Tom Booth

TBooth3@UCLAN.ac.UK Tom.Booth@BAESystems.com

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

Artificial Intelligence Report

IMDB Movie Dataset – Sentiment Analysis: Review Categorisation

# Table of Contents

[1 Table of Contents 1](#_Toc126190114)

[2 Introduction: 2](#_Toc126190115)

[3 Background Reading: 2](#_Toc126190116)

[4 Data: 2](#_Toc126190117)

[4.1 What is the Data type? 2](#_Toc126190118)

[4.2 Where Did the Data Come from? 2](#_Toc126190119)

[4.3 How Much Data Is there? 2](#_Toc126190120)

[4.4 Data Pre-processing Techniques: 2](#_Toc126190121)

[4.4.1 Data Pre-Processing: HTML Stripping 2](#_Toc126190122)

[4.4.2 Data Cleaning 3](#_Toc126190123)

[4.4.3 Lemmatising vs Stemming 3](#_Toc126190124)

[4.4.4 Lowercase 3](#_Toc126190125)

[4.4.5 Removing Stop-Words 3](#_Toc126190126)

[4.4.6 Remove Unneeded Spaces 3](#_Toc126190127)

[4.5 Is New Data Required? 3](#_Toc126190128)

[5 Model Development: 4](#_Toc126190129)

[5.1 Evaluating Approach: Naïve Bayes vs Linear Support Vector Machine 4](#_Toc126190130)

[5.2 Why Linear SVM was better: 4](#_Toc126190131)

[5.3 Alternatives Considered: 4](#_Toc126190132)

[5.4 Data Split: Test Train 4](#_Toc126190133)

[6 Model Evaluation: 5](#_Toc126190134)

[6.1 Classification Reports 5](#_Toc126190135)

[6.2 Confusion Matrix 5](#_Toc126190136)

[6.3 Accuracy and F1 Score 6](#_Toc126190137)

[6.4 Testing the Model – Sample Inputs 6](#_Toc126190138)

[6.5 McNemar Test 6](#_Toc126190139)

[6.6 Common Failure Points – Ambiguous Terminology 7](#_Toc126190140)

[7 Conclusion: 7](#_Toc126190141)

[8 Supplementary Material: 7](#_Toc126190142)

[9 References: 7](#_Toc126190143)

# Introduction:

This report aims to explain the process the author has taken to analyse the IMDB Movie Dataset, **Link to Dataset**, and the approach they have taken to develop and utilise a sentiment analysis to categorise movie reviews as either positive or negative, using the dataset split between a training and testing divide, allowing for high accuracy of prediction. 90% Accuracy. Aiming to provide a clear indication as to the review’s direction.

# Background Reading:

This report has utilised the available recourses online to investigate sentiment analysis and its uses within model development and creation. When determining which analysis types and algorithms to utilise for the model’s development, the author utilised published articles and material to create an estimation of approaches and algorithms to test.

(Medhat et al., 2014) Discusses how the Naïve Bayes Classifier is one of the simplest and most utilised classifiers, “computes the posterior probability of a class based on the distribution of words in the document” This accurately describes the approach to this dataset; there are several reviews to be analysed that contain various keywords that can be used to determine and signify either a positive or negative review. This approach can be used to identify the number of either portion of words. Attaching a determination to the review.

(Rish, 2001) mentions that the Naïve Bayes classifier is ‘surprisingly effective’, as its classification decisions may often be correct even if its probability estimates are ‘inaccurate’; this report, therefore, suggests the introduction of a second method of analysis, introducing a different approach potentially yielding better and more accurate results.

(Medhat et al.) discusses the various analysis methods, algorithms, and approaches, mentioning that Linear Classifiers, specifically Support Vector Machine classifiers, are good ways to determine ‘linear separators’ in the search space. For the dataset this report is analysing, there is a clear difference between positive and negative reviews. They discuss how “text data is ideally suited for SVM” (Support Vector Machines).

# Data:

## What is the Data type?

The dataset utilised within this model is text data specifically formatted to contain ‘Review’ and ‘Sentiment’ Review being either Positive or Negative, and Sentiment being a written review of the movie. This can be easily utilised within a Python Data frame as the columns Review and Sentiment can be easily accessed and utilised. The Data is already somewhat processed and formatted. Further processing is required; however, the dataset is already relatively ‘clean’ of inaccuracies.

## Where Did the Data Come from?

The data was retrieved from Kaggle, a subsidiary of Google, an online community of Data Scientists allowing for the sharing of datasets and learning. The dataset was retrieved from Kaggle directly from the following link: [https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.](https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews) User Lakshmipathi N has uploaded the dataset for utilisation within the data science and artificial intelligence field. The dataset is free and available to use without any additional licence.

## How Much Data Is there?

The data is a collection of movie reviews from IMDB, Internet Movie Database; the dataset contains 25,000 Positive and a further 25,000 Negative reviews for movies; these reviews will be utilised and split into a 70/30 data split between training and testing. Allowing a model to be developed and trained.

## Data Pre-processing Techniques:

The following section will describe the techniques used to prepare the dataset for analysis, using Python modules and libraries to improve and sanitise the dataset.

### Data Pre-Processing: HTML Stripping

As the dataset has been scraped (Retrieved from a website, IMDB in this case, through the process of Copying the text and HTML available on the webpage), There are HTML tags throughout the text, for example, the recurrence of <br> indicating a broken line for paragraph spacing. This is unnecessary and will cause issues within our dataset when analysing the content. Therefore, the author has chosen to utilise Beautiful Soup, a Python library utilised to parse through data and remove any HTML and/or XML data. This could have been completed manually. However, BS provides a fantastic resource for quickly and accurately removing HTML/XML data from the text. For this dataset, the HTML Parser has been utilised; XML was not needed as there isn’t any to remove.

### Data Cleaning

Further data cleaning was required once the HTML tags had been removed. There was still the occurrence of data having surrounding backslashes and speech marks. For example, \’ dream’ \ was identified throughout the dataset. An additional Python module was utilised to remove this and other occurrences; regular Expressions allow for the search and replacement of specific strings and patterns within the data. Regular expressions are an incredibly powerful tool and allow for the removal of all slashes and other non-alphabetical characters, substituting them for their leftmost occurrence.

### Lemmatizing vs Stemming

To further improve the data quality, the dataset was Lemmatised; Initially, Lemmatising and Stemming were both reviewed and compared. However, Stemming appeared to reduce the visual accuracy of the dataset, removing the ending of certain words and leaving them uncomplete, whereas Lemmatizing left a complete and accurate dataset; therefore, lemmatising was chosen and utilised to group the words allowing analysis of groups, ensuring that the words with similar or the same meaning are categorised as one. Improving the model's accuracy as the words are the ‘same’ and regarded as such.

### Lowercase

The dataset was then converted to be all lowercase, removing any capitalisation of words; this helps in the later stages of analysis and model development. It is not as important as the previous steps but is useful.

### Removing Stop-Words

Utilising the NLTK (Natural Language Toolkit) framework, we can remove any stop words from the dataset; these are frequent throughout the dataset and don’t provide any real benefit or insight into the dataset. Therefore, they can be removed to clean up the dataset and reduce its size. Allowing us to view and analyse the more specific words and the data we are interested in.

### Remove Unneeded Spaces

Further removal of unnecessary spaces allows the dataset to be cleaned up; through the prior cleaning stages, there will be double spaces throughout the document; these can be easily removed by joining the spaces together.

## Is New Data Required?

There is an even distribution between the data; there is not an imbalance between Positive or Negative reviews, and there is a perfect balance between the two, allowing for accurate and fair results; therefore, there is no requirement to find additional data for either side, Positive or Negative.

# Model Development:

Discuss your approach to solving the given problems. Why is your approach the right thing to do? Did you consider alternative approaches? You should demonstrate that you have applied ideas and skills built up during the semester to tackling the given problem. It may be helpful to include figures, diagrams, or tables to describe your method.

## Evaluating Approach: Naïve Bayes vs Linear Support Vector Machine

Naive Bayes and Linear SVM are two popular machine learning algorithms. Naive Bayes is a probabilistic classifier based on the Bayes theorem with an assumption of independence among features. Linear SVM is a supervised learning algorithm that uses a linear kernel to classify data. Naive Bayes is generally faster to train and predict than linear SVM and is good for tasks with high dimensional data. On the other hand, Linear SVM is more powerful and can handle more complex data sets. It is also more accurate than Naive Bayes in most cases. However, Linear SVM requires more time to train and predict than Naive Bayes. In terms of accuracy, Linear SVM is generally more accurate than Naive Bayes. However, Naive Bayes has the advantage of being simpler and faster to train and predict. However, in this current scenario, there is no requirement for quick model development. The pursuit of accuracy was more important than speed. Therefore, this report believes that the discussed approach in utilising a Linear SVM classification to develop the model is the correct approach and will provide the best possible results with the given data; there is no requirement to complete it quickly; therefore, there is the luxury and affordability of time, allowing a better and more accurate model to be developed and utilised within the evaluation.

## Why Linear SVM was better:

Linear SVM proved to provide higher accuracy than Naïve Bayes when compared; the accuracy was 0.90288 compared to 0.88992; a relatively small difference when considered, however, enough to justify the utilisation of Linear SVM over Naïve Bayes. When comparing the two options together, the small difference isn’t noticeable within the actual completed model; both models included within the Jupyter file provide the same results in testing. Therefore, either option could have been chosen; however, in pursuit of higher accuracy for the model development, aiming to achieve better results, Linear SVM is applicable.

## Alternatives Considered:

As mentioned above, Naïve Bayes was heavily considered for the data mode; however, after initial testing proved to be less accurate. Despite its ability to train models quicker, this was not a requirement for choosing the ‘correct’ model. When compared to Linear SVM, it wasn’t a better choice.

Logistic Regression: A supervised learning algorithm that is often used in text classification tasks due to its ability to handle high-dimensional data. Linear SVM is better than logistic regression for text data because it can handle a higher number of features more effectively than logistic regression. It is also more robust to outliers and can produce better results in terms of accuracy, precision, and recall.

K-Nearest Neighbours - Knn: It is a powerful tool for text classification tasks due to its ability to capture complex relationships between features. Linear SVM is better than logistic regression for text data because it can handle a higher number of features more effectively than logistic regression. Knn is significantly more computationally demanding. However, Linear SVM is regarded as being easier to interpret and more visually identifiable; Knn can find more complex patterns. However, this was not applicable to this dataset.

## Data Split: Test Train

The 70/30 data split is a commonly used technique for dividing a dataset into separate training and testing components. The split usually involves randomly separating the data into two sets, with 70% of the data used for training the model and the remaining 30% used for testing the trained model. This partitioning of the dataset allows for a more accurate assessment of the model’s performance by testing it on data that it has not seen before. The 70/30 split is advantageous in many cases, as it allows for a balanced evaluation of the model that is not overly biased in the direction of the training data.

Furthermore, it allows for a more accurate evaluation of the model’s generalisation performance, as the model must perform well on data it has not seen before. In addition, the 70/30 split is advantageous because it allows for more efficient use of the available data. By using a smaller portion of the data for testing, more of the data can be used for training, which can lead to improved model performance. Overall, the 70/30 data split is an effective way to partition a dataset into training and testing components. It provides a balanced evaluation of the model’s performance, allows for a more accurate assessment of the model’s generalisation performance, and is more efficient with the available data.

# Model Evaluation:

To properly assess the model and evaluate its performance, this report will utilise metrics developed within the program; these metrics will allow this report and its author to draw conclusions and assessments based on the information present.

## Classification Reports

The Linear SVM model classification, when comparing the Actual and Predicted data together with the target names of Positive and Negative, the classification report reveals an accuracy score of 0.90, a precision score of 0.91 for Negative and 0.89 for Positive, suggesting that the model is leaning towards a preference of Negative results.

## Confusion Matrix

When analysing the confusion matrix for Linear SVM, we can clearly see in the bellow screenshot below that there is a high level of accuracy, there are very high numbers of True Positive and True Negative, with very few, for the dataset and its size, numbers of False Positive and False Negative. This clearly shows the performance and ability of the model and how it can properly predict and analyse the given data.

The high number of True Positives and Negatives identifies that the model can properly categorise and classify data; the data that has been classified does belong to that class and has not been incorrectly classified. The model is, therefore, successful in its task of being able to analyse the data and determine and predict a review’ tone – positive or negative’ through utilising this model.

## Accuracy and F1 Score

The accuracy of the model, 0.90, clearly shows the ability of the model to predict data in an accurate and correct manner; it can determine the response of a review based on the text content and is able to correctly assign a value to it. The accuracy metric has been calculated as a percentage of the correctly classified instances;

However, (Dj Novakovi et al., 2017) discuss how “Accuracy neglects the differences between the types of errors”; therefore, this report has utilised and included an F1 score.

The F1 score is calculated as the mean of precision and recall. The F1 score takes both false positives and false negatives into account. The F1 score is used to measure the accuracy of the model, measuring the balance between precision and recall. Precision is a fraction of true positives that are correctly identified, while recall is the fraction of actual positives that are correctly identified. The F1 score is a better measure of the overall accuracy of the model than either precision or recall alone. The F1 score for this model is 0.90 Negative, 0.91 Positive and 0.90 Accuracy. The recall is defined as the “fraction of examples which were predicted to belong to a class” (Jordan, 2017) Precision is defined as the fraction of relevant examples among all examples predicted.

## Testing the Model – Sample Inputs

The model has been further tested through the use of manual sample inputs; the model has been given strings to evaluate and provide a determination on; these are regarded as sample ‘Reviews’ and aim to test the model and its ability to predict the ‘tone’ either Positive or Negative, for example, a review reading ‘Started good, by the time it ended, it was rubbish’ would be expected to return ‘Negative Review’ Indicating that the model is able to determine the difference between a ‘Good’ and ‘Rubbish’ identifying that the review is not a ‘Positive’ through the use of a Positive word such as ‘Good.’

As expected, the model performs well.

## McNemar Test

To evaluate the chosen model against the next best model available, the author has utilised the McNemar test to understand and identify the performance between the two classifiers. To calculate the values for the McNemar test, the total correct values and total incorrect values, True Positive False Positive True Negative, and False Negative values were added together for each classifier, leaving the Correct total values and Incorrect Values.

(Raschka, 2018) discusses how the McNemar test allows us to determine how different two classifiers are and if they have any difference or disparity in performance. “In McNemar’s Test, we formulate the null hypothesis that the probabilities p(B) and p(C) – where B and C refer to the confusion matrix cells introduced in an earlier figure – are the same, or in simplified terms: None of the two models performs better than the other. Thus, we might consider the alternative hypothesis that the performances of the two models are not equal.”

For the Linear SVM Model, these were, Correct: 11,286 Incorrect: 1,214

For the Naïve Bayes Model, these were, Correct: 11,124 Incorrect: 1,376

When running the McNemar test, the p-value comes back below 0.5, meaning that the Null-Hypothesis is rejected, and the models are statistically significantly different; the conclusion is that the Linear SVM model performs significantly better than the Naïve Bayes model.

## Common Failure Points – Ambiguous Terminology

Despite the model’s high accuracy, recall, precision and F1 score, the model still has room for improvement; the model is currently unable to determine ambiguous phrases; a phrase such as ‘Kinda good’ returns a Negative Review, this could be argued as a ‘Positive’ review as the phase suggests that the movie is in fact, good, however, maybe not great. Therefore, this report believes that the model fails where there is ambiguity and judgement as to the perspective of the review required. Given a larger training set with some potential training material handling ambiguous statements, this report believes that the model would be able to handle this level of text, being able to accurately determine a result.

Given the possibility, the model could be developed to handle these phrases; when faced with a sentence that is not clear as to the review’s intention, as shown above, the model defaults to a Negative review, even when it is possible that the review could be taken as positive and is more obviously positive on reflection. (Deng et al., 2017) state, “The semantic ambiguity (i.e., polysemy) of single words and the sparsity of phrases negatively affect the robustness of sentiment analysis, especially in the context of short social media texts.” If given more time to further develop this model, this report would aim to utilise published articles such as this, incorporating the use of dependency features in supervised sentiment analysis to further improve the model's capability at understanding and handling ambiguity within sentences; this would in theory, provide more context to the sentence and review, allowing it to make a more accurate determination as to the content of the review and its Positive/Negative status.

# Conclusion:

Throughout this experiment, the model and machine learning platform has been developed to handle the large text-based data set, allowing it to predict the outcome of specific inputted and tested reviews. It is quite accurate for data that is unambiguous and clear in its indicated tone, either Positive or Negative. However, it is unable to determine that it is ambiguous in its totality and does not have a clearly defined preference; these words could be handled with the inclusion of additional libraries. However, this would be an advancement of the current system and is not something included at this time. This report advocates for the further development of the model to include this feature.

During the development of this model, the author has gained many new skills, utilising new libraries to further expand knowledge, understanding and skillset, adding utilities to be able to utilise in further work. For example, the author had not used Beautiful Soup or Regex within a Machine learning application; Regex has been utilised previously but not within this environment; this was a good opportunity to develop these skills and further the author's understanding.

# Supplementary Material:

Screenshots:

Figure 1: Classification Reports

Calendar

Description automatically generated

Figure 2: Confusion Matrix

Chart, treemap chart

Description automatically generated

Figure 3: Test Inputs

Graphical user interface, text

Description automatically generated

Figure 4: Test Inputs

Graphical user interface, text, application

Description automatically generated

Figure 5: McNemar Test

Text

Description automatically generated

Source Code: [Github – Source Code Link](https://github.com/Boothey07/AIAssignment)

#%%  
import os  
import pickle  
  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
  
  
from bs4 import BeautifulSoup  
import re  
import nltk  
from nltk.corpus import stopwords  
from nltk.stem import WordNetLemmatizer  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.svm import LinearSVC  
%matplotlib inline  
  
from sklearn import \*  
  
import pickle  
import seaborn as sns  
  
  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
  
from sklearn.feature\_selection import SelectKBest, chi2  
  
#%%  
dataset = pd.read\_csv("IMDB Dataset.csv")  
dataset.head()  
#%%  
dataset.describe()  
#%% md  
  
#%%  
dataset.info()  
#%%  
dataset['sentiment'].unique()  
#%%  
dataset['sentiment'].value\_counts()  
#%%  
sns.countplot(dataset['sentiment'])  
#%%  
review = dataset['review'].loc[1]  
review  
#%%  
soup = BeautifulSoup(review,"html.parser")  
review = soup.get\_text()  
review  
#%%  
review = re.sub('\[[^]]\*\]',' ',review)  
review = re.sub('[^a-zA-Z]',' ',review)  
review  
#%% md  
  
#%%  
review = review.lower()  
review  
#%%  
review = review.split()  
review  
#%%  
nltk.download('stopwords')  
review = [word for word in review if not word in set(stopwords.words('english'))]  
review  
#%%  
nltk.download('wordnet')  
nltk.download('omw-1.4')  
lem = WordNetLemmatizer()  
review = [lem.lemmatize(word) for word in review]  
review  
#%%  
review = ' '.join(review)  
review  
#%%  
corpus = []  
corpus.append(review)  
#%%  
countVec = CountVectorizer()  
reviewCountVec = countVec.fit\_transform(corpus)  
reviewCountVec.toarray()  
#%%  
countVecBinary = CountVectorizer(binary=True)  
reviewCountVecBinary = countVecBinary.fit\_transform(corpus)  
reviewCountVecBinary.toarray()  
#%%  
tfidVec = TfidfVectorizer()  
reviewTfidVec = tfidVec.fit\_transform(corpus)  
reviewTfidVec.toarray()  
#%%  
dataset\_train, dataset\_test, train\_label, test\_label = train\_test\_split(dataset['review'],dataset['sentiment'],test\_size=0.25,random\_state=42)  
#%%  
train\_label = (train\_label.replace({'positive':1,'negative':0})).values  
test\_label = (test\_label.replace({'positive':1,'negative':0})).values  
#%%  
  
if os.path.isfile('corpusTrain.csv') :  
 print ('File Found')  
  
 corpusTrain = pd.read\_csv('corpusTrain.csv')  
 corpusTest = pd.read\_csv('corpusTest.csv')  
 corpusTrain = corpusTrain["0"].tolist()  
 corpusTest = corpusTest["0"].tolist()  
  
else:  
 print ('Not Found')  
 corpusTrain = []  
 corpusTest = []  
  
 for i in range(dataset\_train.shape[0]):  
 soup = BeautifulSoup(dataset\_train.iloc[i],"html.parser")  
 review = soup.get\_text()  
 review = re.sub('\[[^]]\*\]',' ',review)  
 review = re.sub('[^a-zA-Z]',' ',review)  
 review = review.lower()  
 review = review.split()  
 review = [word for word in review if not word in set(stopwords.words('english'))]  
 lem = WordNetLemmatizer()  
 review = [lem.lemmatize(word) for word in review]  
 review = ' '.join(review)  
 corpusTrain.append(review)  
 print(i)  
  
 for j in range(dataset\_test.shape[0]):  
 soup = BeautifulSoup(dataset\_test.iloc[j],"html.parser")  
 review = soup.get\_text()  
 review = re.sub('\[[^]]\*\]',' ',review)  
 review = re.sub('[^a-zA-Z]',' ',review)  
 review = review.lower()  
 review = review.split()  
 review = [word for word in review if not word in set(stopwords.words('english'))]  
 lem = WordNetLemmatizer()  
 review = [lem.lemmatize(word) for word in review]  
 review = ' '.join(review)  
 corpusTest.append(review)  
 print(j)  
  
 tmpTest = pd.DataFrame(corpusTest)  
 tmpTest.to\_csv("corpusTest.csv")  
  
 tmpTrain = pd.DataFrame(corpusTrain)  
 tmpTrain.to\_csv("corpusTrain.csv")  
  
#%%  
corpusTrain[-1]  
#%%  
corpusTest[-1]  
#%%  
tfidVec = TfidfVectorizer(ngram\_range=(1, 3))  
tfidVec\_train = tfidVec.fit\_transform(corpusTrain)  
tfidVec\_test = tfidVec.transform(corpusTest)  
#%%  
#Suppoer Vector Model  
  
linear\_svc = LinearSVC(C=0.5, random\_state=42)  
linear\_svc.fit(tfidVec\_train, train\_label)  
  
predict = linear\_svc.predict(tfidVec\_test)  
#%%  
print("Classification Report: \n", classification\_report(test\_label,predict,target\_names=['Negative','Positive']))  
print("Confusion Matrix: \n", confusion\_matrix(test\_label,predict))  
print("Accuracy: \n", accuracy\_score(test\_label,predict))  
#%%  
conf\_matrix = metrics.confusion\_matrix(y\_true=test\_label, y\_pred=predict)  
  
fig, ax = plt.subplots(figsize=(7.5, 7.5))  
ax.matshow(conf\_matrix, cmap=plt.cm.Blues, alpha=0.3)  
for i in range(conf\_matrix.shape[0]):  
 for j in range(conf\_matrix.shape[1]):  
 ax.text(x=j, y=i,s=conf\_matrix[i, j], va='center', ha='center', size='xx-large')  
  
plt.xlabel('Predictions', fontsize=18)  
plt.ylabel('Actuals', fontsize=18)  
plt.title('Confusion Matrix', fontsize=18)  
plt.show()  
#%%  
from sklearn.naive\_bayes import MultinomialNB  
  
multiNB = MultinomialNB()  
multiNB\_train = tfidVec.fit\_transform(corpusTrain)  
multiNB\_test = tfidVec.transform(corpusTest)  
  
multiNB.fit(multiNB\_train, train\_label)  
  
predict\_NB = multiNB.predict(multiNB\_test)  
#%%  
print("Classification Report: \n",classification\_report(test\_label, predict\_NB,target\_names=['Negative','Positive']))  
print("Confusion Matrix: \n",confusion\_matrix(test\_label,predict\_NB))  
print("Accuracy: \n",accuracy\_score(test\_label,predict\_NB))  
#%%  
conf\_matrix = metrics.confusion\_matrix(y\_true=test\_label, y\_pred=predict\_NB)  
  
fig, ax = plt.subplots(figsize=(7.5, 7.5))  
ax.matshow(conf\_matrix, cmap=plt.cm.Blues, alpha=0.3)  
for i in range(conf\_matrix.shape[0]):  
 for j in range(conf\_matrix.shape[1]):  
 ax.text(x=j, y=i,s=conf\_matrix[i, j], va='center', ha='center', size='xx-large')  
  
plt.xlabel('Predictions', fontsize=18)  
plt.ylabel('Actuals', fontsize=18)  
plt.title('Confusion Matrix', fontsize=18)  
plt.show()  
#%%  
dataset\_predict = dataset\_test.copy()  
dataset\_predict = pd.DataFrame(dataset\_predict)  
dataset\_predict.columns = ['review']  
dataset\_predict = dataset\_predict.reset\_index()  
dataset\_predict = dataset\_predict.drop(['index'],axis=1)  
dataset\_predict.head()  
#%%  
test\_actual\_label = test\_label.copy()  
test\_actual\_label = pd.DataFrame(test\_actual\_label)  
test\_actual\_label.columns = ['sentiment']  
test\_actual\_label['sentiment'] = test\_actual\_label['sentiment'].replace({1: 'positive', 0: 'negative'})  
#%%  
test\_predicted\_label = predict.copy()  
test\_predicted\_label = pd.DataFrame(test\_predicted\_label)  
test\_predicted\_label.columns = ['predicted\_sentiment']  
test\_predicted\_label['predicted\_sentiment'] = test\_predicted\_label['predicted\_sentiment'].replace({1: 'positive', 0: 'negative'})  
#%%  
test\_result = pd.concat([dataset\_predict,test\_actual\_label,test\_predicted\_label],axis=1)  
test\_result.head()  
#%%  
save\_classifier = open("tfidVec.pickle","wb")  
save\_model = open("linearSVC.pickle","wb")  
pickle.dump(tfidVec,save\_classifier)  
pickle.dump(linear\_svc,save\_model) # 1: pos , 0:Neg  
save\_classifier.close()  
save\_model.close()  
#%%  
# Hint  
save\_classifier = open("tfidVec.pickle","rb")  
save\_model = open("linearSVC.pickle","rb")  
save\_cv = pickle.load(save\_classifier)  
model = pickle.load(save\_model)  
save\_classifier.close()  
save\_model.close()  
#%%  
# An example of how to define a function  
def test\_model(sentence):  
 sen = save\_cv.transform([sentence]).toarray()  
 res = model.predict(sen)[0]  
 if res == 1:  
 return 'Positive review'  
 else:  
 return 'Negative review'  
#%%  
# Sample of example test  
sen = 'Started good, by the time it ended, it was rubbish'  
res = test\_model(sen)  
print(res)  
#%%  
sen = "Kinda Good"  
res = test\_model(sen)  
print(res)  
#%%  
nb\_save\_classifier = open("tfidVecNB.pickle","wb")  
nb\_model = open("naivebayes.pickle","wb")  
pickle.dump(tfidVec,nb\_save\_classifier)  
pickle.dump(multiNB,nb\_model) # 1: pos , 0:Neg  
nb\_save\_classifier.close()  
nb\_model.close()  
#%%  
nb\_save\_classifier = open("tfidVecNB.pickle","rb")  
nb\_model = open("naivebayes.pickle","rb")  
save\_nb = pickle.load(nb\_save\_classifier)  
model\_nb = pickle.load(nb\_model)  
nb\_save\_classifier.close()  
nb\_model.close()  
#%%  
# An example of how to define a function  
def test\_model(sentence):  
 sen = save\_nb.transform([sentence]).toarray()  
 res = model\_nb.predict(sen)[0]  
 if res == 1:  
 return 'Positive review'  
 else:  
 return 'Negative review'  
#%%  
# Sample of example test  
sen = 'Waste of time'  
res = test\_model(sen)  
print(res)  
#%%  
from statsmodels.stats.contingency\_tables import mcnemar  
  
#mcnemar test for linear svc model and multinomial naive bayes model  
mcnemar\_test = mcnemar([[11286,1214], [11124,1376]], exact=False, correction=True)  
print('statistic=%.3f, p-value=%.3f' % (mcnemar\_test.statistic, mcnemar\_test.pvalue))  
#%%  
data = [[11286,1214], [11124,1376]]  
  
# calculate mcnemar test  
print(mcnemar(data, exact=True))  
print(" ")  
print(mcnemar(data, exact=False, correction=False))  
#%%

# References:

Deng, S., Sinha, A. P., & Zhao, H. (2017). Resolving Ambiguity in Sentiment Classification. *ACM Transactions on Management Information Systems (TMIS)*, *8*(2–3). https://doi.org/10.1145/3046684

Dj Novakovi, J., Veljovi, A., Ili, S. S., Zeljko Papi, ˇ, & Tomovi, M. (2017). Evaluation of Classification Models in Machine Learning. *Theory and Applications of Mathematics & Computer Science*, *7*(1), 39–46.

Jordan, J. (2017). *Evaluating a machine learning model.* https://www.jeremyjordan.me/evaluating-a-machine-learning-model/

Medhat, W., Hassan, A., & Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, *5*(4), 1093–1113. https://doi.org/10.1016/J.ASEJ.2014.04.011

Raschka, S. (2018). *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*.

Rish, I. (2001). *An empirical study of the naive Bayes classifier*.