename', stop_words='english')
CV2 = CountVectorizer(input='filename', stop_words='english')

# create document term matrices
pos_dtm = CV1.fit_transform(pos_filepaths)
neg_dtm = CV2.fit_transform(neg_filepaths)

print(pos_dtm[0])
print(neg_dtm[0])
```

```
  (0, 6769)	4
  (0, 23060)	5
  (0, 7948)	1
  (0, 9951)	1
  (0, 39624)	1
  (0, 49722)	1
  (0, 38481)	1
  (0, 43160)	2
  (0, 28746)	1
```

Once the two corpora are in a DTM, the final step is to vectorize the DTMs using the .toarray() method and Pandas. The final output is shown below as a structured dataframe with rows and columns.



```python
# create dataframes with data from DTM and column names
pos_df = pd.DataFrame(pos_dtm.toarray(), columns=pos_colnames)
neg_df = pd.DataFrame(neg_dtm.toarray(), columns=neg_colnames)
```

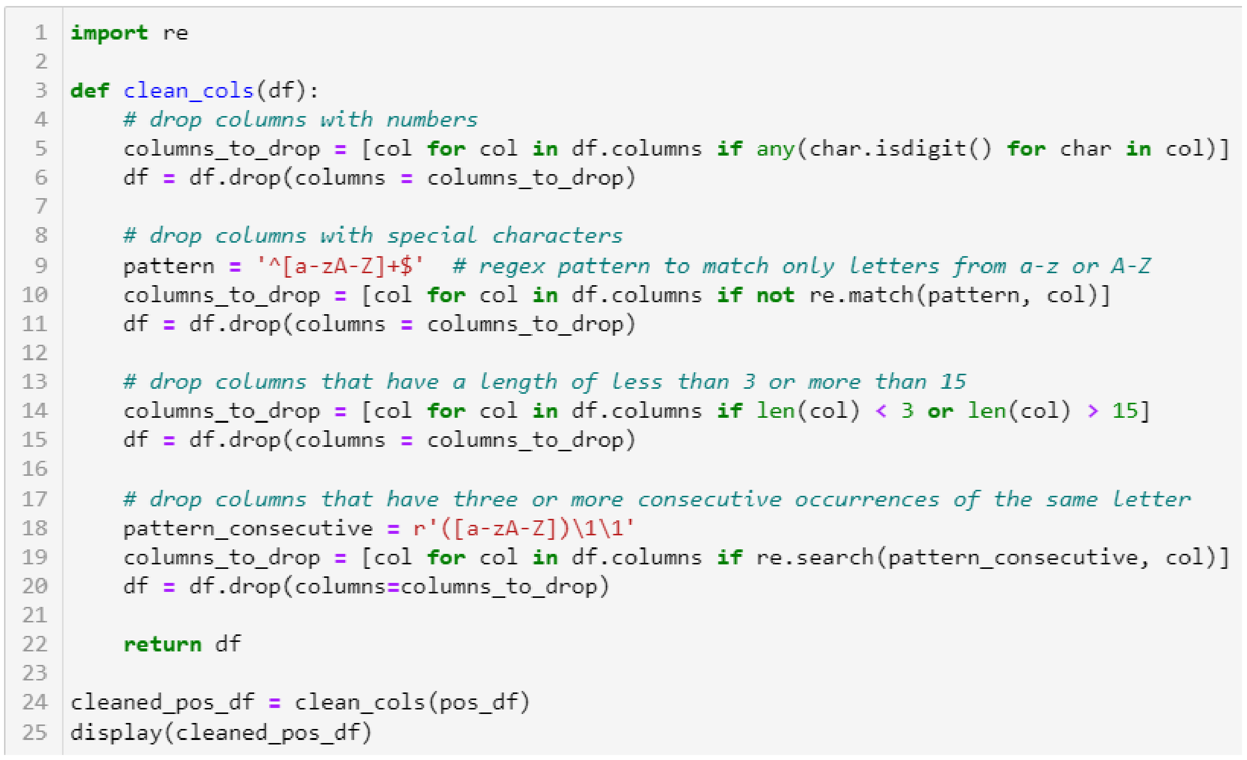
```python
display(pos_df)
```

	00	000	000s	003830	006	007	0079	0080	0083	0093638	...	élan	émigré	émigrés	était	état	étc	êxtase	ís	østbye	über
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12495	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12496	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12497	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12498	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12499	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

12500 rows × 55428 columns

Each row represents a document, and each column represents a unique token (essentially a word) from that document. As seen in the screenshot above, there are 12,500 documents in the positive reviews corpus, with 55,428 unique tokens across all the documents.

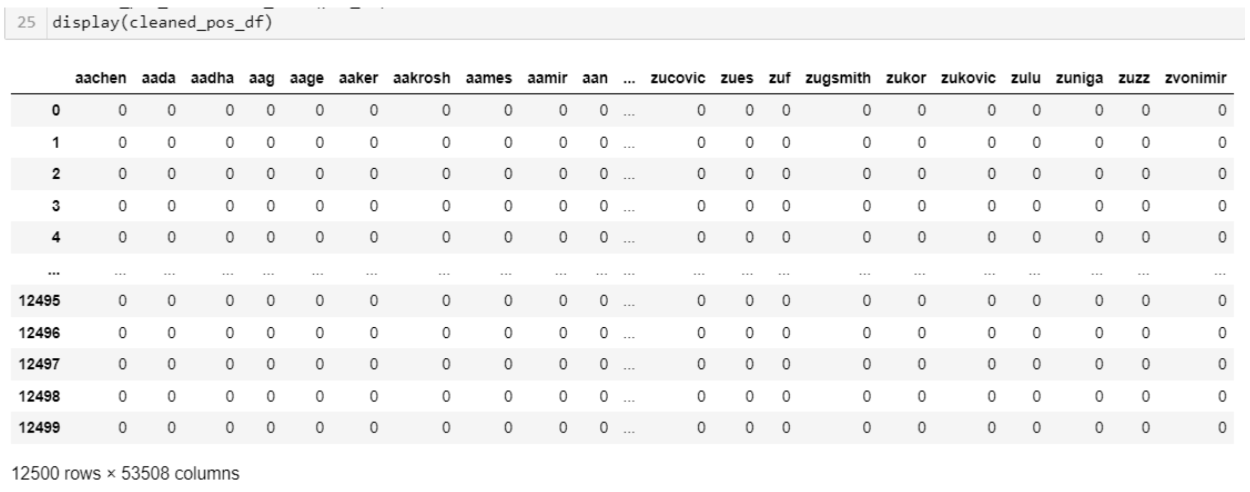
We are not done yet. Additional data cleaning must be done to clean up unwanted or unnecessary tokens from the dataframes. The first cleaning step we will perform is to drop columns that contain numbers, as we only want tokens that potentially represent words. The next cleaning step is to drop any columns with special characters. After that, we will drop columns that are less than three letters long or are more than 15 letters long. This will help cut down on short words like ‘a’, ‘to’, and ‘the’ and also drop tokens with exceptionally long lengths, such as tokens with repeated letters like ‘zzzzzzzzzzzzzzzz’. Finally, we want only dictionary words as tokens, so we will drop tokens that contain three or more consecutive letters such as ‘aaab’ or ‘azzzzaa’.



```python
import re

def clean_cols(df):
    # drop columns with numbers
    columns_to_drop = [col for col in df.columns if any(char.isdigit() for char in col)]
    df = df.drop(columns = columns_to_drop)

    # drop columns with special characters
    pattern = '^[a-zA-Z]+$'  # regex pattern to match only letters from a-z or A-Z
    columns_to_drop = [col for col in df.columns if not re.match(pattern, col)]
    df = df.drop(columns = columns_to_drop)

    # drop columns that have a length of less than 3 or more than 15
    columns_to_drop = [col for col in df.columns if len(col) < 3 or len(col) > 15]
    df = df.drop(columns = columns_to_drop)

    # drop columns that have three or more consecutive occurrences of the same letter
    pattern_consecutive = r'([a-zA-Z])\1\1'
    columns_to_drop = [col for col in df.columns if re.search(pattern_consecutive, col)]
    df = df.drop(columns=columns_to_drop)

    return df

cleaned_pos_df = clean_cols(pos_df)
display(cleaned_pos_df)
```

This data cleaning step will hopefully accomplish two things. 1. Cut out unhelpful tokens that do not add any meaning to the text which should help the models be more accurate. And 2., reduce the dimensionality of the dataframes to decrease training and processing times.

Positive reviews dataframe (cleaned)

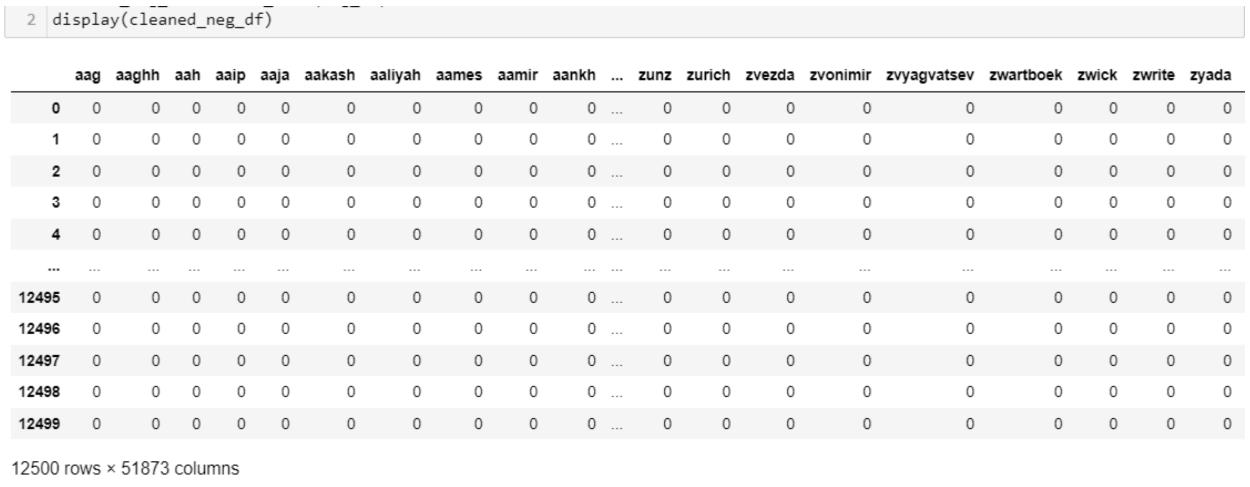


```
display(cleaned_pos_df)
```

	aachen	aada	aadha	aag	aage	aaker	aakrosh	aames	aamir	aan	...	zucovic	zues	zuf	zugsmith	zukor	zukovic	zulu	zuniga	zuzz	zvonimir
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12495	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12496	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12497	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12498	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
12499	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

12500 rows × 53508 columns

Negative reviews dataframe (cleaned)

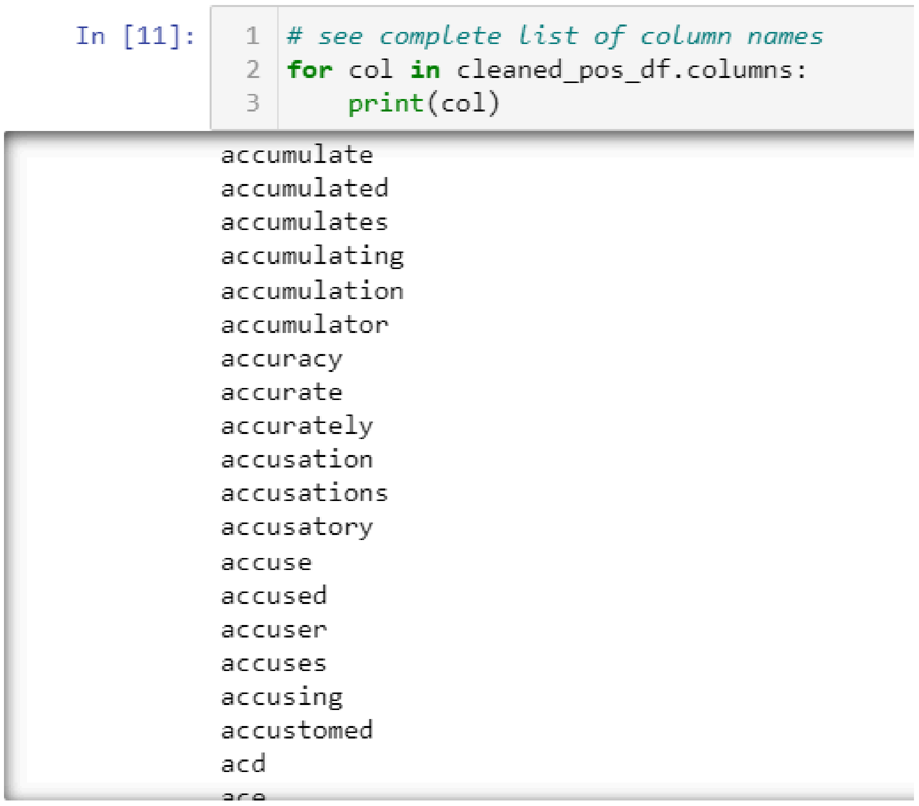


```
display(cleaned_neg_df)
```

	aag	aaghh	aah	aaip	aaja	aakash	aaliyah	aames	aamir	aankh	...	zunz	zurich	zvezda	zvonimir	zvyagvatsev	zwartboek	zwick	zwrite	zyada
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12495	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
12496	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
12497	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
12498	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
12499	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

12500 rows × 51873 columns

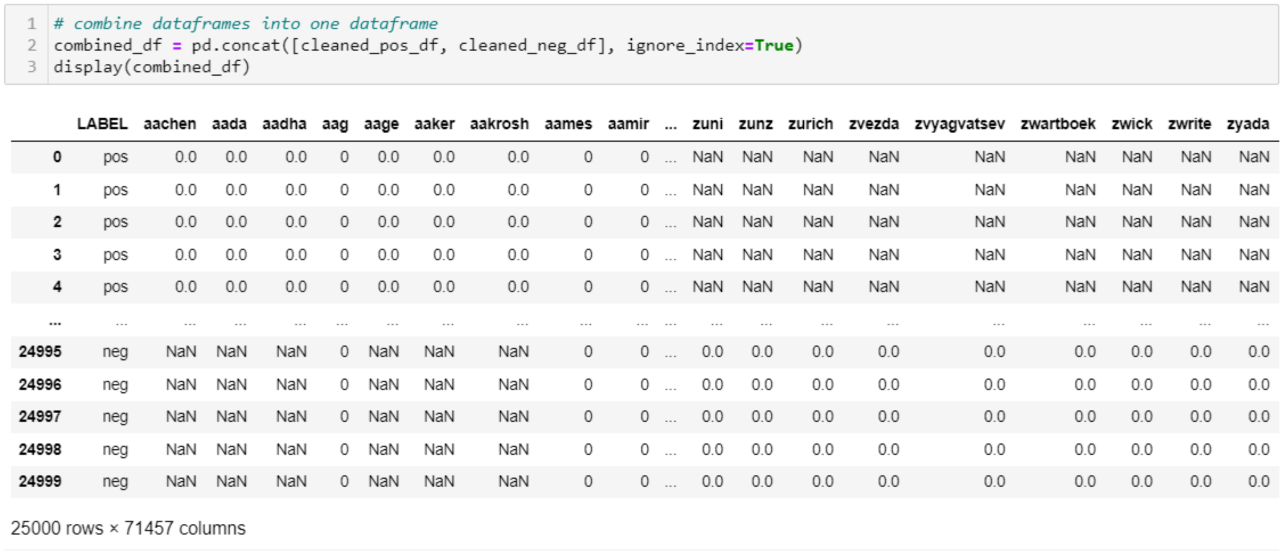
The cleaned dataframes are seen above. While there are still non-dictionary words in these cleaned dataframes, the cleaning steps have helped to reduce the dimensionality and noise. Most of the columns are now dictionary words, as seen in the screen shot below.



```
In [11]:  # see complete list of column names
          for col in cleaned_pos_df.columns:
              print(col)
```

```
accumulate
accumulated
accumulates
accumulating
accumulation
accumulator
accuracy
accurate
accurately
accusation
accusations
accusatory
accuse
accused
accuser
accuses
accusing
accustomed
acd
ace
```

The last data preparation step before moving onto model training is to combine the positive and negative review dataframes into one dataframe with a label column for supervised learning. As seen in the screenshot below, there are now 25,000 rows (documents) and 71,457 columns (words/tokens) in the combined dataframe. However, there are now null values in some of the rows as a result of the concatenation step.

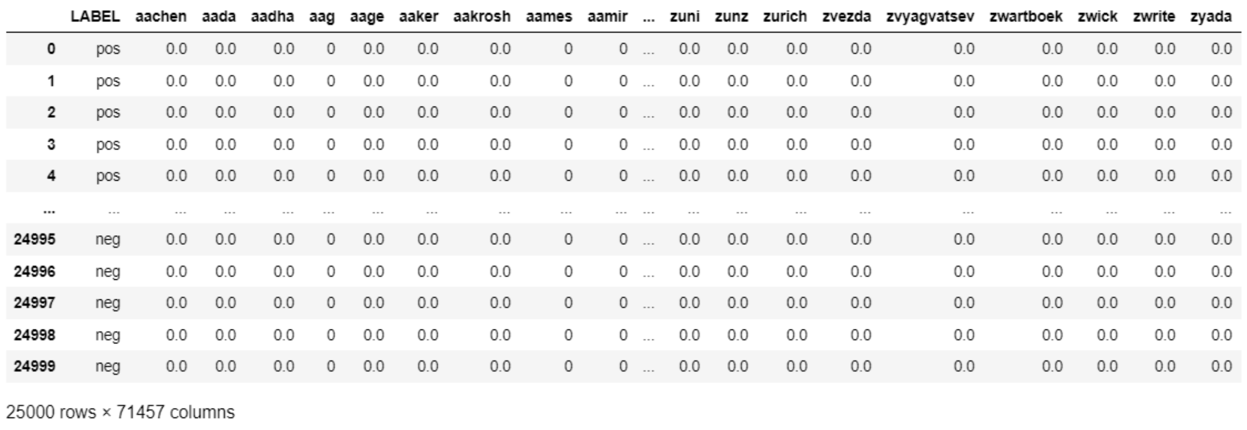


```
# combine dataframes into one dataframe
combined_df = pd.concat([cleaned_pos_df, cleaned_neg_df], ignore_index=True)
display(combined_df)
```

	LABEL	aachen	aada	aadha	aag	aage	aaker	aakrosh	aames	aamir	...	zuni	zunz	zurich	zvezda	zvyagvatsev	zwartboek	zwick	zwrite	zyada
0	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
24995	neg	NaN	NaN	NaN	0	NaN	NaN	NaN	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24996	neg	NaN	NaN	NaN	0	NaN	NaN	NaN	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24997	neg	NaN	NaN	NaN	0	NaN	NaN	NaN	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24998	neg	NaN	NaN	NaN	0	NaN	NaN	NaN	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24999	neg	NaN	NaN	NaN	0	NaN	NaN	NaN	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

25000 rows × 71457 columns

To eliminate null values, we will null values with a zero. The finalized, combined, vectorized and cleaned dataframe is seen below.



	LABEL	aachen	aada	aadha	aag	aage	aaker	aakrosh	aames	aamir	...	zuni	zunz	zurich	zvezda	zvyagvatsev	zwartboek	zwick	zwrite	zyada
0	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	pos	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
24995	neg	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24996	neg	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24997	neg	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24998	neg	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
24999	neg	0.0	0.0	0.0	0	0.0	0.0	0.0	0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

25000 rows × 71457 columns

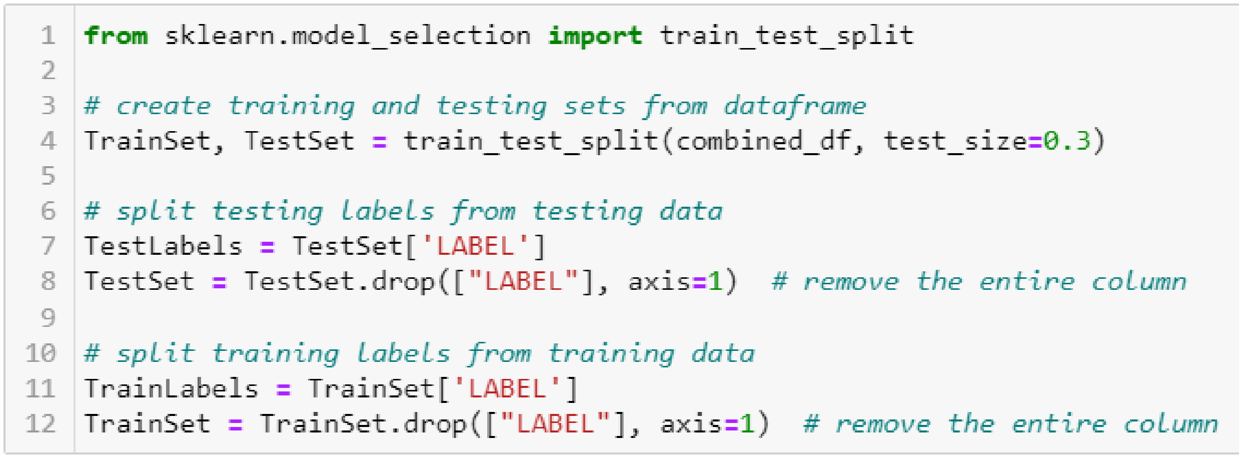
Models

Now that the data is ready, it's time to build and train the models. This exercise will create five distinct models for comparison:

1. Multinomial Naive Bayes,
2. Linear SVC Support Vector Machine (SVM),
3. SVM with polynomial kernel,
4. SVM with Radial Basis Function (RBF) kernel,
5. SVM with Stochastic Gradient Descent (SGD) classifier.

Each model will be instantiated with specific hyperparameters to tune the model for optimal performance and training speed. The dataframe will be split into training and testing sets with a 70/30 split. This means that 70% of the data will be used for training and 30% will be set aside for testing.

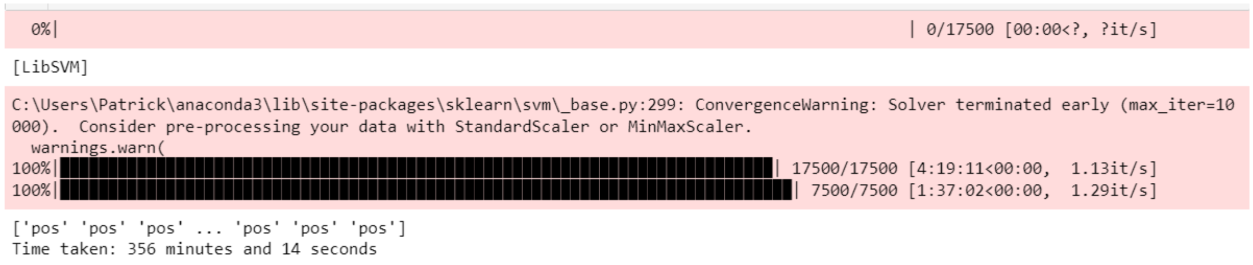
Train/Test Split



```
from sklearn.model_selection import train_test_split

# create training and testing sets from dataframe
TrainSet, TestSet = train_test_split(combined_df, test_size=0.3)

# split testing labels from testing data
TestLabels = TestSet['LABEL']
TestSet = TestSet.drop(["LABEL"], axis=1)  # remove the entire column

# split training labels from training data
TrainLabels = TrainSet['LABEL']
TrainSet = TrainSet.drop(["LABEL"], axis=1)  # remove the entire column
```

Scaling the data

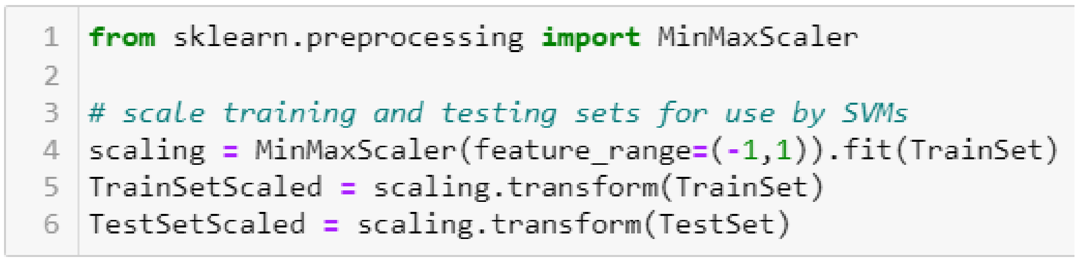
Additionally, the SVM models (models 2-5) will benefit from scaled data to increase training speed. Due to their complexity, SVM models may require several hours to train when used on a large dataset and may fail to converge or reach an acceptable level of accuracy before training time ceases. When I tried running the Linear SVC model (model 2) on data that was not scaled, I received this warning:



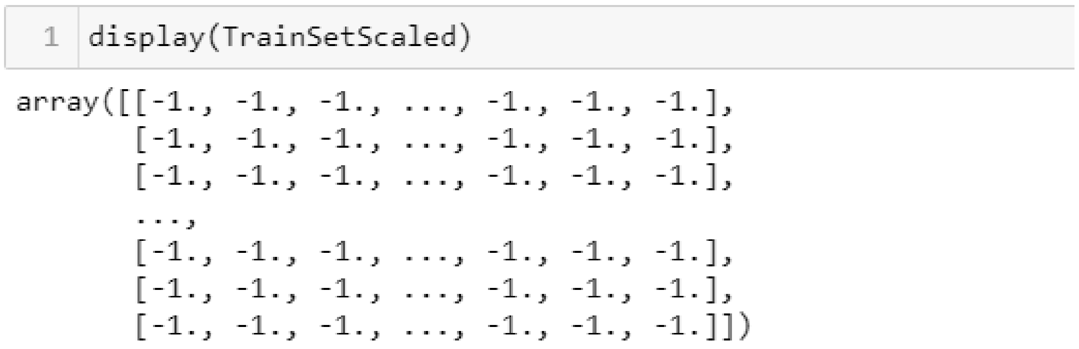
```
  0%|                                                                  | 0/17500 [00:00<?, ?it/s]
[LibSVM]
C:\Users\Patrick\anaconda3\lib\site-packages\sklearn\svm\_base.py:299: ConvergenceWarning: Solver terminated early (max_iter=10
000).  Consider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
100%|██████████████████████████████████████████████████████████████████| 17500/17500 [4:19:11<00:00,  1.13it/s]
100%|████████████████████████████████████████████████████████████████████| 7500/7500 [1:37:02<00:00,  1.29it/s]
['pos' 'pos' 'pos' ... 'pos' 'pos' 'pos']
Time taken: 356 minutes and 14 seconds
```

Training time took almost 6 hours and I got a convergence warning after 10,000 iterations. The warning recommends scaling the data beforehand with either the StandardScaler or MinMaxScaler. When tested, this model achieved an accuracy of 83.2%, which is pretty good, but we will try scaling the data to see if we get better accuracy and faster training time.

While discussed in more detail in the RESULTS section, this model did perform better with scaled data using the MinMaxScaler with a range of -1 to 1 than with unscaled data. The training time was cut down from 6 hours to 4.5 hours, and the accuracy increased from 83.2% to 86.7%. See the screenshot below for how scaling was accomplished on both the training and testing sets.



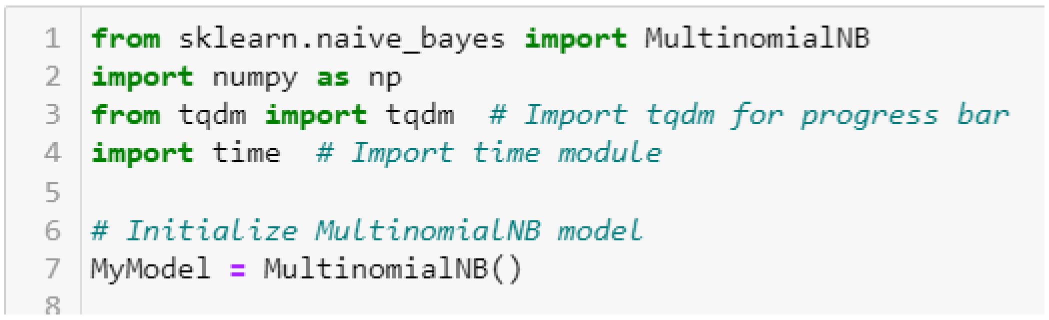
```python
from sklearn.preprocessing import MinMaxScaler

# scale training and testing sets for use by SVMs
scaling = MinMaxScaler(feature_range=(-1,1)).fit(TrainSet)
TrainSetScaled = scaling.transform(TrainSet)
TestSetScaled = scaling.transform(TestSet)
```



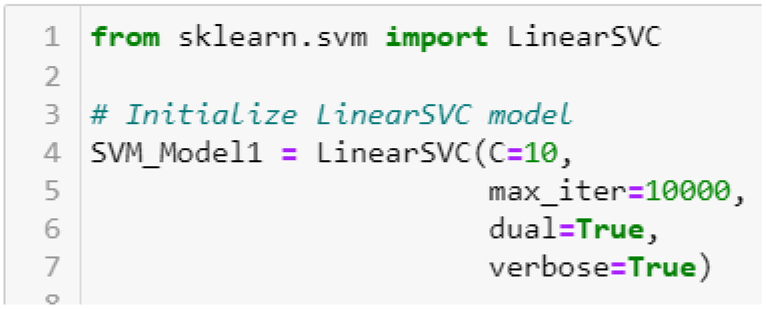
```python
display(TrainSetScaled)
```

```
array([[-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.],
       ...,
       [-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.],
       [-1., -1., -1., ..., -1., -1., -1.]])
```

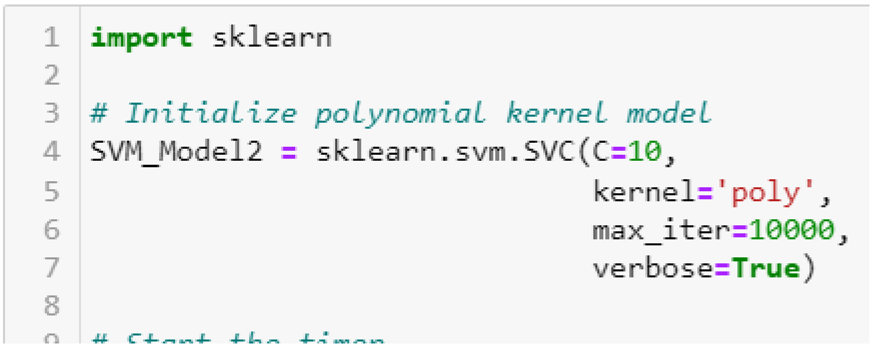
The instantiation and hyperparameters for the 5 models are detailed below.



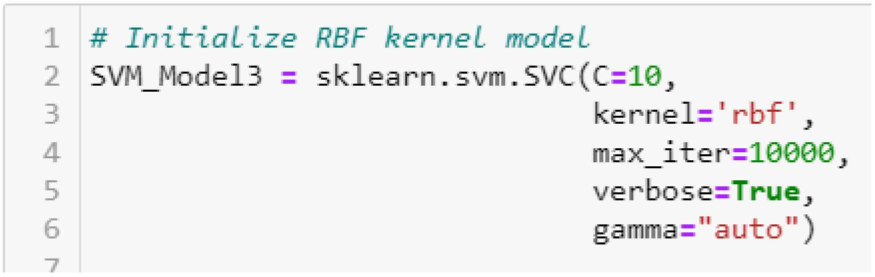
```python
from sklearn.naive_bayes import MultinomialNB
import numpy as np
from tqdm import tqdm  # Import tqdm for progress bar
import time  # Import time module

# Initialize MultinomialNB model
MyModel = MultinomialNB()

```



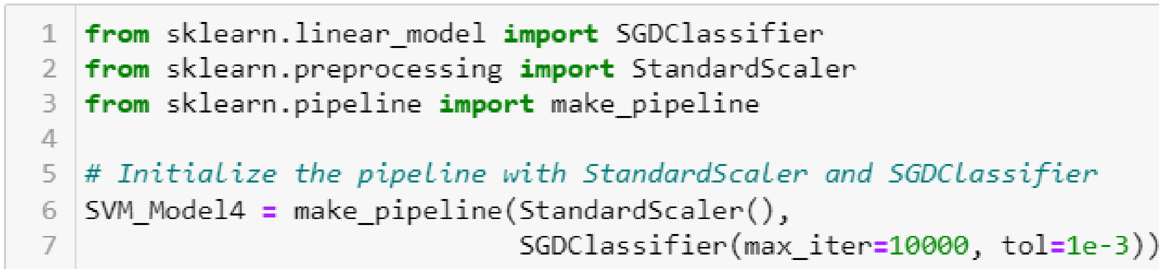
```python
from sklearn.svm import LinearSVC

# Initialize LinearSVC model
SVM_Model1 = LinearSVC(C=10,
                       max_iter=10000,
                       dual=True,
                       verbose=True)
```



```python
import sklearn

# Initialize polynomial kernel model
SVM_Model2 = sklearn.svm.SVC(C=10,
                             kernel='poly',
                             max_iter=10000,
                             verbose=True)

# Start the timer
```



```python
# Initialize RBF kernel model
SVM_Model3 = sklearn.svm.SVC(C=10,
                             kernel='rbf',
                             max_iter=10000,
                             verbose=True,
                             gamma="auto")
```

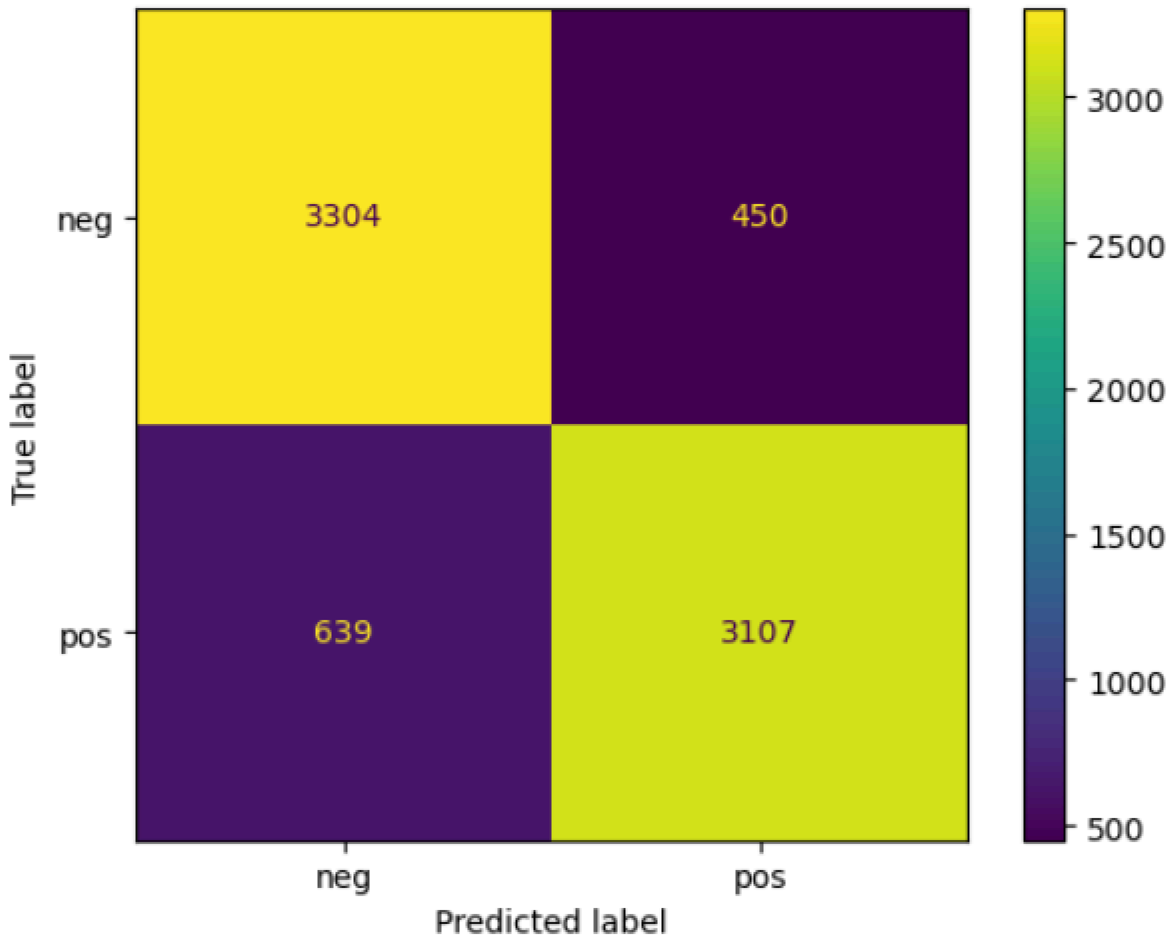


```python
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Initialize the pipeline with StandardScaler and SGDClassifier
SVM_Model4 = make_pipeline(StandardScaler(),
                           SGDClassifier(max_iter=10000, tol=1e-3))
```

RESULTS

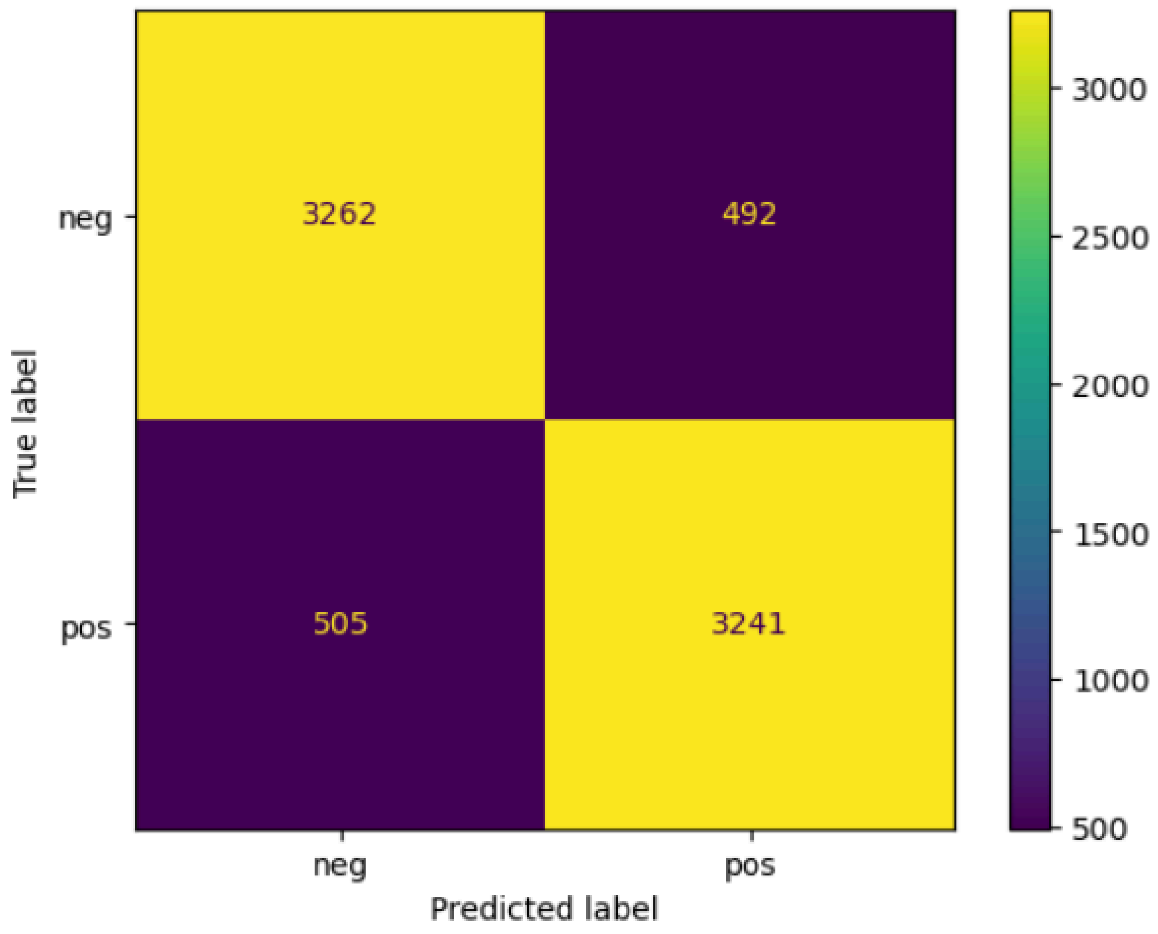
The next few screenshots capture the performance of the 5 models. Performance is measured confusion matrices and accuracy scores. Summary results will also follow.

```
The Model 1 (Multinomial Naive Bayes) confusion matrix is:
```



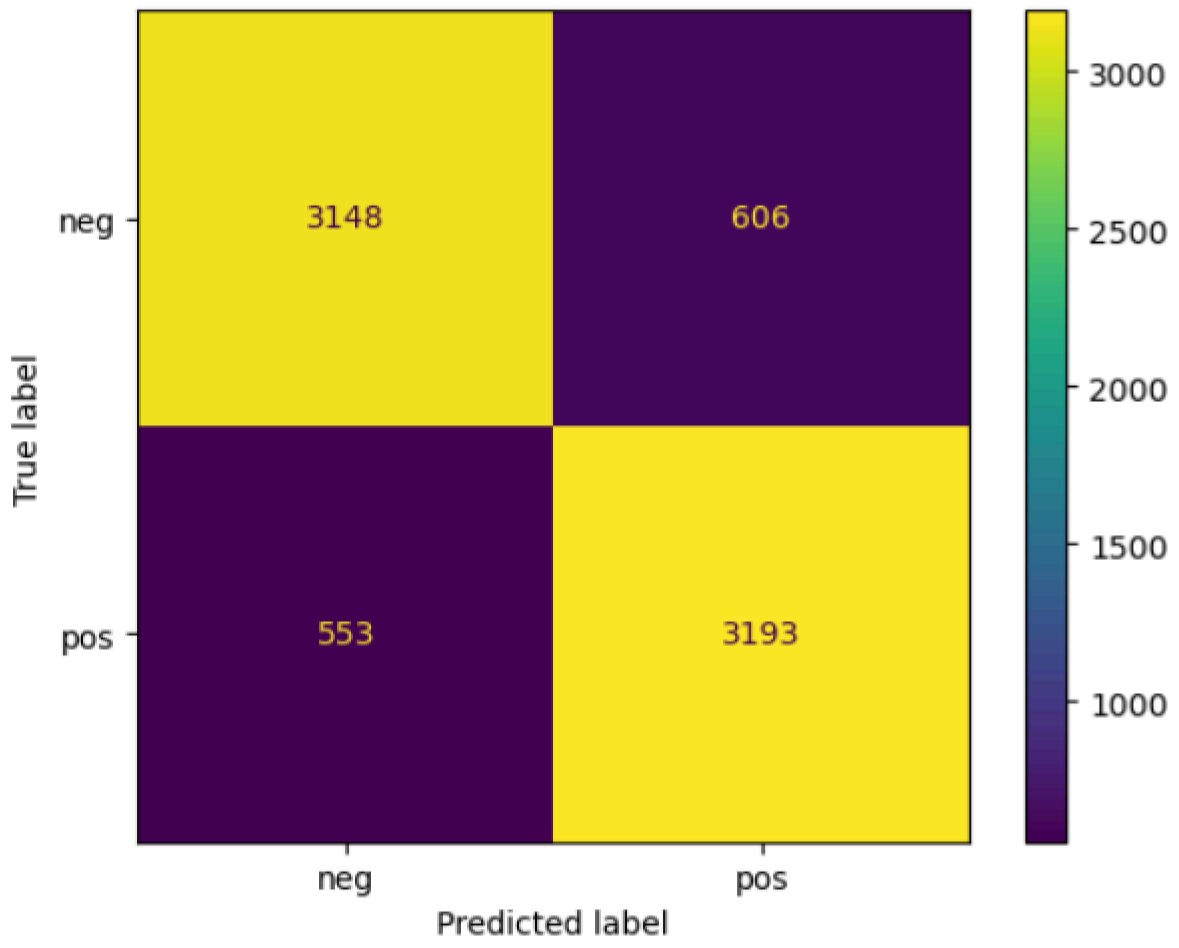
```
Accuracy:

0.8548
```

```
The Model 2 (SVM with Linear SVC) confusion matrix is:
```



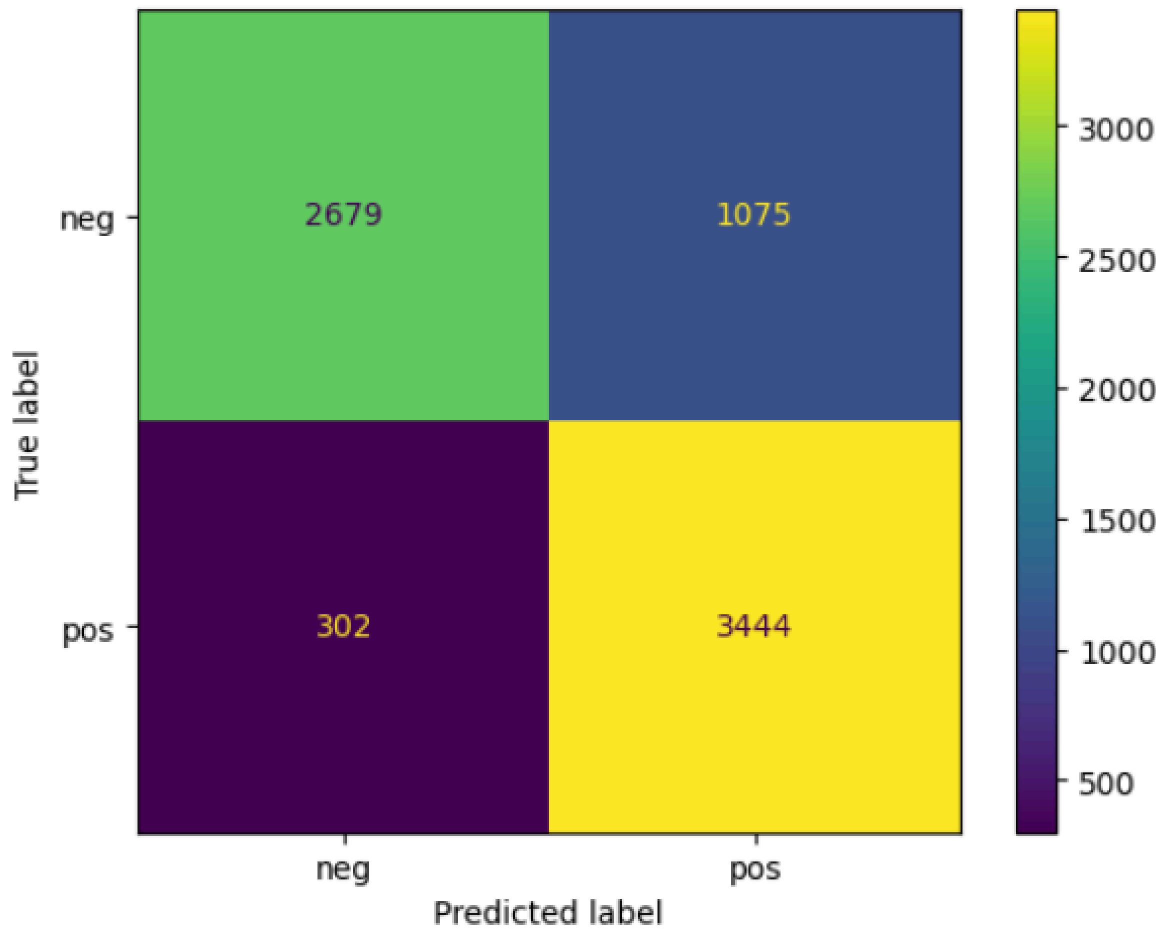
```
Accuracy:
0.8670666666666667
```

```
The Model 3 (SVM with Polynomial kernel) confusion matrix is:
```



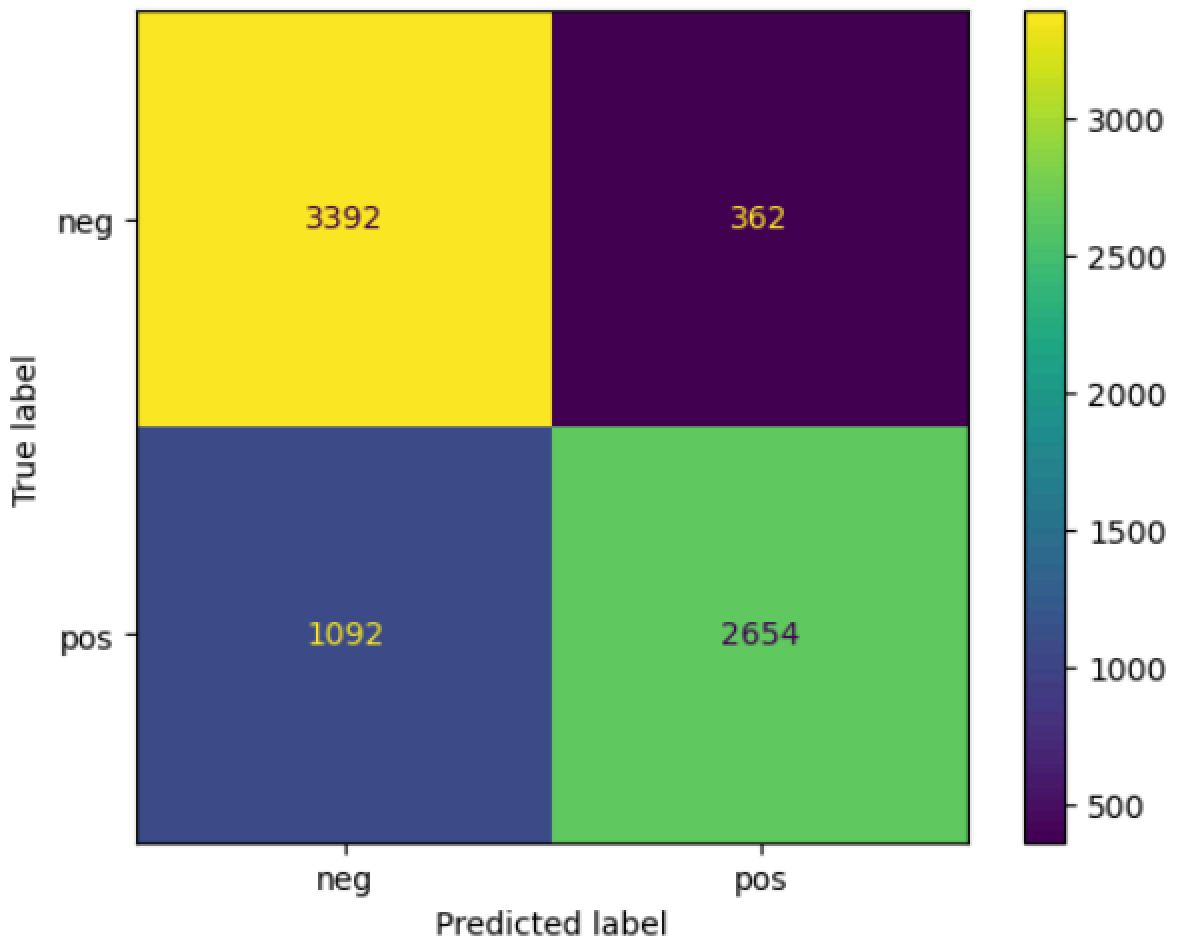
```
Accuracy:

0.8454666666666667
```

```
The Model 4 (SVM with RBF kernel) confusion matrix is:
```



```
Accuracy:

0.8164
```

```
The Model 5 (SVM with SGD Classifier) confusion matrix is:
```



```
Accuracy:

0.8061333333333334
```

Summary Results

Model	Algorithm	Data scaling	Training time	Accuracy
1	Naive Bayes	None	11 seconds	85.5%
2	Linear SVC SVM	MinMaxScaler	4.58 hours	**86.7%**
3	SVM with polynomial kernel	MinMaxScaler	3.58 hours	84.5%
4	SVM with RBF kernel	MinMaxScaler	9.75 hours	81.6%
5	SVM with SGD classifier	StandardScaler	2.5 minutes	80.6%

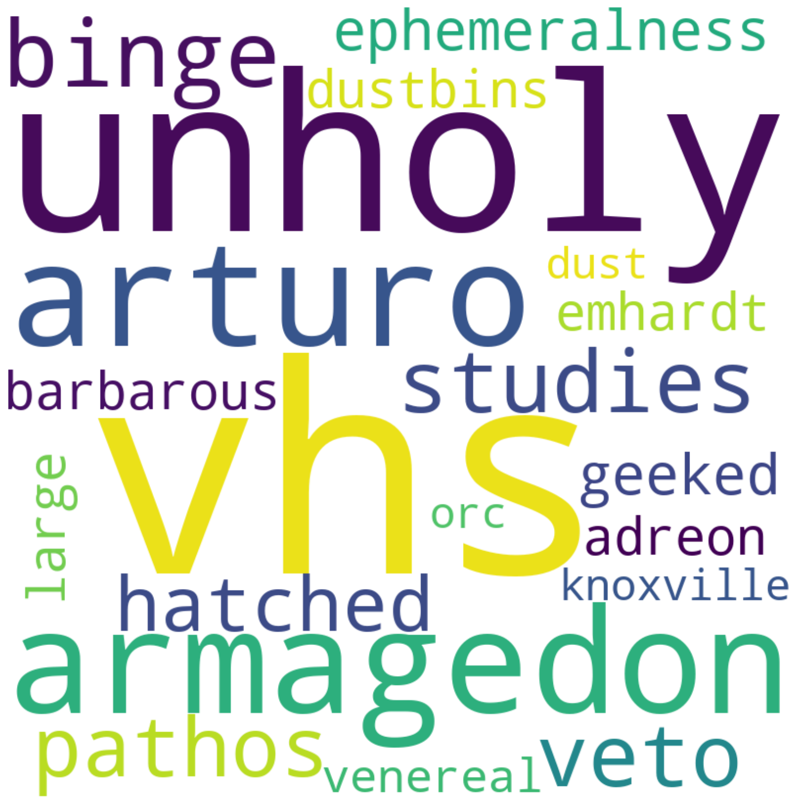
Overall, the model that performed the best was model 2, with 86.7% accuracy. Linear SVC models are well suited to large and complex datasets, however, much work could be done to try to improve the model accuracy. For example, the hyperparameters could be further tuned to optimize training time and performance, such as adjusting the cost, max iterations, and the type of scaling to use, whether MinMaxScaler, StandardScaler, or preprocessing.scale(), for example. We could also return to the data preprocessing and cleaning steps to further reduce dimensionality by dropping unnecessary or unuseful columns from the dataset. This step alone could drastically improve model training time and accuracy. However, due to excessive training times from multiple models, I did not have enough time to explore these tuning steps in greater detail. As such, much more work could be done in this area to find the optimum model.

Feature Importance

The feature importance determines what are the most important words or features that each model uses to make decisions. What follows is the feature importance for the first two models.

```
MODEL 1
Top 10 most important words for negative class:
movie
film
like
just
good
bad
time
really
don
story

Top 10 most important words for positive class:
film
movie
like
good
just
story
time
great
really
people
```

```
MODEL 2
Top 10 most important words for negative class:
['winterbolt' 'vonbraun' 'arkin' 'artistical' 'bitching' 'tashed'
 'winningham' 'plasticy' 'hinders' 'exaturated']
Top 10 most important words for positive class:
['geeked' 'barbarous' 'emhardt' 'large' 'adreon' 'dustbins' 'venereal'
 'knoxville' 'dust' 'orc']
```



I was unable to create feature importance lists for the remaining models because creating the permutation calculations for the non-linear SVM models required too much processing power and as a result, my kernel froze up when trying to run the last few cells. However, we have a pretty good sense of feature importance from the first two models.

CONCLUSIONS

Our analysis has demonstrated the power of machine learning in accurately gauging audience sentiment. By leveraging models such as Naive Bayes and Support Vector Machines, we have achieved high levels of accuracy in classifying movie reviews as positive or negative. This capability enables businesses to swiftly identify trends and patterns in audience reactions, allowing them to respond promptly to emerging issues or capitalize on positive feedback.

Furthermore, our exploration into feature importance has revealed the most influential words and features driving audience perceptions of films. Understanding which words carry the most weight in shaping sentiment empowers businesses to tailor their marketing campaigns, promotional materials, and even storyline elements to resonate more deeply with their target audience. By aligning messaging and content with the preferences and emotions of moviegoers, businesses can enhance engagement, build stronger connections with audiences, and ultimately drive box office success.

Moreover, the scalability and efficiency of machine learning models offer significant advantages for businesses looking to streamline their operations and optimize resource allocation. By automating the process of sentiment analysis, movie studios and distributors can analyze vast volumes of reviews in real-time, enabling them to stay abreast of audience reactions and market trends. This agility allows businesses to make data-driven decisions swiftly, seize opportunities, and mitigate risks in an ever-evolving industry landscape.

In conclusion, sentiment analysis of movie reviews using machine learning holds immense potential for businesses seeking to thrive in the competitive film industry. By harnessing the insights gleaned from our analysis, businesses can refine their strategies, enhance audience engagement, and drive success in the dynamic world of cinema. Embracing the power of data-driven decision-making, businesses can unlock new opportunities, captivate audiences, and chart a course towards sustained growth and prosperity in the digital age.