NLP Sentiment Analysis Assignment

Table of Contents

Abstract

Sentiment Analysis is used for understanding the emotions of a given text and classifying them either into positive or negative reviews. The main aim of this project is to classify movie reviews from the IMDB dataset [1] as either positive or negative, by using four distinct algorithms: Naïve Bayes (implemented from scratch and using MultinomialNB), Support Vector Classifier (SVC), Logistics Regression, and BERT. For this classification task the steps taken are preprocessing and feature selection, including tokenization, N-grams, stopwords removal, lemmatization, stemming, and TF-IDF calculation. The models were evaluated using a classification report that included accuracy, precision, recall, and F1-score. Results show that the BERT model outperformed all the other algorithms, showing its effectiveness in sentiment analysis for movie reviews. The report gives a detailed explanation of each of the algorithms used and explains the reasons behind BERT's performance. In conclusion, this assignment not only identifies the best-performing model but also suggests some potential for future improvements.

1. Introduction and Motivation

Sentiment Analysis is the process of automatically detecting the sentiment in text (i.e., positive, negative, or neutral) [Spec].

It is an essential tool which is used by everyone, especially companies, to understand sentiments in all types of data. By automatically examining customer input, that is sentiments expressed in a survey, movie reviews or discussions, companies can gain insights of the text written and can understand what the reaction of the customer is. This information helps them to customize the product and services and helps them align with their customer's preferences and requirements [2].

Sentiment Analysis is used by many top companies like Apple, Google, KFC and many more. One of the use cases of Google using sentiment analysis is to make Chrome better. They check on what users want to say, not just words, but also check if it is positive or negative and what they want to say about different parts of Chrome. This helps them see what is good or bad and helps them make Chrome better. [3]

Even though it a great tool for automatically detecting sentiment in a text there are limitations, some of them are [4]: Context sensitive error – trying to express a negative statement but using positive wordings (e.g.: ''Sure, what a brilliant suggestion!''). This phrase can be treated as positive or negative and sentiment analysis might wrongly classify them as positive instead of negative depending on the situation. Multilingual data – although English is the language used worldwide since sentiment analysis is used by many companies worldwide the reviews or feedback received from customers maybe of a different language. Since sentiment analysis is mainly trained in English, some of the data might get lost or mismatched during translations and it will not provide the correct analysis. Emojis – In this century everyone responds or replies by using emojis. Sentiment Analysis is a text-based tool which depends on the text written to classify it into positive or negative. If the text consists of emojis then that text will not be analysed [4]. These are some of the limitations or challenges faced by sentiment analysis.

Different Machine Learning and Deep Learning techniques are used for Sentiment Analysis. Machine Learning techniques uses classification algorithms like Naïve Bayes, Support Vector

Machines (SVMs), Logistic Regression, Linear Regression to do Sentiment Analysis [5]. Deep Learning techniques transform the text into embeddings and some of the algorithms include Long Short-Term Memory (LSTM) and other recurrent neural networks (RNNs), Convolutional Neural Networks (CNNs) and Transformers (BERT and GPT-3) [6,7].

In this project I am testing to see which algorithm is better for sentiment analysis. The IMDB Reviews data [1] have been used for sentiment analysis and have been tested on 4 different algorithms (Naïve Bayes (from scratch and MultinomialNB), Logistic Regression, SVM, and BERT). Different feature selection and feature generation methods were applied on the dataset and different combinations of features were tested on these algorithms.

Different strategies were used and tested to see which works best. For example, Logistic Regression and SVM was first tested using default hyperparameters and then hyperparameter optimizations was done similarly in the case of Naïve Bayes it was tested using the built-in version and scratch version. Out of all the models tested the BERT model performed better than other models.

2. Related Work

There are a several existing methods for Sentiment Analysis some of them include [5,6,7]:

- **Machine Learning methods** like Naïve Bayes [10], Support Vector Machines (SVMs) [12], Random Forests (RFs) [13] and many more.
  - Naïve Bayes (NB) – In this paper [9], Naïve Bayes algorithm was used to do sentiment analysis on Twitter tweets. That is classifying tweets into positive and negative. Multinomial NB is a probabilistic learning method. The goal of this method is to find the best class in a document (Maximum Posterior class (MAP) or most likely class) [10].
  - Support Vector Machines (SVMs) – It is a binary classification algorithm. It works by transforming data into a high-dimensional space and drawing a line to separate them into two groups. It works well when there is a limited amount of data provided to the model [11,12,18].
  - Random Forests (RF) – It is a machine learning method which works based on the principle of several decision trees put together into one to make predictions. It works for both classification and regression tasks. The larger the number of decisions trees put together, the better will be the performance given by Random Forest [13,14].

- **Deep Learning methods** include Long Short-Term Memory (LSTM) [15], Recurrent Neural Networks (RNNs) [16], and Transformer based models like BERT [17] and many more.
  - Long Short-Term Memory (LSTM) – It is a type of neural network used in Deep Learning. It is great at learning from sequences of data, like speech recognition and translation. It is a special kind of Recurrent Neural Network (RNN) that works well for different tasks. There are two problems with RNNs vanishing gradient and exploding gradient, but these two problems are solved by LSTMs [15,19]
  - Recurrent Neural Networks (RNNs) – It is a type of artificial neural network which uses sequential data. Unlike other methods, RNN remember what they have seen before and use that to understand what is happening now. They cannot predict the future

because they only look at the past. It is used in translation, speech recognition and voice assistants [16, 20].

- BERT (Bidirectional Encoder Representations from Transformers) – "It is transformers based Deep Learning Model where each output element is linked to the input element and the weights are calculated dynamically between them." [21,17].
- VADER – A Rule-based model for Sentiment Analysis of social media Text. This model was compared with eleven other methods (i.e., LIWC, ANEW, the General Inquirer, SentiWordNet, and machine learning oriented techniques relying on Naive Bayes, Maximum Entropy, and Support Vector Machine (SVM) algorithms) and it was found that VADER works better than most of these models and performed better than humans who had classified these texts manually. Two strategies were used in this method, first they made a list of words on how people feel in the online text messages and figured out the strength of these words in showing those feelings. Then made five simple rules on how these words are used to show feelings [8].

3. Experiment and Results

In the CodeA folder there are three .ipynb files, three .csv files for BERT and data folder with pos and neg reviews:

1. Codes for experiment and results of Naïve Bayes (scratch), MultinomialNB, SVCs and Logistic Regression can be found in **CodeA – Part1.ipynb**
2. Code for experimenting different features can be found in **Feature Test.ipynb**
3. Code for BERT model can be found in **BERT.ipynb**

3.1 Preprocessing and Feature Selection:

These are the list of libraries which have been imported for running each of the techniques mentioned below:

```
#Libraries

import numpy as np
import pandas as pd
import nltk
from nltk import ngrams

import string
import math

nltk.download('stopwords')
nltk.download('wordnet')

stoplist = set(nltk.corpus.stopwords.words('english'))

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.lancaster import LancasterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
```

Figure 1: List of Libraries

Preprocessing and Feature Selection of the data are the important steps for carrying out any NLP tasks. The different ways are [22]:

- Tokenization
- N-Grams
- Stopwords
- Lemmatization
- Stemming

3.1.1 Preprocessing

1. **Tokenization**[1] - is the process of dividing continuous textual information into words, phrases, sentences, symbols, and other components known as tokens [22]. In the data provided, I have converted the whole 'Reviews' into tokens.

```
#Tokenization - Data
from nltk.tokenize import word_tokenize

tokenized = []

for i in df.index:
    tokenized.append(word_tokenize(df['reviews'][i].lower()))
    #tokenized.sort()

df.insert(3,'tokenized_review', tokenized,True)
df.head()
```

Figure 2: Tokenization of Reviews

The tool used for tokenization is - from nltk.tokenize import word_tokenizer

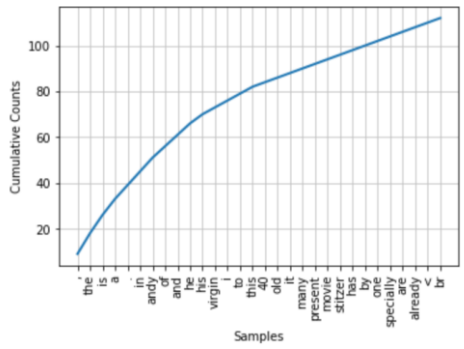
2. **Frequency Distribution**[2] – This was used to check the frequency of words in a particular document. I have tested to see which words are frequent in two of the reviews text files and plotted a graph to see the frequencies. From the graph it is evident that Stopwords and Punctuation are more frequent in two of the text files. This means even though they are more frequent they do not carry any useful meanings. So, they have been removed in the Feature Selection part.

```
frequency_words = nltk.FreqDist(tokenized[0])
print(frequency_words.most_common(50))
```

[1] https://github.com/KuWathsala/NLP-Sentiment-Analysis-Beginers-/blob/master/NLP_SA.ipynb
https://colab.research.google.com/drive/1C5RLXOvaKgbdmeW4PPjg_Q4xooX-r4Oz
https://colab.research.google.com/drive/1WJ2HEQtwpq7TsGRyV5E1tO6IFGnCbW9A?usp=sharing

[2] https://colab.research.google.com/drive/1mkYrOnFt60iDzcH0bhOIP__Us0dvDmlt?usp=sharing#scrollTo=TjTSsQxCvNIF

```
frequency_words.plot(30,cumulative=True)
```




```
<AxesSubplot:xlabel='Samples', ylabel='Cumulative Counts'>
```

Figure 3: Checking Frequency Distribution of Tokenized first document.

The tool used for Frequency Distribution – nltk.FreqDist()

3. **N-Grams**[3] – It refers to the series on N-words. There are different types of N-Grams that are Unigram, Bigrams, Trigrams and more. In N-1 word contexts, N-gram models predict the most frequently appearing words that come after the given sentence. This model is a probabilistic model trained on several texts [23]. For example, sentence -' This is an assignment.'
    1. Unigram = ['This', 'is', 'an', 'assignment']
    2. Bigrams = ['This is', 'is an', 'an assignment']
    3. Trigrams = ['This is an 'is an assignment' '… …']
    4. Similarly, this applies for other grams as well.

```
from nltk import ngrams
terms = list(ngrams(doc[0].split(), n))
```

Figure 4: Applying N-grams on the reviews (features /terms)

The tool used for N-Grams is - from nltk import ngrams.

On this movie review dataset, I have tested using Unigram, Bigrams and Trigrams. All the grams were applied on the tokenized reviews.

4. **Stopwords**[4] **–** It is a collection of frequently used words in the language. That is, 'a', 'is', 'an', 'the' and many more. These words are often removed because they usually carry a limited amount of information [24].

    The tool used for Stopwords is - from nltk.corpus import stopwords.

[3] https://spotintelligence.com/2023/04/05/n-grams/

[4] https://colab.research.google.com/drive/1IsqqtoEVG8n21tb0tUMCN0mLqiBbgnJ8 and from previous Coursework done in Machine Learning.

5. **Stemming**[5] – It is a text normalization method used in NLP, which converts the text to its base form. For example, 'rain' will be stem for 'raining', 'rained', 'rains [25]. It removes the final characters of a word, which can sometimes result in inaccuracies in meanings and spellings (i.e., for 'caring' the stemmed word will be 'car'). Stemming is usually used when there is large amount of data where performance is a concern [26].

```
# Apply stemming
st = LancasterStemmer()
stemmed_documents_1 = [[' '.join([st.stem(word) for word in words])] for words in documents]
```

Figure 5: Applying Stemming to the words in the documents.

The tool used for Stemming is - from nltk.stem.lancaster import LancasterStemmer.

6. **Lemmatization**[6] - It is also a text normalization method used in NLP and works almost like Stemming. It considers the context and transforms the word into its base form known as 'lemma'. (i.e., for 'caring' the lemma would be correctly identified as 'care'). It is computationally very expensive since it has lookup tables and other features [26].

```
# Apply Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_documents_1 = [[' '.join([lemmatizer.lemmatize(word) for word in words if word not in
                                     stoplist and word not in string.punctuation])] for words in documents]
```

Figure 6: Applying Lemmatization to the words in the documents

The tool used for Lemmatization is - from nltk.stem import WordNetLemmatizer.

7. **TF-IDF** [7] **–** A statistical method which is used to evaluate the relevancy of a word is to a document in the collections of documents. The TF_IDF of a word is calculated by multiplying the Term Frequency of word in the document and Inverse Document Frequency of word across a set of documents [27].

Why TF-IDF was chosen over other?

There are three different kinds of Normalization techniques: TF-IDF, Term Frequency and PPMI (Positive Pointwise Mutual Information).

Out of these three techniques the one I chose for normalizing the movie reviews was TF-IDF. This is because TF-IDF normalizes term frequencies based on the importance of not just current document but across the entire collection of documents in this case of movie reviews. TF-IDF gives higher weights to terms that are rare across the entire collection of documents, making it more effective in identifying important terms [29,30]. PPMI takes semantic relations between terms. It is used in distributional semantics to measure association between words based on their occurrence in the context of document. PPMI is used to capture semantic relationships between terms rather than sentiment [31]. So, the widely used approach for sentiment analysis is TF-IDF as it

[5] https://colab.research.google.com/drive/1IsqqtoEVG8n21tb0tUMCN0mLqiBbgnJ8 and from previous Coursework done in Machine Learning from Ekaterina.

[6] https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

[7] https://colab.research.google.com/drive/1IsqqtoEVG8n21tb0tUMCN0mLqiBbgnJ8 and from previous coursework done in Machine Learning from Ekaterina

takes importance of words within a text by considering the entire document and helps in identifying if the phrase is in positive or negative sentiment [32].

3.1.2 Feature Selection

Within the Feature Test.ipynb several features have been tested on Unigrams without threshold, Bigrams and Trigrams with Threshold and are applied to both Stemmed and Lemmatized words. Of the ones tested the three features below are the ones which gave a higher accuracy when tested using Multinomial NB. So, these were taken as three main features for testing other models.

Three sets of Features have been developed with a combination of Stopwords, Stemming, Lemmatization and N-grams.

Feature 1 – Stemming, Bigrams with threshold and Applied TF-IDF

```python
# Function to calculate TF-IDF with bi-grams and
# filter based on frequency threshold and after applying stemming
def calculate_tfidf_ngrams_with_threshold(documents, n, threshold):
    # Apply stemming
    st = LancasterStemmer()
    stemmed_documents_1 = [[' '.join([st.stem(word) for word in words])] for words in documents]

    # Generate n-grams and calculate frequency
    all_bigrams = []
    for doc in stemmed_documents_1:
        #Splitting terms in the document to words(bigrams)
        terms = list(ngrams(doc[0].split(), n))
        all_bigrams.extend(terms)

    # Calculate the frequency of all bigrams
    frequency_word_bigrams = nltk.FreqDist(all_bigrams)

    # Filter bigrams based on the threshold
    filtered_bigrams = {bigram: freq for bigram, freq in frequency_word_bigrams.items() if freq >= threshold}

    # Create a vocabulary
    all_terms = set(filtered_bigrams.keys())
    vocabulary_1 = sorted(list(all_terms))

    # Calculate TF - Asked ChatGPT on how to create a TF-matrix so it suggested using pd.Dataframe
    #https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html
    tf_matrix = pd.DataFrame(0, index=df.index, columns=vocabulary_1, dtype=float)
    #Iterating through index and documents
    for i, doc in enumerate(stemmed_documents_1): #https://realpython.com/python-enumerate/
        terms = list(ngrams(doc[0].split(), n))
        for term in terms:
            #counting the frequency of each term in each document.
            #Incrementing the tf matrix by 1 if the term is in the vocabulary
            if term in vocabulary_1:
                tf_matrix.at[i, term] += 1

    # Calculate IDF - Asked ChatGPT on how to create a IDF-vectors so it suggested using pd.Series
    #https://pandas.pydata.org/docs/reference/api/pandas.Series.html
    idf_vector = pd.Series(0, index=vocabulary_1, dtype=float)
    N = len(stemmed_documents_1)
    for term in vocabulary_1:
        #counting the number of documents in which the bigram appears
        df_term = sum([1 for doc in stemmed_documents_1 if term in list(ngrams(doc[0].split(), n))])
        #Calculating the IDF value of each bigram
        idf_vector.at[term] = math.log(N / (1 + df_term), 10)

    # Calculate TF-IDF
    tfidf_matrix_1 = tf_matrix * idf_vector

    return tfidf_matrix_1, vocabulary_1

# Calculate TF-IDF matrix with bigrams and filter based on frequency threshold
tfidf_matrix_1, vocabulary_1 = calculate_tfidf_ngrams_with_threshold(df['tokenized_review'], n=2, threshold=100)
```

Figure 7: Implementation of Feature 1

In this all the tokenized words of the documents are first stemmed. Bigrams are applied to the terms in those stemmed documents. After that filtering of the bigram is done so that bigrams which are less frequent are removed. A threshold of 100 is set, this means any bigram which has frequency less than 100 is not included in the vocabulary list. TF-IDF calculation is then done to this set of features. TF matrix is created to calculate the TF and IDF vector to calculate the IDF. Then both are multiplied to get the TF-IDF matrix which is used as transformed feature data for the Data split.

TF-Matrix is calculated by taking all the documents in the stemmed_document and again applying Bigrams to that and then checking to see if those terms are there in the vocabulary (filtered_bigrams) and incrementing the matrix by 1 if the term exists. That is counting the frequency of each term (bigram) in the document.
IDF-Vector is calculated by taking the term from the vocabulary and then counting the number of documents in which this bigram appears. Then it calculates the IDF value of that current bigram using the formula (log (N/1 + df_term),10) where N is total number of documents and df.term is document frequency of the bigram.
Then both the TF-Matrix and IDF-Vector is then multiplied to get the TF-IDF matrix

Code for calculation of TF-IDF is taken from the Practice Labs provided and changes have been made accordingly.

Feature 2 – Stemming, Unigrams without threshold, removing Stopwords and punctuations and Applied TF-IDF

```python
# Function to calculate TF-IDF after applying stemming
# and stopwords and punctuations removal on unigrams
def calculate_tfidf(documents):
    # Apply stemming
    st = LancasterStemmer()
    stemmed_documents_2 = [[' '.join([st.stem(word) for word in words if word not in
                                      stoplist and word not in string.punctuation])] for words in documents]


    # Create a vocabulary
    all_terms = set() #only unique terms are allowed
    for doc in stemmed_documents_2:
        all_terms.update(doc[0].split()) #.update() to add items to dictionary
    vocabulary_2 = sorted(list(all_terms))

   # Calculate TF - Asked ChatGPT on how to create a TF-matrix so it suggested using pd.Dataframe
    tf_matrix = pd.DataFrame(0, index=df.index, columns=vocabulary_2, dtype=float)
    #Iterating through index and document
    for i, doc in enumerate(stemmed_documents_2): #https://realpython.com/python-enumerate/
        #counting the frequency of each term in each document.
        #Incrementing the tf matrix by 1 if the term is in the document
        for term in doc[0].split():
            tf_matrix.at[i, term] += 1   #adding to the matrix

    # Calculate IDF - Asked ChatGPT on how to create a IDF-vectors so it suggested using pd.Series
    # https://pandas.pydata.org/docs/reference/api/pandas.Series.html
    idf_vector = pd.Series(0, index=vocabulary_2, dtype=float)
    N = len(stemmed_documents_2)
    for term in vocabulary_2:
        #counting the number of documents in which the term appears
        df_term = sum([1 for doc in stemmed_documents_2 if term in doc[0].split()])
        #Calculating the IDF value of each unigram
        idf_vector.at[term] = math.log(N / (1 + df_term), 10)


    # Calculate TF-IDF
    tfidf_matrix_2 = tf_matrix * idf_vector

    return tfidf_matrix_2, vocabulary_2

# Calculate TF-IDF matrix and get vocabulary
tfidf_matrix_2, vocabulary_2 = calculate_tfidf(df['tokenized_review'])
```

Figure 8: Implementation of Feature 2

A similar approach has been taken for generating this feature, the only differences are it is not considering the threshold, so this means that the vocabulary is created by splitting all the stemmed words of the document (unigram) and the Stopwords and Punctuations are removed. And then again TF-IDF matrix is created as explained above without any n-grams splitting but instead all the words in the document are split and compared against the terms in each document. And then IDF is calculated to count the number of documents in which that term appears. One of the drawbacks of this method is that since it's not considering any threshold it will be checking against all the terms within documents which is time consuming. So, as a future development I will incorporate threshold into this method.

Feature 3 – Lemmatization, Unigrams without threshold, removing Stopwords and punctuations and Applied TF-IDF

```python
# Function to calculate TF-IDF after applying Lemmatization
# and stopwords and punctuations removal on unigrams
def calculate_tfidf(documents):
    # Apply Lemmatization
    lemmatizer = WordNetLemmatizer()
    lemmatized_documents_1 = [[' '.join([lemmatizer.lemmatize(word) for word in words if word not in
                                         stoplist and word not in string.punctuation])] for words in documents]


    # Create a vocabulary
    all_terms = set()  #only unique terms are allowed
    for doc in lemmatized_documents_1:
        all_terms.update(doc[0].split()) #.update() to add items to dictionary
    vocabulary_3 = sorted(list(all_terms))

    # Calculate TF - Asked ChatGPT on how to create a TF-matrix so it suggested using pd.Dataframe
    tf_matrix = pd.DataFrame(0, index=df.index, columns=vocabulary_3, dtype=float)
    #Iterating through index and document
    for i, doc in enumerate(lemmatized_documents_1):#https://realpython.com/python-enumerate/
        #counting the frequency of each term in each document.
        #Incrementing the tf matrix by 1 if the term is in the document
        for term in doc[0].split():
            tf_matrix.at[i, term] += 1 #adding to matrix

    # Calculate IDF - Asked ChatGPT on how to create a IDF-vectors so it suggested using pd.Series
    # https://pandas.pydata.org/docs/reference/api/pandas.Series.html
    idf_vector = pd.Series(0, index=vocabulary_3, dtype=float)
    N = len(lemmatized_documents_1)
    for term in vocabulary_3:
        #counting the number of documents in which the term appears
        df_term = sum([1 for doc in lemmatized_documents_1 if term in doc[0].split()])
        #Calculating the IDF value of each unigram
        idf_vector.at[term] = math.log(N / (1 + df_term), 10)

    # Calculate TF-IDF
    tfidf_matrix_3 = tf_matrix * idf_vector

    return tfidf_matrix_3, vocabulary_3

# Calculate TF-IDF matrix and get vocabulary
tfidf_matrix_3, vocabulary_3 = calculate_tfidf(df['tokenized_review'])

```

Figure 9: Implementation of Feature 3

This approach is very similar to the Feature 2 development, the only difference is that it does Lemmatization instead of Stemming.

3.2 Data Splits[8]

Sklearn's train_test_split [28] function was used to split the features extracted into train, development (dev), and test sets.
First the vectorized feature is divided into train and test sets with train set having 80% and test set having 20% of the whole data.
Then, the train set is further divided into train and development set with train set having 80% and development set having 20% of the further divided train data.

Split data into train, developement and test sets with Feature 1 - Stemming, bigrams and Applied TF-IDF

```python
X_train1, X_test1, y_train1, y_test1 = train_test_split(tfidf_matrix_1, df.labels, test_size=0.2, random_state=42)
X_train1, X_dev1, y_train1, y_dev1 = train_test_split(X_train1, y_train1,test_size=0.2, train_size=0.8, random_state=42)
```

Split data into train, developement and test sets with Feature 2 - Stemming, removed stopwords and punctuations, unigrams and TF-IDF

```python
X_train2, X_test2, y_train2, y_test2 = train_test_split(tfidf_matrix_2, df.labels, test_size=0.2, random_state=42)
X_train2, X_dev2, y_train2, y_dev2 = train_test_split(X_train2, y_train2,test_size=0.2, train_size=0.8, random_state=42)
```

Split data into train, developement and test sets with Feature 3 - Lemmatization, removed stopwords and punctuations, unigrams and TF-IDF

```python
X_train3, X_test3, y_train3, y_test3 = train_test_split(tfidf_matrix_3, df.labels, test_size=0.2, random_state=42)
X_train3, X_dev3, y_train3, y_dev3 = train_test_split(X_train3, y_train3,test_size=0.2, train_size=0.8, random_state=42)
```

Figure 10: Splitting of data

The tool used for Data Splits is - from sklearn.model_selection import train_test_split

[8] https://realpython.com/train-test-split-python-data/

3.3 Evaluation[9]

Evaluation on the Test set for all the models was done by checking the Accuracy, Precision, Recall and F1 Score [40]:

1) Classification report – It shows the Precision, Recall, F1-Score, Support and Accuracy that is related to the model tested.
2) Precision – Calculates how exactly the classifier has made the decisions.
3) Recall – Measure the ability of the classifier to correctly identify all the positive values.
4) F1- Score – It calculates the harmonic mean of the Precision and Recall.
5) Support – Number of occurrences of that class in the dataset.
6) Accuracy – Shows the percentage of how well the model has performed.
7) Confusion Matrix – Measures the performance of the classification model. Consists of TP (True Positives), FP (False Positives), TN (True Negatives) and FN (False Negatives). TP – Predicts all the positive reviews correctly, TN – Predicts all negative reviews correctly, FP – Predicted some negative reviews as positive, FN – Predicted some positive reviews as negative [41].

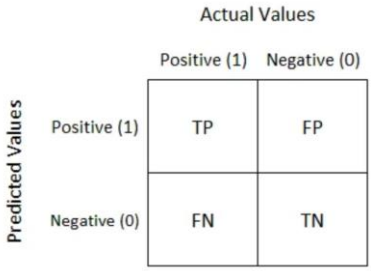


Figure 11: Confusion Matrix - (Image courtesy: My Photoshopped Collection) [41]

3.4 Testing on Machine Learning Algorithms

3.4.1 Naïve Bayes

Naïve Bayes is a classification algorithm which uses the probabilistic approach. The basis of Naïve Bayes comes from the Bayes Theorem [33].

According to Bayes Theorem [33]:

$$P(C|x) = \frac{P(x|C)\,P(C)}{P(x)}$$

That is:
P (C | x) - probability of finding a class given the observation (Posterior Probability)
P (x | C) - probability of finding the observation given a particular class (Likelihood)
P (C) – probability of class (Prior Probability)
P(x) – probability of observations

[9] https://github.com/ReiCHU31/Sentiment-analysis-of-IMDb-movie-reviews/blob/master/Sentiment%20analysis%20of%20IMDb%20movie%20reviews.ipynb

3.4.1.1 Naïve Bayes from Scratch[10]

Naïve Bayes algorithm calculates the posterior probability of each class given the observation and assigns the prediction to the class with the highest posterior probability [33].

The code for the implementation of the Naïve Bayes from scratch was **adapted from** [33] and with the help of ChatGPT.

```python
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Train the Naive Bayes model
def train_naive_bayes(X_train, y_train):
    # Calculate class priors
    prior = {}
    classes, counts = np.unique(y_train, return_counts=True)
    #calculate P(class) = count(class 0 or 1)/total number of values in the class
    #Prior probability
    for class_0_1, count in zip(classes, counts):
        prior[class_0_1] = count / len(y_train)

    # Calculate means and variances for each feature in each class
    means = {}
    var = {}

    #Taking feature of each of the class - (x)
    for cls in classes:
        class_indices = [i for i in range(len(y_train_encoded)) if y_train_encoded[i] == cls]
        #https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html
        class_data = X_train.iloc[class_indices] #Access the  feature of a particular class using iloc

        class_means = np.mean(class_data, axis=0)
        class_var = np.var(class_data, axis=0)

        means[cls] = class_means
        var[cls] = class_var

    return prior, means, var


#Gaussian Distribution
def Normal(n, mu, var):
    sd = np.sqrt(var)
     # Check if standard deviation is zero
    if sd == 0:
        return np.zeros_like(n)  # Return an array of zeros to avoid division by zero

    pdf = (np.e ** (-0.5 * ((n - mu)/sd) ** 2)) / (sd * np.sqrt(2 * np.pi))
    return pdf

# Prediction of Naive Bayes
def predict_naive_bayes(X, prior, means, var):
    classes = list(prior.keys())
    predictions = []

    # calculate P(reviews|class)
    for i in range(len(X)):
        class_likelihoods = []

        for cls in classes:
            feature_likelihoods = []
            # log-prior - P(class)
            feature_likelihoods.append(np.log(prior[cls]))

            for j in range(X.shape[1]):  # Use shape[1] to iterate over features
                data = X.iloc[i, j]  # Access the j-th feature of the i-th sample using iloc
                mean = means[cls][j]
                variance = var[cls][j]
                #likelihoods - p(reviews1|class) + p(reviews2|class) + ... + p(reviewsN|class)|P(class)
                likelihood = Normal(data, mean, variance)
                # log-Likelihood
                if likelihood != 0:
                    likelihood = np.log(likelihood)
                else:
                    likelihood = 1 / X.shape[1]  # Adjust the handling of zero likelihood

                feature_likelihoods.append(likelihood)

            total_likelihood = sum(feature_likelihoods)
            # Calculate posterior - P(class|reviews)
            class_likelihoods.append(total_likelihood)

        # Take class with max log-likelihood
        max_index = class_likelihoods.index(max(class_likelihoods))
        prediction = classes[max_index]
        predictions.append(prediction)

    return predictions
```

Figure 12: Implementation of Naïve Bayes from scratch

[10] https://blog.devgenius.io/implementing-na%C3%AFve-bayes-classification-from-scratch-with-python-badd5a9be9c3

There are three functions in the code (train_naive_bayes (), Normal () and predict_naive_bayes ()). The train_naive_bayes () function calculates the prior probability (P(class)) of the class and then calculates the mean and variance of the features of a particular class.

Predict_naive_bayes function is used to predict the posterior probability when the dev and test datasets are given.
As explained before to calculate the posterior probability (P (class | reviews)) we need calculate the likelihood (P (reviews | class)) and prior P(class) probabilities [33]. To calculate the likelihood of the distribution of reviews the method used here is Gaussian distribution. The function Normal () calculates the gaussian distribution given the mean and variance of the features of a particular class.

To calculate the log-likelihood, steps are taken which calculates the np.log(likelihood) if the likelihood calculated before using Gaussian distribution is not 0 and 1/X.Shape [1] (1/number of columns (features) in the array) if the likelihood is 0.

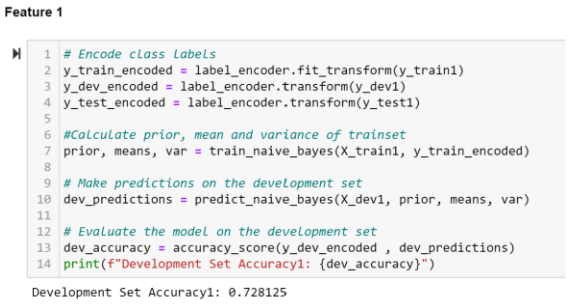
All the np.log (prior [class]) and np.log(likelihood) values are then added to the feature_likelihood list. And they are all summed together and added into the class_likelihood list. From that max_index takes the class with maximum_likelihood and prediction of the class is done in that way to decide if it a positive or negative review (feature).

A Label Encoder was used to convert all the Labels (classes) into numerical format so it can be used in the Naïve Bayes classifier above.

```
label_encoder = LabelEncoder()
# Encode class labels
y_train_encoded = label_encoder.fit_transform(y_train1)
y_dev_encoded = label_encoder.transform(y_dev1)
y_test_encoded = label_encoder.transform(y_test1)
```

Figure 13: Label Encoder on the Class Labels

Evaluation of this model was done on all the three features created above:



Feature 2

```
# Encode class labels
y_train_encoded = label_encoder.fit_transform(y_train2)
y_dev_encoded = label_encoder.transform(y_dev2)
y_test_encoded = label_encoder.transform(y_test2)

#Calculate prior, mean and variance of trainset
prior, means, var = train_naive_bayes(X_train2, y_train_encoded)

# Make predictions on the development set
dev_predictions = predict_naive_bayes(X_dev2, prior, means, var)

# Evaluate the model on the development set
dev_accuracy = accuracy_score(y_dev_encoded , dev_predictions)
print(f"Development Set Accuracy2: {dev_accuracy}")
```

```
Development Set Accuracy2: 0.55
```

Feature 3

```
# Encode class labels
y_train_encoded = label_encoder.fit_transform(y_train3)
y_dev_encoded = label_encoder.transform(y_dev3)
y_test_encoded = label_encoder.transform(y_test3)

#Calculate prior, mean and variance of trainset
prior, means, var = train_naive_bayes(X_train3, y_train_encoded)

# Make predictions on the development set
dev_predictions = predict_naive_bayes(X_dev3, prior, means, var)

# Evaluate the model on the development set
dev_accuracy = accuracy_score(y_dev_encoded , dev_predictions)
print(f"Development Set Accuracy3: {dev_accuracy}")
```

```
Development Set Accuracy3: 0.571875
```

Figure 14: Evaluating Naïve Bayes from scratch on three features.

From the figure above it is evident that the model with feature 1 gave a higher accuracy than other features when predicted on the dev set. So, since that feature 1 gave a higher accuracy, it was then used to predict the accuracy on the test set.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

y_test_encoded = label_encoder.transform(y_test1)
# Evaluating on test set
prior, means, var = train_naive_bayes(X_train1, y_train_encoded)
test_predictions = predict_naive_bayes(X_test1, prior, means, var)


print('Accuracy:',accuracy_score(y_test_encoded,test_predictions))
print('Confusion matrix:','\n',confusion_matrix(y_test_encoded,test_predictions))
print('Classification Report:','\n',classification_report(y_test_encoded,test_predictions))
```

```
Accuracy: 0.71375
Confusion matrix:
 [[254 124]
 [105 317]]
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.67      0.69       378
           1       0.72      0.75      0.73       422

    accuracy                           0.71       800
   macro avg       0.71      0.71      0.71       800
weighted avg       0.71      0.71      0.71       800
```

Figure 15: Evaluation of Naïve Bayes from scratch on test set

From the figure above it is evident that the accuracy the model gave for the test set was **0.71**

3.4.1.2 Multinomial Naïve Bayes[11]

The classifier is used for classifying data with discreate features. This is an inbuilt Naïve Bayes Classifier.

Feature 1

```
#Training a machine learning model (e.g., Naive Bayes) - for Feature 1
model_1 = MultinomialNB()
model_1.fit(X_train1, y_train1)
model_1.predict(X_dev1)
# Evaluate the model's performance on the dev data
accuracy1 = model_1.score(X_dev1, y_dev1)
print(accuracy1)
```

```
0.765625
```

Feature 2

```
# Training a machine learning model (e.g., Naive Bayes) - for Feature 2
model_2 = MultinomialNB()
model_2.fit(X_train2, y_train2)
model_2.predict(X_dev2)
# Evaluating the model's performance on the dev data
accuracy2 = model_2.score(X_dev2, y_dev2)
print(accuracy2)
```

```
0.784375
```

Feature 3

```
# Training a machine learning model (e.g., Naive Bayes) - for Feature 3
model_3 = MultinomialNB()
model_3.fit(X_train3, y_train3)

# Evaluating the model's performance on the dev data
model_3.predict(X_dev3)
accuracy3 = model_3.score(X_dev3, y_dev3)
print(accuracy3)
```

```
0.7859375
```

Figure 16: Evaluating Multinomial Naïve Bayes on three features.

Experimenting the three features on the MultinomialNB model gave an accuracy shown in the Figure above and from that it is evident that feature 3 gave a higher accuracy when predicted on the dev set. So, since that model_3 with feature 3 gave a higher accuracy, it was then used to predict the accuracy on the test set.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Evaluating on test set
predictions = model_3.predict(X_test3)

print('Accuracy:',accuracy_score(y_test3,predictions))
print('Confusion matrix:','\n',confusion_matrix(y_test3,predictions))
print('Classification Report:','\n',classification_report(y_test3,predictions))
```

```
Accuracy: 0.77875
Confusion matrix:
 [[308  70]
 [107 315]]
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.81      0.78       378
           1       0.82      0.75      0.78       422

    accuracy                           0.78       800
   macro avg       0.78      0.78      0.78       800
weighted avg       0.78      0.78      0.78       800
```

Figure 17: Evaluation of Multinomial NB on test set

From the figure above it is evident that the accuracy the model gave for the test set was **0.78**

[11] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

3.4.2 Logistic Regression and Support Vector Machines (SVMs)

3.4.2.1 Logistic Regression[12]

Logistic regression is a classification model which is used for linear or binary classification problems. It uses a Logistic function to get the binary outcomes. It does not require a linear relationship between the input and output variables. This is because it applies nonlinear log transformation to the odds ratio, allowing the complex relationship between the variables [34].

$$\text{Logistic Function} = \frac{1}{1 + e^{-x}}$$

These are the default hyperparameters which are set for Logistic Regression: (penalty="l2",*,dual=False, tol=1e-4, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,random_state=None,solver="lbfgs",max_iter=100, multi_class="auto", verbose=0, warm_start=False, n_jobs=None, l1_ratio=None) [35]

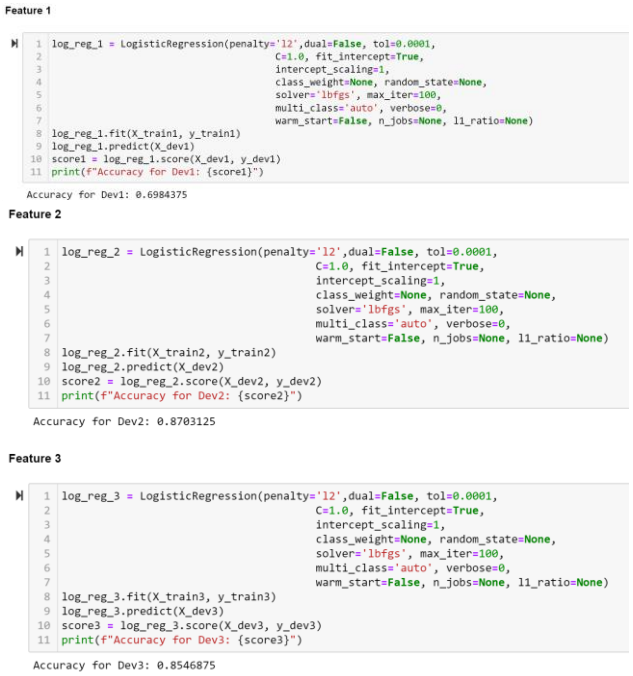


Figure 18: Evaluating Logistic Regression on three features.

Experimenting the three features on the Logistic Regression model gave an accuracy shown in the Figure above and from that it is evident that feature 2 gave a higher accuracy when predicted on the dev set. So, since the model (log_reg_2) with feature 2 gave a higher accuracy, it was then used to predict the accuracy on the test set.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Evaluating on test set
predictions = log_reg_2.predict(X_test2)

print('Accuracy:',accuracy_score(y_test2,predictions))
print('Confusion matrix:','\n',confusion_matrix(y_test2,predictions))
print('Classification Report:','\n',classification_report(y_test2,predictions))
```

```
Accuracy: 0.84125
Confusion matrix:
 [[314  64]
 [ 63 359]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83       378
           1       0.85      0.85      0.85       422

    accuracy                           0.84       800
   macro avg       0.84      0.84      0.84       800
weighted avg       0.84      0.84      0.84       800
```

Figure 19: Evaluation of Logistic Regression on test set

From the figure above it is evident that the accuracy the model gave for the test set was **0.84**

3.4.2.2 SVM (Support Vector Machines)[13]

A supervised learning method which is used for classification, regression, and outliers' detections tasks. Effective in high dimensional tasks where number of dimensions is greater than the number of samples. SVC (Support Vector Classification) is the one which is responsible for binary and multi-class classification tasks.

These are the default hyperparameters which are set for SVC:
(*, C=1.0, kernel="rbf", degree=3, gamma="scale", coef0=0.0, shrinking=True, probability=False, tol=1e-3, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape="ovr", break_ties=False, random_state=None.) [36,37].

Feature 1

```
svm_clf_1 = SVC(C=1.0, kernel='rbf',
                degree=3, gamma='scale',
                coef0=0.0, shrinking=True,
                probability=False, tol=0.001,
                cache_size=200, class_weight=None,
                verbose=False, max_iter=-1,
                decision_function_shape='ovr',
                break_ties=False, random_state=None)

svm_clf_1.fit(X_train1, y_train1)
svm_clf_1.predict(X_dev1)
score1 = svm_clf_1.score(X_dev1, y_dev1)
print(f"Accuracy for Dev1: {score1}")
```

```
Accuracy for Dev1: 0.7453125
```

Feature 2

```
svm_clf_2 = SVC(C=1.0, kernel='rbf',
                degree=3, gamma='scale',
                coef0=0.0, shrinking=True,
                probability=False, tol=0.001,
                cache_size=200, class_weight=None,
                verbose=False, max_iter=-1,
                decision_function_shape='ovr',
                break_ties=False, random_state=None)

svm_clf_2.fit(X_train2, y_train2)
svm_clf_2.predict(X_dev2)
score2 = svm_clf_2.score(X_dev2, y_dev2)
print(f"Accuracy for Dev2: {score2}")
```

```
Accuracy for Dev2: 0.8375
```

Feature 3

```
svm_clf_3 = SVC(C=1.0, kernel='rbf',
                degree=3, gamma='scale',
                coef0=0.0, shrinking=True,
                probability=False, tol=0.001,
                cache_size=200, class_weight=None,
                verbose=False, max_iter=-1,
                decision_function_shape='ovr',
                break_ties=False, random_state=None)

svm_clf_3.fit(X_train3, y_train3)
svm_clf_3.predict(X_dev3)
score3 = svm_clf_3.score(X_dev3, y_dev3)
print(f"Accuracy for Dev3: {score3}")
```

```
Accuracy for Dev3: 0.8375
```

Figure 20: Evaluating SVC on three features.

Experimenting the three features on the SVC model gave an accuracy shown in the Figure above and from that it is evident that feature 2 and feature 3 gave same accuracy when predicted on the dev set. So, since both models gave the same accuracy, model (svm_clf_3) with feature 3 was taken to check the accuracy on the test set.

[13] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Now apply those above metrics to evaluate your model
predictions = svm_clf_3.predict(X_test3)

print('Accuracy:',accuracy_score(y_test3,predictions))
print('Confusion matrix:','\n',confusion_matrix(y_test3,predictions))
print('Classification Report:','\n',classification_report(y_test3,predictions))
```

```
Accuracy: 0.805
Confusion matrix:
 [[265 113]
 [ 43 379]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.70      0.77       378
           1       0.77      0.90      0.83       422

    accuracy                           0.81       800
   macro avg       0.82      0.80      0.80       800
weighted avg       0.81      0.81      0.80       800
```

Figure 21: Evaluation of SVC on test set

From the figure above it is evident that the accuracy the model gave for the test set was **0.81**

3.4.2.2 Hyperparameter Optimisation[14]

Some of the Hyperparameters of both Logistic Regression and SVM was changed and tested to see the performance on the model which gave a higher accuracy when default parameters were used. From that the model with higher accuracy after the hyperparameters optimisation was tested on the test dataset to see the performance.

Table 1: Hyperparameter Optimisation of Logistic Regression Classifier

Parameter	log_reg_4	log_reg_5	log_reg_6	log_reg_7	log_reg_8
Penalty	L1	L2	L2	L2	L2
Dual	False	False	False	False	False
Tol	0.0001	0.0001	0.0001	0.0001	0.0001
C	1.0	1.0	1.0	1.0	1.0
Fit Intercept	True	True	True	True	True
Intercept Scaling	1	1	1	1	1
Class Weight	Balanced	Balanced	Dict	Dict	Balanced
Random State	None	None	None	None	42
Solver	liblinear	lbfgs	lbfgs	liblinear	sag
Max Iter	1000	1000	100	10000	100
Multi-Class	Auto	Multinomial	Auto	Auto	Auto
Verbose	5	0	0	10	10
Warm Start	False	False	False	False	False
N Jobs	None	None	None	None	None
L1 Ratio	None	None	None	None	None
Accuracy	**0.846875**	**0.8671875**	**0.8703125**	**0.875**	**0.88125**

[14] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```
#penalty=l2, class_weight='balanced',max_iter=100, solver='sag',random_state=42
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Now apply those above metrics to evaluate your model
predictions = log_reg_8.predict(X_test2)

print('Accuracy:',accuracy_score(y_test2,predictions))
print('Confusion matrix:','\n',confusion_matrix(y_test2,predictions))
print('Classification Report:','\n',classification_report(y_test2,predictions))
```

```
Accuracy: 0.83375
Confusion matrix:
 [[310  68]
 [ 65 357]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.82      0.82       378
           1       0.84      0.85      0.84       422

    accuracy                           0.83       800
   macro avg       0.83      0.83      0.83       800
weighted avg       0.83      0.83      0.83       800
```

Figure 22: Evaluation on test set of Logistic Regression after hyperparameter Optimisation

- The evaluation of the test set using the optimal hyperparameters for log_reg_8 resulted in an accuracy of **0.83**.
- While running Logistic Regression with default parameters on feature 1, a warning of 'Max Iterations reached' was encountered. To address this, the Max Iterations parameter was adjusted from 100 to 1000 and 10000.

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Figure 23: Maximum Iteration reached Warning.

- Table 1 highlights the performance of the data with Logistic Regression, particularly when various hyperparameters are modified. Testing using different solvers, 'liblinear,' 'lbfgs,' and 'sag'—revealed that the 'sag' solver with a random state of 42 and with other default parameters produced the highest accuracy.
- Additional parameters, such as Class Weight (dict or Balanced), Max Iterations (1000, 10000), and multi-class (Auto, Multinomial), were explored by having different solvers to check which gave a better accuracy.
- These are the few hyperparameters which have been randomly changed and tested, a lot of combinations can be further tested.

Table 2: Hyperparameter Optimisation of SVM (SVC) Classifier

Parameter	svm_clf_4	svm_clf_5	svm_clf_6	svm_clf_7	svm_clf_8
Kernel	Poly	Sigmoid	RBF	Sigmoid	Poly
Degree	3	3	3	3	3
Gamma	Auto	Scale	Auto	Scale	Scale
Coef0	0.1	0.1	0.1	0.1	0.1
Shrinking	True	True	True	True	True
Probability	False	False	False	False	False
Tol	0.001	0.001	0.001	0.001	0.001
Cache Size	200	200	200	200	200

Class Weight	Balanced	None	Balanced	Balanced	Balanced
Verbose	False	False	False	False	False
Max Iter	-1	-1	-1	-1	-1
Decision Function Shape	OVO	OVO	OVO	OVR	OVR
Break Ties	False	False	False	False	False
Random State	42	42	42	42	None
Accuracy	**0.4859375**	**0.85625**	**0.515625**	**0.853125**	**0.540625**

```
#kernel='sigmoid', gamma='scale', coef0 = 0.1,
#class_weight= None, decision_function_shape='ovo',
#random_state=42
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Now apply those above metrics to evaluate your model
predictions = svm_clf_5.predict(X_test3)

print('Accuracy:',accuracy_score(y_test3,predictions))
print('Confusion matrix:','\n',confusion_matrix(y_test3,predictions))
print('Classification Report:','\n',classification_report(y_test3,predictions))
```

```
Accuracy: 0.83625
Confusion matrix:
 [[305  73]
 [ 58 364]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.81      0.82       378
           1       0.83      0.86      0.85       422

    accuracy                           0.84       800
   macro avg       0.84      0.83      0.84       800
weighted avg       0.84      0.84      0.84       800
```

Figure 24: Evaluation on test set of SVC after hyperparameter Optimisation

- The test set evaluation using the best hyperparameters for svm_clf_5 gave an accuracy of **0.84**.
- Analysis from Table 2 indicates that the SVM model performs well on this data when the kernel is set to 'Sigmoid' with a random state of 42. Conversely, it exhibits suboptimal performance with the 'Poly' kernel.
- The default kernel for SVC is 'RBF,' exploration of other hyperparameters such as Gamma (from 'Scale' to 'Auto'), Class Weight (from 'None' to 'Balanced'), and Decision Function shape (from 'OVR' to 'OVO') also gave suboptimal performance.
- The main hyperparameters that were tested are different kernels (Poly, Sigmoid, and RBF). Depending on the chosen kernel, additional hyperparameters were adjusted. Notably, Gamma was assessed using both 'Auto' and 'Scale.'
- These are the few hyperparameters which have been randomly changed and tested, a lot of combinations can be further tested.

3.5 Testing on Deep Learning Algorithm

3.5.1 BERT

BERT[15] **(Bidirectional Encoder Representation for Transformers**) is a Machine Learning transformer model which is used for many different applications and one which is sentiment

[15] https://huggingface.co/transformers/v3.2.0/custom_datasets.html
https://huggingface.co/docs/transformers/tasks/sequence_classification
https://github.com/dipanjanS/deep_transfer_learning_nlp_dhs2019/tree/master/notebooks

analysis. This method is useful when there are large amounts of data, and this transform model brings parallelization which makes it easier to train large amounts of data [38].
List of Libraries imported to run the BERT model:

```
#Libraries

import numpy as np
import pandas as pd
import nltk
import string
import os
import evaluate
import torch


from sklearn.model_selection import train_test_split
from transformers import DistilBertTokenizerFast
```

Figure 25: List of Libraries

Data Split[16] : Splitting dataset into Train, Development, and Test Set and Formatting them into CSV files. The dataset for BERT was divided into train, development, and test splits and all of them having same number of splits as tested with other models. That is the whole data is first divided into train and test set with train set having 80% of data and test set having 20%. Then the train set is then further divided into train and development set in which train set having 80% of data and development set having 20% of the data.

The data split into train, test and dev sets are then formatted into .csv files for further processing.

```
# Taking 'reviews' and 'labels' columns from the whole dataset
data = {'Reviews': df['reviews'],'Labels': df['labels']}
df_bert = pd.DataFrame(data)
```

```
# Split data into train, dev, and test sets
X_train, X_test, y_train, y_test = train_test_split(df_bert['Reviews'], df_bert['Labels'],
                                                    test_size=0.2, random_state=42)
X_train, X_dev, y_train, y_dev = train_test_split(X_train, y_train, test_size=0.2,
                                                  train_size=0.8, random_state=42)

train_df = pd.DataFrame({'Reviews': X_train, 'Labels': y_train})
dev_df = pd.DataFrame({'Reviews': X_dev, 'Labels': y_dev})
test_df = pd.DataFrame({'Reviews': X_test, 'Labels': y_test})

# Save to CSV files
train_df.to_csv('train.csv', index=False)
dev_df.to_csv('dev.csv', index=False)
test_df.to_csv('test.csv', index=False)
```

Figure 26: Splitting data into train, test and dev sets and converting them into csv files.

Two pre-trained versions of BERT were used for this sentiment analysis. BERT- base cased and BERT- base uncased. In uncased version all the words are converted to lowercase and in cased version all the uppercase and lowercase words are kept as it is [39].

Tokenization was done using DistilBertTokenizerFast for cased and uncased.

[16] https://stackoverflow.com/questions/38250710/how-to-split-data-into-3-sets-train-validation-and-test

```
from transformers import DistilBertTokenizerFast
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-cased')
```

```
train_encodings = tokenizer(train_df['Reviews'].tolist(), truncation=True, padding=True)
dev_encodings = tokenizer(dev_df['Reviews'].tolist(), truncation=True, padding=True)
test_encodings = tokenizer(test_df['Reviews'].tolist(), truncation=True, padding=True)
```

Figure 27: Tokenization of based cased.

```
#Tokenizer for Uncased version of the Base model
from transformers import DistilBertTokenizerFast
tokenizer1 = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
```

```
train_encodings1 = tokenizer1(train_df['Reviews'].tolist(), truncation=True, padding=True)
dev_encodings1 = tokenizer1(dev_df['Reviews'].tolist(), truncation=True, padding=True)
test_encodings1 = tokenizer1(test_df['Reviews'].tolist(), truncation=True, padding=True)
```

Figure 28: Tokenization of based uncased.

A class called IMDbDataset was created which takes all the labels and the encoded reviews and processed so that PyTorch datasets can be easily used with a PyTorch DataLoader during the training process [46].

```
import torch

class IMDbDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx): #Asked chatGPT
        if idx < len(self.labels):
            item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
            item['labels'] = torch.tensor(self.labels.iloc[idx])  # Use iloc to access the label by index
            return item
        else:
            # Handle the case where the index is out of range
            return None  # or raise an exception, depending on your needs

    def __len__(self):
        return len(self.labels)



train_dataset = IMDbDataset(train_encodings, train_df['Labels'])
dev_dataset = IMDbDataset(dev_encodings, dev_df['Labels'])
test_dataset = IMDbDataset(test_encodings, test_df['Labels'])

```

Figure 29: Labels and reviews proceed so it can be used with PyTorch.

Then the transformed dataset is then given into the transform model for Sequence Classification (DistilBertSequenceClassification).

```
from transformers import DistilBertForSequenceClassification, Trainer, TrainingArguments
```

Figure 30: Loading the transformer model for training.

These are the arguments(hyperparameters) which are used to train the model. For the model to run faster, the number of epochs was reduced to 1 and learning rate was also reduced to a smaller value 1e-5. Then the model is trained on the trainset and then evaluated on the development set. That is Fine-Tuning the BERT model on the dataset provided.

```
from transformers import DistilBertForSequenceClassification, Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=1,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    learning_rate=1e-5,
    warmup_steps=10,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)

model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-cased")

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=dev_dataset,
    compute_metrics=compute_metrics
)

trainer.train()
```

Figure 31: Setting parameters and fine-tuning the model.

This model is used for cased and uncased versions.

```
model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")
```

```
model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-cased")
```

Figure 32: Model for cased and uncased

```
#To evaluate the performace of the transformer
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return accuracy.compute(predictions=predictions, references=labels)
```

```
accuracy = evaluate.load("accuracy")
```

Figure 33: Evaluating[17] the performance of the model.

The Accuracy of the model was evaluated first on the Dev dataset and from the Figures below BERT-base cased version gave an accuracy of 0.865 and BERT-base uncased version gave an accuracy of 0.875.

```
trainer.evaluate()
```

```
{'eval_loss': 0.36846449971199036,
 'eval_accuracy': 0.865625,
 'eval_runtime': 482.9082,
 'eval_samples_per_second': 1.325,
 'eval_steps_per_second': 0.021,
 'epoch': 1.0}
```

Figure 34: Accuracy on the dev set of BERT cased version.

[17] https://www.kaggle.com/code/pritishmishra/text-classification-with-distilbert-92-accuracy
https://huggingface.co/docs/transformers/tasks/sequence_classification

```
1 trainer.evaluate()
```

```
{'eval_loss': 0.3671916127204895,
 'eval_accuracy': 0.875,
 'eval_runtime': 480.8928,
 'eval_samples_per_second': 1.331,
 'eval_steps_per_second': 0.021,
 'epoch': 1.0}
```

Figure 35: Accuracy on the dev set of BERT uncased version.

Then Accuracy on the test dataset was checked to see how the model was performing. The cased version gave an accuracy of 0.845 for the test set and uncased version gave an accuracy of 0.878.

```
1 pred_test = trainer.predict(test_dataset)
```

```
1 pred_test.metrics
```

```
{'test_loss': 0.38908982276916504,
 'test_accuracy': 0.845,
 'test_runtime': 621.418,
 'test_samples_per_second': 1.287,
 'test_steps_per_second': 0.021}
```

Figure 36: Accuracy on the test set of BERT cased version

```
1 pred_test1 = trainer.predict(test_dataset1)
```

```
1 pred_test1.metrics
```

```
{'test_loss': 0.367340087890625,
 'test_accuracy': 0.87875,
 'test_runtime': 608.4871,
 'test_samples_per_second': 1.315,
 'test_steps_per_second': 0.021}
```

Figure 37: Accuracy on the test set of BERT uncased version

4. Discussion

Table 3: Results of each model on the Test set

Method0.86 f	Accuracy
Naïve Bayes (from scratch)	0.71
Naïve Bayes (scikit-learn)	0.78
Logistic Regression	0.84
SVM	0.81
Logistic Regression (with Hyperparameter Optimisation)	0.83
SVM (with Hyperparameter Optimisation)	0.84
BERT (Cased)	0.845
BERT (Uncased)	0.87875

Table 3 shows the overall result of the test data on each of the machine learning models. BERT-base uncased version gave the highest accuracy compared to the other models with accuracy **~0.88**.

Naïve Bayes from scratch gave a lower accuracy of 0.71 this maybe because of the way the implementation was done and by changing the way the distribution was calculated, that is instead of using Gaussian Distribution maybe calculating the distribution using Multinomial or Bernoulli [42] would have given a higher accuracy. Multinomial Naïve bayes gave an accuracy of 0.78 for test set. Which again is not that high compared to accuracies given by other models. So overall I feel Naïve Bayes is not a good model for classifying this dataset.

Next algorithm the data was tested on was Logistic Regression and it gave an accuracy of 0.84 with default hyperparameters and when the hyperparameters of the model was changed to calculate the performance of the model it gave an accuracy of 0.83. From testing it was clear that this data works well with Logistic Regression model. Even after hyperparameter Optimisation the model was giving a higher accuracy but the accuracy on the test was slightly lower.

Then it was tested on SVM (SVC) where again it performed almost like Logistic Regression and gave an accuracy of 0.81 with default parameter and 0.84 after Hyperparameter Optimisation. So, again this model is also a good choice for this dataset.

Some reasons why Logistic Regression and SVC outperformed Naïve Bayes is because these algorithms try to capture more complex information from the data compared to Naïve Bayes where it checks for the conditional probability of the feature of the give class to make assumptions and then classify the model. Both Logistic Regression and SVC can handle non-linear relationship between the transformed data and Naive Bayes being linear may not be able to handle it. For Naïve Bayes there is not Hyperparameter Optimisation so no changes can be made to the model to improve the performance [43].

The IMDB reviews dataset was then applied onto the BERT Transformer model. This model outperformed all the other classification models. BERT-base uncased version gave the highest accuracy when compared to all other models. This is because BERT having this transformer architecture allows for parallelization during training so as result it will be able to capture more long-range dependencies of the texts [45]. Since the data is being fine-

tuned on the pre-trained model (a model which has been trained on a large corpus of data), the model will be able to capture any intricate information and patterns from the data and can more accurately make the classification [47].

5. Conclusion and Future Work

In this project the IMDB Reviews dataset was tested on a few Machine Learning algorithms and with a Deep Learning algorithm (BERT).

Overall, the best performing model for this IMDB Reviews dataset is BERT-base uncased version. This is because BERT, being a pretrained model that is it has been trained on a large corps of data the model is able to capture any form of patterns from the data after fine-tuning. Fine-tuning on the pre-trained model, it can adapt to the data provided and make changes accordingly for sentiment analysis and BERT model is able to handle any complex tasks. The accuracy of the BERT model can further be improved by proper Hyperparameter tuning and using different Tokenization algorithms [44].

To improve the results in the future I would consider using built-in functions for TF-IDF which will give better accuracy for the features generated. Trying out many more feature generation and feature selection techniques. Testing with more hyperparameters for Logistic Regression and SVM. For the BERT model increasing the learning rate and epochs. For Naïve Bayes from scratch try to implement most of the code by myself in the future to see the changes in the accuracy. Due to time constraints these could not be done and tested.

References

- I acknowledge that this work is my own, and I have used ChatGPT 3.5 (OpenAI, https://chat.openai.com/) in few places for code development and to proofread the final draft of my report.

1. Learning Word Vectors for Sentiment Analysis, Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher, June 2011, online - http://www.aclweb.org/anthology/P11-1015 , https://ai.stanford.edu/~amaas/data/sentiment/
2. MonkeyLearn, Sentiment Analysis: A Definitive Guide, online: https://monkeylearn.com/sentiment-analysis/
3. THE APP SOLUTIONS, Why Business Applies Sentiment Analysis? 5 Successful Examples, 2023, online: https://theappsolutions.com/blog/development/sentiment-analysis-for-business/
4. AIMultiple, Top 5 Sentiment Analysis Challenges and Solutions in 2023, online: https://research.aimultiple.com/sentiment-analysis-challenges/
5. Dovetail Editorial Team, 2023, Understanding sentiment analysis using machine learning online: https://dovetail.com/customer-research/sentiment-analysis-using-machine-learning/#:~:text=Four%20machine%20learning%20techniques%20for,reinforcement%20and%20semi%2Dsupervised%20learning.

6. DataRobot, 2019, Using Machine Learning for Sentiment Analysis: a Deep Dive, online: https://www.datarobot.com/blog/using-machine-learning-for-sentiment-analysis-a-deep-dive/#:~:text=RNNs%20are%20probably%20the%20most,piece%20of%20text%20is%20ingested.
7. Yulia Gavrilova, 2023, Transformers in ML: What They Are and How They Work, online: https://serokell.io/blog/transformers-in-ml
8. C.J. Hutto, Eric Gilbert, January 2015, VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Test. Online:https://www.researchgate.net/publication/275828927_VADER_A_Parsimonious_Rule-based_Model_for_Sentiment_Analysis_of_Social_Media_Text
9. Kavya Suppala, Narasinga Rao, June 2019, Sentiment Analysis using Naïve Bayes Classifier, online: https://www.ijitee.org/wp-content/uploads/papers/v8i8/H6330068819.pdf
10. Christopher D. Manning Prabhakar Raghavan Hinrich Schütze, 2009, An Introduction to Information Retrieval, online: https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf
11. Tony Mullen and Nigel Collier, Sentiment analysis using support vector machines with diverse information sources, online: https://aclanthology.org/W04-3253.pdf
12. Corinna Cortes, Vladimir Vapnik, 1995, Support- Vector Networks, online: https://link.springer.com/article/10.1023/A:1022627411411
13. Leo Breiman, 2001, Random Forests, online: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf
14. Shoffan Saifullah1,a, Yuli Fauziah1,b , Agus Sasmito Aribowo1,2,c, Comparison of Machine Learning for Sentiment Analysis in Detecting Anxiety Based on Social Media Data, online: https://arxiv.org/ftp/arxiv/papers/2101/2101.06353.pdf#:~:text=Random%20Forest%20(RF)%20is%20a,the%20sample%20data%20they%20have.
15. Sepp Hochreiter, Jürgen Schmidhuber, 1997, Long Short-Term Memory, online: https://www.bioinf.jku.at/publications/older/2604.pdf
16. Robin M. Schmidt, November 2019, Recurrent Neural Networks (RNNs): A gentle Introduction and Overview, online: https://arxiv.org/pdf/1912.05911.pdf
17. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, January 2019, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, online: https://arxiv.org/pdf/1810.04805v2.pdf
18. Bruno Stecanella, June 2017, MonkeyLearn, Support Vector Machines (SVM) Algorithm Explained, online: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/
19. Intellipaat Software Solutions, What is LSTM? Introduction to Long Short-Term Memory, online: https://intellipaat.com/blog/what-is-lstm/
20. IBM, What are Recurrent Neural Networks?, online: https://www.ibm.com/topics/recurrent-neural-networks#:~:text=A%20recurrent%20neural%20network%20(RNN,data%20or%20time%20series%20data.
21. Ben Lutkevich, BERT Language mode, online: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-

model#:~:text=BERT%2C%20which%20stands%20for%20Bidirectional,calculated%20based%20upon%20their%20connection.
22. Amal Menzli, August 2023, Tokenization in NLP: Types, Challenges, Examples, and Tools, online: https://neptune.ai/blog/tokenization-in-nlp
23. Neeraj Agarwal, July 2022, N-gram Language Modeling in Natural Language Processing, online: https://www.kdnuggets.com/2022/06/ngram-language-modeling-natural-language-processing.html
24. Kavita Ganesan, What are Stopwords? Online: https://www.opinosis-analytics.com/knowledge-base/stop-words-explained/#:~:text=Stop%20words%20are%20a%20set,carry%20very%20little%20useful%20information.
25. Aditya Beri, May 2020, Stemming vs Lemmatization, online: https://towardsdatascience.com/stemming-vs-lemmatization-2daddabcb221
26. Saumyab271, July 2023, Stemming vs Lemmatization in NLP: Must-Know Differences, online: https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/#:~:text=Stemming%20is%20a%20process%20that,'%20would%20return%20'Car'.
27. MonkeyLearn, Understanding TF-IDF: A Simple Introduction, online: https://monkeylearn.com/blog/what-is-tf-idf/
28. Sklear.model_selection.train_test_split, online: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
29. Tf-idf, Online: https://en.wikipedia.org/wiki/Tf%E2%80%93idf#:~:text=A%20high%20weight%20in%20tf,to%20filter%20out%20common%20terms.
30. Fatih Karabiber, TF-IDF – Term Frequency- Inverse Document Frequency , online: https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/
31. Lena Voita, Word Embeddings, online: https://lena-voita.github.io/nlp_course/word_embeddings.html
32. Jay Singh, July 2023, TF-IDF in Sentiment Analysis, online: https://www.tutorialspoint.com/tf-idf-in-sentiment-analysis#:~:text=A%20popular%20technique%20in%20sentiment,express%20positive%20or%20negative%20moods.
33. Gerry Christian Ongko, Implementing Naïve Bayes Classification from Scratch with Python, Jan 2023, online: https://blog.devgenius.io/implementing-na%C3%AFve-bayes-classification-from-scratch-with-python-badd5a9be9c3
34. Igor Kononenko, Matjaž Kukar, Chapter 10 - Statistical Learning, Machine Learning and Data Mining, Woodhead Publishing, 2007, Pages 259-274, ISBN 9781904275213, online: https://www.sciencedirect.com/topics/computer-science/logistic-regression#:~:text=Logistic%20regression%20is%20a%20process,%2Fno%2C%20and%20so%20on.
35. Sklearn.linear_model.LogisticRegression, online: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
36. Support Vector Machines, online: https://scikit-learn.org/stable/modules/svm.html

37. Sklearn.svm.SVC, online: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
38. Britney Muller, BERT 101 State of the Art NLP Model Explained, March 2022, online: https://huggingface.co/blog/bert-101
39. K. Kyritsis, N. Spatiotis, I. Perokos, M. Paraskevas, A Comparative Performance Evaluation of Algorithms for the Analysis and Recognition of Emotional Content, DOI: 10.5772/intechopen.112627, July 2023, online: https://www.intechopen.com/online-first/87923
40. Yellowbrick, Classification Report, online: https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html#:~:text=The%20classification%20report%20visualizer%20displays,with%20a%20color%2Dcoded%20heatmap.
41. Sarang Narkhede, Understanding Confusion Matrix, 2018, online: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
42. Nagesh Singh Chauhan, Naïve Bayes Algorithm: Everything You Need To Know, April 2022, Online: https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html
43. P. Golpour , M. Ghayour-Mobarhan, A. Sak., H. Esmaily, A. Taghipour, M. Tajfard, H. Ghazizadeh, M. Moohebati, G.A. Ferns, 2020, Comparison of Support Vector Machine, Naïve Bayes and Logistic Regression for Assessing the Necessity for Coronary Angiography. International journal of environmental research and public health, 17(18), 6449. Online: https://doi.org/10.3390/ijerph17186449
44. S. Alaparthi, M. Mishra, BERT: a sentiment analysis odyssey, J Market Anal 9, 118–126, 2021, online: https://doi.org/10.1057/s41270-021-00109-8
45. Edoardo Guerriero, Why does the transformer do better than RNN and LSTM in long-range context dependencies?, Stack Exchange, online: https://ai.stackexchange.com/questions/20075/why-does-the-transformer-do-better-than-rnn-and-lstm-in-long-range-context-depen
46. Hugging Face, Fine-Tuning with custom Datasets, online: https://huggingface.co/transformers/v3.2.0/custom_datasets.html
47. Nikolaj Buhl, Training vs. Fine-tuning: What is the Difference, Nov 2023, online: https://encord.com/blog/training-vs-fine-tuning/