**INTELLIGENT DATA AND TEXT ANALYTICS (IDTA)**

| Student ID number | Title of degree studying | Year |
|---|---|---|
| **2301022** | MSC ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING | 2024 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Short unit name: | M33880 | | | Due date: 18 Nov 2024 | | |
| Full unit name: | INTELLIGENT DATA AND TEXT ANALYTICS (IDTA) | | | | | |
| **Unit lecturer name:** | Ms. Aarushi Vadhavkar | | | | aarushi.vadhavkar@kaplan.edu.sg | |
| Additional items e.g., CD/disk/USB: | Yes | | No | ✓ | **Details:** | |

All additional items should be clearly labelled with ID number and unit name and securely attached to your work.

Candidates are reminded that the following are defined as Assessment Offences and will be dealt with in accordance with theUniversity's Code of Student Discipline:

a) Any attempt to complete an assessment by means considered to be unfair;
b) Plagiarism, which the University defines as the incorporation by a student in work for assessment of material which is not their own, in the sense that all or a substantial part of the work has been copied without any adequate attempt at attribution or has been incorporated as if it were the student's own work when in fact it is wholly or substantially the work of another person or persons.

Please note: Group **coursework** will be filed under the first Student ID number at the top of the list. Ensure you know all **group member's ID numbers.**

NB: **Coursework not collected** will be disposed of six months **after** the hand-in date.

---

**FOR OFFICIAL USE ONLY**

| Date received/Office stamp | Provisional mark % / Comments |
|---|---|
| | |

Administration Office

Academic Staff Member

# From Text to Insight: NLP-Powered Sentiment Analysis of Yelp Restaurant Reviews

An In-Depth Analysis of the Yelp's Restaurant Reviews Based on Customer Sentiments using Natural Language Processing (NLP)-Driven Insights

## Table of Contents


I. Abstract
II. Keywords
III. Introduction
IV. Data Loading
V. Data Viewing & Preprocessing
- 5.1 Viewing the portion of yelp restaurant customer review sentiment
  1. Figure. *Loading and showing first 10 rows of data*
- 5.2 Naming columns
  1. Figure. *Naming the respective columns*
- 5.3 Dataset Imbalance and Null Check
  1. Figure. *Label Data are balanced and having no null values*
- 5.4 Expand Contraction
- 5.5 Add Space Around Punctuation
- 5.6 Remove Punctuation
- 5.7 Convert to Lowercase
- 5.8 Removing numerical data
- 5.9 Lemmatization
- 5.10 Handling Stop Words using NLTK
- 5.11 Reshuffling the processed Dataframe
  1. Figure. *Reshuffled dataframe after all pre-processing*
- 5.12 Distribution of Processed Sentence by reviews text by Sentiments
  1. Figure. *Distribution of Reviews by Sentiment – Positive (1) or Negative (0) reviews*
- 5.13 Box Plot Visualization
  1. Figure. *Showing the distribution of review lengths in training data*
- 5.14 Visualization on how sentences are in the training data
  1. Figure. *Histogram showing how long our sentences are in the training data*
- 5.15 WordCount for Review Text
  1. Figure. *Word Count for Negative Reviews*
  2. Figure. *Word Count for Positive Reviews*

VI. Performing Classification & Explanation
- 6.1 Logistic Regression
  1. Figure. *Confusion Matrix for Logistic Regression after applying TfidfVectorizer*
  2. Figure. *Confusion Matrix for Logistic Regression after applying CountVectorizer*
- 6.2 Naïve Bayes Classification
  1. Figure. *Confusion Matrix for Naive Bayes after applying TfidfVectorizer*
- 6.3 Support Vector Classifier
  1. Figure. *Confusion Matrix for Support Vector Classifier after applying TfidfVectorizer and CountVectorizer*
- 6.4 Random Forest Classifier
  1. Figure. *Confusion Matrix for Random Forest Classifier after applying TfidfVectorizer and CountVectorizer*
- 6.5 Gradient Boosting Classifier
  1. Figure. *Confusion Matrix for* Gradient Boosting Classifier *after applying TfidfVectorizer*
- 6.7 Model Performance Comparison

VII. BERT-Based Model for Classification
- 7.1 Modelling BERT
  1. Figure: Showing steps in Modelling BERT
  2. Figure: Showing BERT model Summary

3. Figure: Plotting steps in Modelling BERT
4. Figure: Graph showing Training vs Loss and Accuracy
5. Figure: Showing Confusion Matrix and Classification Report

- 7.2 Evaluate the model
- 7.3 Comparative Analysis of BERT Model vs Other Classification models

VIII. Performing Topic Detection

- 8.1 LDA Model Setup
  1. Figure. *Typic flow in Topic detection*
- 8.2 Topic Interpretation
- 8.3 Topic model Visualization using pyLDAvis
  1. Figure. *The pyLDAvis visualization provides an interactive way to understand the topics discovered by your LDA model*

IX. Conclusion

X. References

XI. Key Terms and Definitions

# I. ABSTRACT

This project explores sentiment analysis of Yelp restaurant reviews using Natural Language Processing (NLP) techniques. With the growing importance of online reviews in consumer decision-making, businesses are increasingly relying on feedback platforms like Yelp to gauge customer satisfaction. This sentiment analysis aims to classify restaurant reviews as positive, neutral, or negative based on textual data.

We employ a range of NLP methods, including data pre-processing (tokenization, stopword removal, and lemmatization) and feature extraction techniques such as TF-IDF and word embeddings. A variety of machine learning models, including Logistic Regression, Support Vector Machines, were trained on the processed review data. Later BERT classification was performed and compared the results. Topic detection has been employed also.

The performance of these models was evaluated using metrics such as accuracy, precision, recall, and F1-score. Our findings indicate that machine learning algorithms, particularly deep learning models, exhibit strong performance in accurately predicting the sentiment of Yelp reviews. The results highlight the effectiveness of NLP techniques in sentiment analysis and offer insights into consumer preferences, which can help restaurant businesses improve their services based on customer feedback.

# II. KEYWORDS

Sentiment, Yelp, Linear model, Regression Models, NLTK, Machine Learning, Correlation, Mean Squared Error, Classification Report, Heatmap, Silhouette, RoC Curve, Confusion Matrix, Jaccard, GridSearchCV, Naïve Bayes, skLearn Pipeline, WordNetLemmatizer, tensorflow_text, AdamW optimizer

# III. INTRODUCTION

In today's digital age, online reviews have become a critical factor in shaping consumer behaviour. Platforms like Yelp serve as a significant source of information, where millions of users share their experiences with restaurants, influencing the choices of potential customers. Given the volume and variety of reviews, it is challenging for businesses to manually assess and respond to customer feedback in a timely manner. This has led to the growing importance of sentiment analysis, an area of Natural Language Processing (NLP) that aims to automatically determine the emotional tone behind textual data.

Sentiment analysis of Yelp restaurant reviews offers valuable insights into customer satisfaction, service quality, food preferences, and overall dining experiences. By classifying reviews as positive, negative, or neutral, restaurant owners can identify areas of improvement and capitalize on strengths, while consumers can quickly assess the reputation of a business. This automation enables a more efficient analysis of large datasets, allowing businesses to stay competitive in an industry where customer perception can make or break success.

This report investigates sentiment analysis using NLP techniques to classify Yelp restaurant reviews. By leveraging various machine learning algorithms and text processing methods, the project aims to develop a robust model capable of accurately predicting the sentiment of a given review. Additionally, the report explores the impact of different feature extraction techniques, such as Term Frequency-Inverse Document Frequency (TF-IDF) and word embeddings, on model performance. The results will contribute to understanding how automated sentiment analysis can be used to enhance the dining industry’s customer service strategies.

# IV. DATA LOADING

All required libraries have been imported and the provided yelp_labelled.txtdata is loaded into Dataframe object using panda's library and basic data review performed as first step.

# V. DATA VIEWING & PREPROCESSING

1. **Viewing the portion of yelp restaurant customer review sentiment**

```
pd.set_option('display.max_colwidth', None) # full text in text column

df = pd.read_csv('yelp_labelled.txt', sep='\t', header=None)
df.head(10)
```

| | 0 | 1 |
|---|---|---|
| 0 | Wow... Loved this place. | 1 |
| 1 | Crust is not good. | 0 |
| 2 | Not tasty and the texture was just nasty. | 0 |
| 3 | Stopped by during the late May bank holiday off Rick Steve recommendation and loved it. | 1 |
| 4 | The selection on the menu was great and so were the prices. | 1 |
| 5 | Now I am getting angry and I want my damn pho. | 0 |
| 6 | Honeslty it didn't taste THAT fresh.) | 0 |
| 7 | The potatoes were like rubber and you could tell they had been made up ahead of time being kept under a warmer. | 0 |
| 8 | The fries were great too. | 1 |
| 9 | A great touch. | 1 |

```
[ ] df.shape
```

```
(1000, 2)
```

Figure. *Loading and showing first 10 rows of data*

2. **Naming columns**

```
[ ] col_names=['ReviewText','Sentiment']
    df.columns=col_names

    df.head()
```

| | ReviewText | Sentiment |
|---|---|---|
| 0 | Wow... Loved this place. | 1 |
| 1 | Crust is not good. | 0 |
| 2 | Not tasty and the texture was just nasty. | 0 |
| 3 | Stopped by during the late May bank holiday off Rick Steve recommendation and loved it. | 1 |
| 4 | The selection on the menu was great and so were the prices. | 1 |

Figure. *Naming the respective columns*

3. **Dataset Imbalance and Null Check**

```
df['Sentiment'].value_counts() # Balanced labels
```

| Sentiment | count |
|---|---|
| 1 | 500 |
| 0 | 500 |

dtype: int64

```
df.isnull().sum() # no null
```

| | 0 |
|---|---|
| ReviewText | 0 |
| Sentiment | 0 |

dtype: int64

Figure. *Label Data are balanced and having no null values*

4. **Expand Contraction**

`text = contractions.fix(text)`

The contractions.fix(text) method from the contractions module replaces contractions (like "I'm" → "I am"). This is critical for standardizing the text input, making it easier to tokenize and analyze.

The values like 60+ and Dec-18 have been replaced with range values to have consistency as shown.

5. **Add Space Around Punctuation**

**`text = text.translate(str.maketrans({key: f" {key} " for key in string.punctuation}))`**

This line ensures that punctuation marks are surrounded by spaces. For example, " Everything was fresh and delicious!" becomes " Everything was fresh and delicious !" which makes punctuation easier to separate from words during tokenization.

6. **Remove Punctuation**

`text = text.translate(str.maketrans('', '', string.punctuation))`

This removes all punctuation from the text (now that spaces have been added around them). After this step, only the words remain.

7. **Convert to Lowercase**

`text = text.lower()`

Converting all text to lowercase ensures that the words " Omelets" and " omelets" are treated the same during processing.

8. **Removing numerical data**

**`text = text.translate(str.maketrans('', '', string.digits))`**

This line of code is responsible for removing numerical digits from the input text during the preprocessing stage

### 9. Lemmatization

```
lemmatizer = WordNetLemmatizer()
```

```
lemmatized_words = [lemmatizer.lemmatize(word) for word in text]
```

Lemmatization reduces words to their base or dictionary form (lemma). For example, " driving" becomes "drive," and "better" might become "good," depending on the context.

### 10. Handling Stop Words using NLTK

```
# stop-words in english language
nltk.download('stopwords')

from nltk.corpus import stopwords
STOP_WORDS = stopwords.words('english')
print(STOP_WORDS)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

### 11. Reshuffling the processed Dataframe

May be worth shuffling the data to prevent any ordering having an influence on the performance.

```
#may be worth shuffeling the data to prevent any ordering having an influence on the performance

df = df.sample(frac=1, random_state=1)
df.reset_index(drop=True, inplace=True)

df.head()
```

| | ReviewText | Sentiment | processed_sentence |
|---|---|---|---|
| 0 | My gyro was basically lettuce only. | 0 | gyro basically lettuce |
| 1 | It kept getting worse and worse so now I'm officially done. | 0 | kept getting worse worse officially done |
| 2 | I am far from a sushi connoisseur but I can definitely tell the difference between good food and bad food and this was certainly bad food. | 0 | far sushi connoisseur definitely tell difference good food bad food certainly bad food |
| 3 | The staff are great, the ambiance is great. | 1 | staff great ambiance great |
| 4 | By this time our side of the restaurant was almost empty so there was no excuse. | 0 | time side restaurant almost empty excuse |

Figure. *Reshuffled dataframe after all pre-processing*

### 12. Distribution of Processed Sentence by reviews text by Sentiments

```
# The distribution of sentiments
df[["processed_sentence", "Sentiment"]].groupby('Sentiment').count().plot(kind='bar')
```

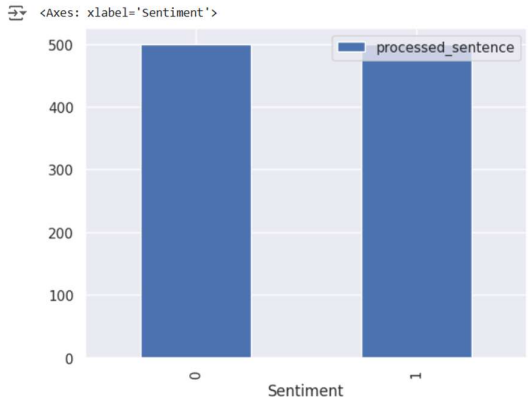



Figure. *Distribution of Reviews by Sentiment – Positive (1) or Negative (0) reviews*

### 13. Box Plot Visualization

The plot is showing the distribution of review lengths in training data.

Understanding the distribution of review lengths can help you decide if you need to truncate or pad them to a similar length for input to your models. Extreme outliers may also be worth investigating to understand what type of reviews they represent.

```
# review_len: This is a pandas Series that you created to store the number of words in each preprocessed review
review_len = pd.Series([len(review.split()) for review in df['processed_sentence']])

# The distribution of review text lengths
review_len.plot(kind='box')
```

```
<Axes: >
```

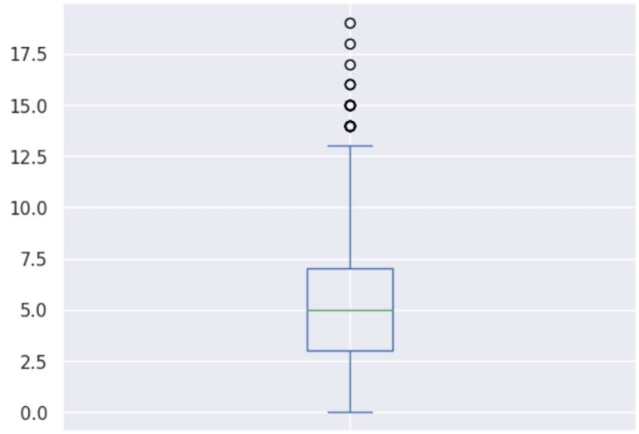



Figure. *Showing the distribution of review lengths in training data*

### 14. Visualization on how sentences are in the training data

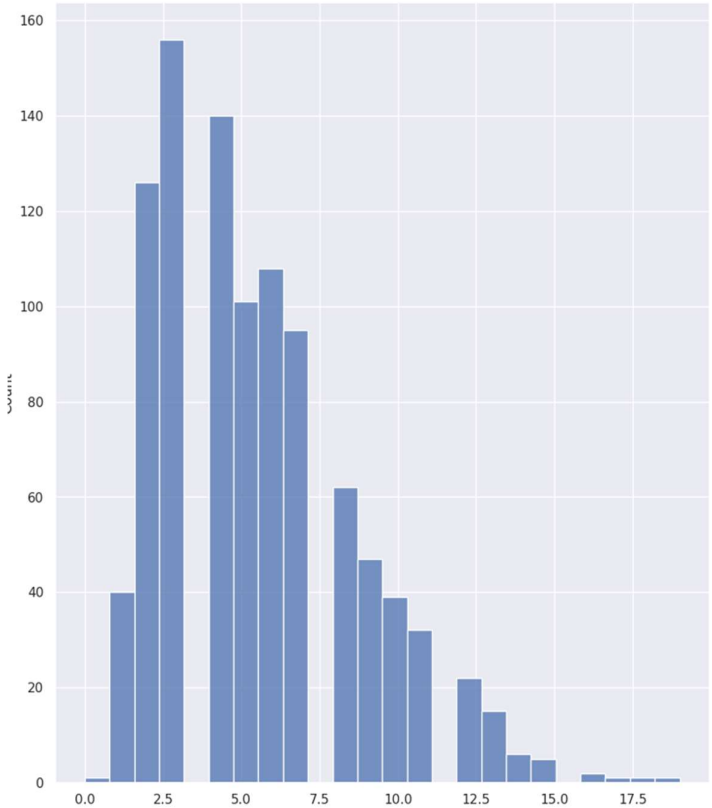



Figure. *Histogram showing how long our sentences are in the training data*

**15. WordCount for Review Text**

The word cloud provides a visual summary of the frequent words in negative/positive reviews, making it easy to identify key terms.

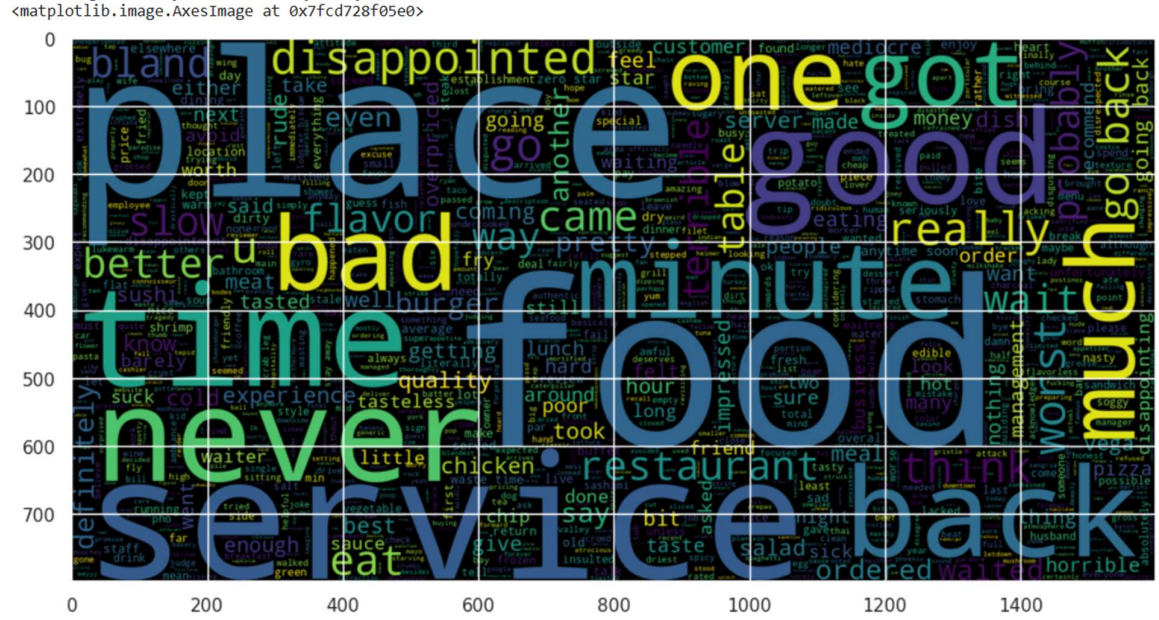



Figure. *Word Count for Negative Reviews*


Figure. *Word Count for Positive Reviews*

## VI. PERFORMING CLASSIFICATION & EXPLANATION

## 1. Logistic Regression

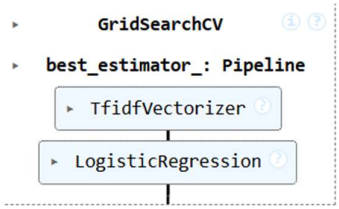
Defines two pipelines for logistic regression: one using **TF-IDF vectorization (tfpipe)** and another using **Bag-of-Words (pipe).** Each pipeline includes:

Vectorization:

**TF-IDF (tfpipe):** TfidfVectorizer converts text into numerical representations based on word frequency and importance (TF-IDF).

**Grid Search**: GridSearchCV is used to find the best hyperparameters for the model and vectorizer (like C for regularization and ngram_range for considering word combinations).

```
tfpipe = Pipeline([
    ('tfidf', text_learn.TfidfVectorizer()),
    ('logistic_regression', linear_model.LogisticRegression(max_iter=250))
])
```




```
Accuracy: 0.765
F1 score: 0.7648530331457161
Recall: 0.765
Precision: 0.7656641604010025
Jaccard_ Score:  0.6194331983805668

 confusion matrix:
 [[79 21]
 [26 74]]

 clasification report:
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       100
           1       0.78      0.74      0.76       100

    accuracy                           0.77       200
   macro avg       0.77      0.77      0.76       200
weighted avg       0.77      0.77      0.76       200
```

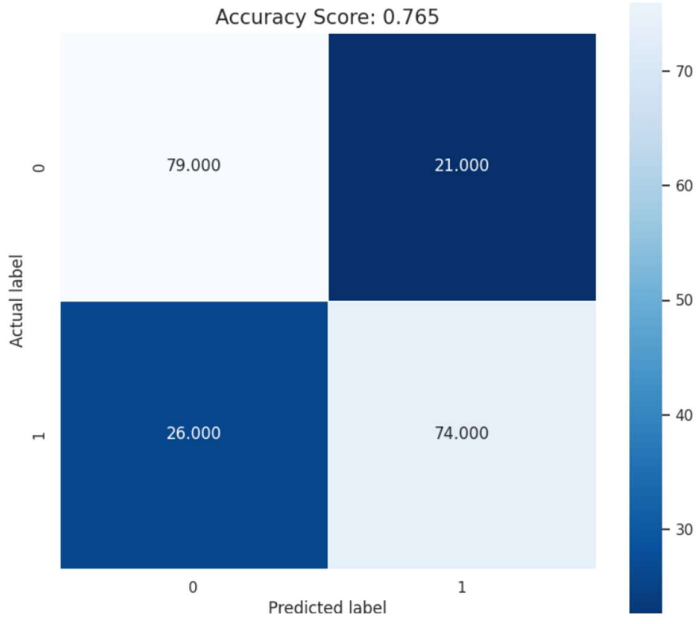
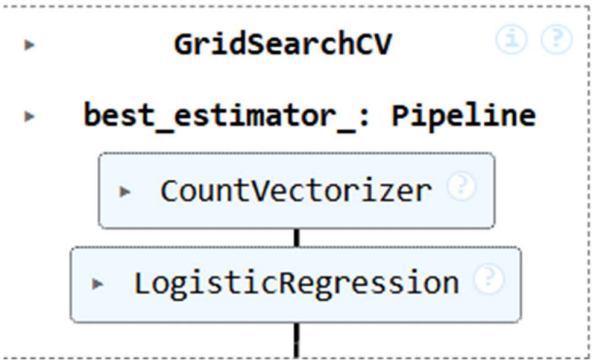



Figure. *Confusion Matrix for Logistic Regression after applying TfidfVectorizer*

**Bag-of-Words (pipe):** CountVectorizer creates a simple count of words in each document.
Model: LogisticRegression is used for binary classification (Sentiment 0 or 1).




```
Accuracy: 0.77
F1 score: 0.7676767676767676
Recall: 0.77
Precision: 0.78125
Jaccard_ Score:  0.6260162601626016

 confusion matrix:
 [[87 13]
 [33 67]]

 clasification report:
              precision    recall  f1-score   support

           0       0.72      0.87      0.79       100
           1       0.84      0.67      0.74       100

    accuracy                           0.77       200
   macro avg       0.78      0.77      0.77       200
weighted avg       0.78      0.77      0.77       200
```

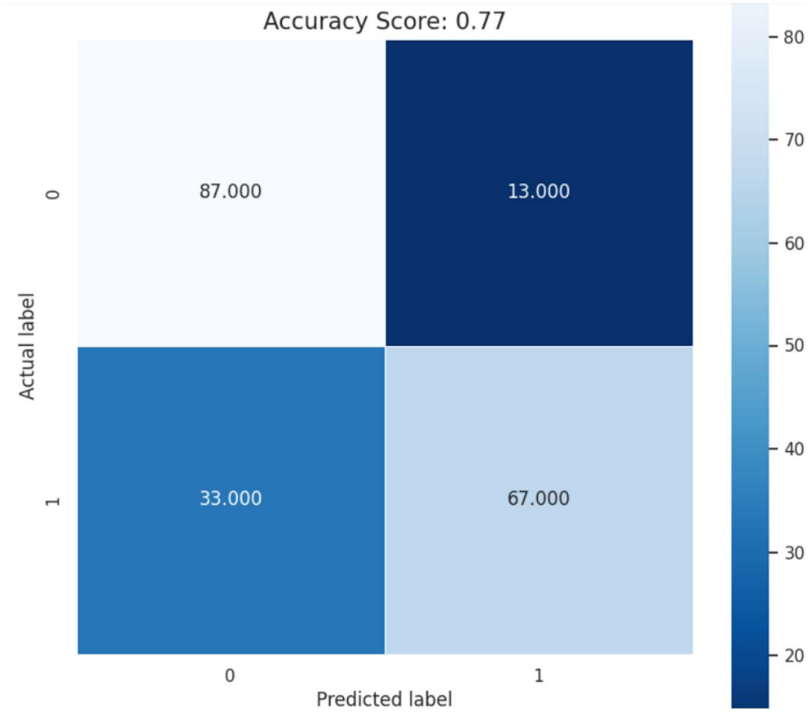



Figure. *Confusion Matrix for Logistic Regression after applying CountVectorizer*

## 2. NAÏVE BAYES CLASSIFICATION

Uses a pipeline to train a Bernoulli Naive Bayes model for sentiment classification. It vectorizes the text using TF-IDF and then applies the Naive Bayes algorithm to predict sentiment labels based on the probabilities of words belonging to each class. The model is trained on training data and is then ready to make predictions on new reviews.

```
Accuracy: 0.755
F1 score: 0.7546995068959476
Recall: 0.755
Precision: 0.7562556526982211
Jaccard_ Score:  0.606425702811245

 confusion matrix:
 [[72 28]
 [21 79]]

 clasification report:
              precision    recall  f1-score   support

           0       0.77      0.72      0.75       100
           1       0.74      0.79      0.76       100

    accuracy                           0.76       200
   macro avg       0.76      0.76      0.75       200
weighted avg       0.76      0.76      0.75       200
```

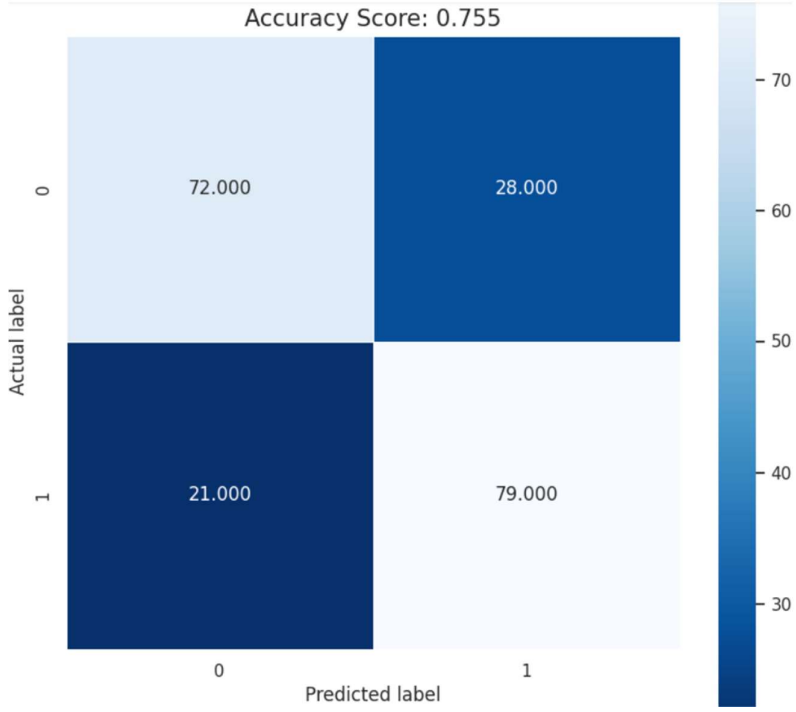



Figure. *Confusion Matrix for Naive Bayes after applying TfidfVectorizer*

*Why Bernoulli Naive Bayes is Used Here*:

Binary Features: TF-IDF features are typically binary (presence or absence of words), making Bernoulli Naive Bayes a suitable choice.
Text Classification: Naive Bayes algorithms are widely used for text classification tasks like sentiment analysis.
Efficiency: Naive Bayes models are generally fast to train and predict, making them efficient for large datasets.

## 3. SUPPORT VECTOR CLASSIFIER (SVC)

Here set up a pipeline to perform sentiment classification using an SVC (Support Vector Classifier), a powerful algorithm from the SVM family.
It pre-processes the text with Bag-of-Words and TF-IDF, then applies the SVC algorithm to find the optimal hyperplane for separating positive and negative reviews. Hyperparameter tuning is likely done using GridSearchCV to optimize the model's performance.

```
from sklearn.svm import SVC, LinearSVC
pipeline_svm = Pipeline([
    ('bow', text_learn.CountVectorizer()),
    ('tfidf', text_learn.TfidfTransformer()),
    ('classifier', SVC()),])
```

```
Accuracy: 0.78
F1 score: 0.7792051384985951
Recall: 0.78
Precision: 0.7840909090909091
Jaccard_ Score:  0.639344262295082

 confusion matrix:
 [[84 16]
 [28 72]]

 clasification report:
              precision    recall  f1-score   support

           0       0.75      0.84      0.79       100
           1       0.82      0.72      0.77       100

    accuracy                           0.78       200
   macro avg       0.78      0.78      0.78       200
weighted avg       0.78      0.78      0.78       200
```

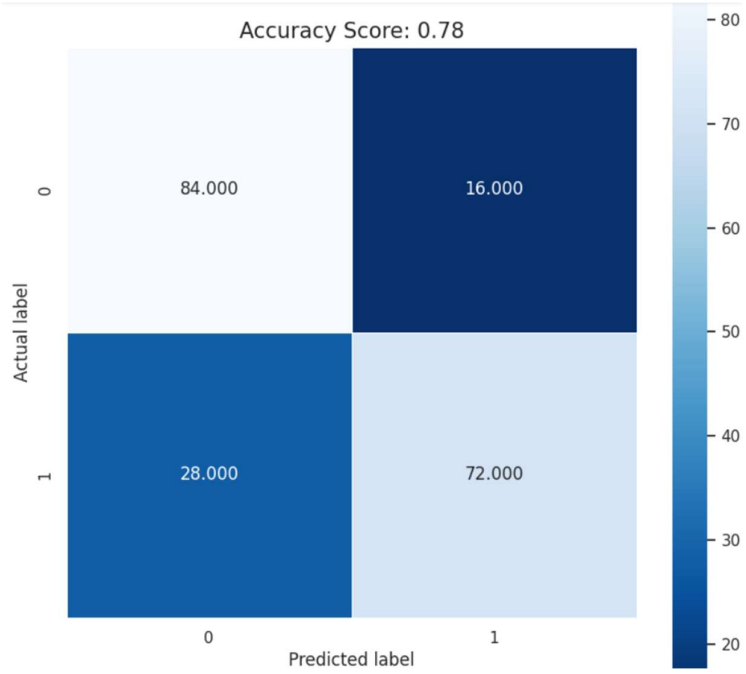



Figure. *Confusion Matrix for Support Vector Classifier after applying TfidfVectorizer and CountVectorizer*

*Why SVC is Used Here:*

Effectiveness in High Dimensions: SVC works well with high-dimensional data like text, where there are many features (words).

Flexibility with Kernels: The kernel trick allows SVC to adapt to different types of data and relationships between features.

Robustness to Outliers: SVC is relatively robust to outliers due to its focus on support vectors.

## 4. RANDOM FOREST CLASSIFIER

Uses a pipeline to train a RandomForestClassifier for sentiment classification. It pre-processes text data using Bag-of-Words and TF-IDF, and then applies the random forest algorithm to create an ensemble of decision trees for robust and accurate sentiment prediction. The parameters are set to handle potential class imbalance and ensure reproducibility.

```
pipeline_rfc = Pipeline([
    ('bow', text_learn.CountVectorizer()),
    ('tfidf', text_learn.TfidfTransformer()),
    ('classifier',
RandomForestClassifier(class_weight="balanced",random_state=101)),])
```

```
Accuracy: 0.75
F1 score: 0.7469379491851401
Recall: 0.75
Precision: 0.7627154266498528
Jaccard_ Score:  0.6

 confusion matrix:
 [[86 14]
 [36 64]]

 clasification report:
              precision    recall  f1-score   support

           0       0.70      0.86      0.77       100
           1       0.82      0.64      0.72       100

    accuracy                           0.75       200
   macro avg       0.76      0.75      0.75       200
weighted avg       0.76      0.75      0.75       200
```

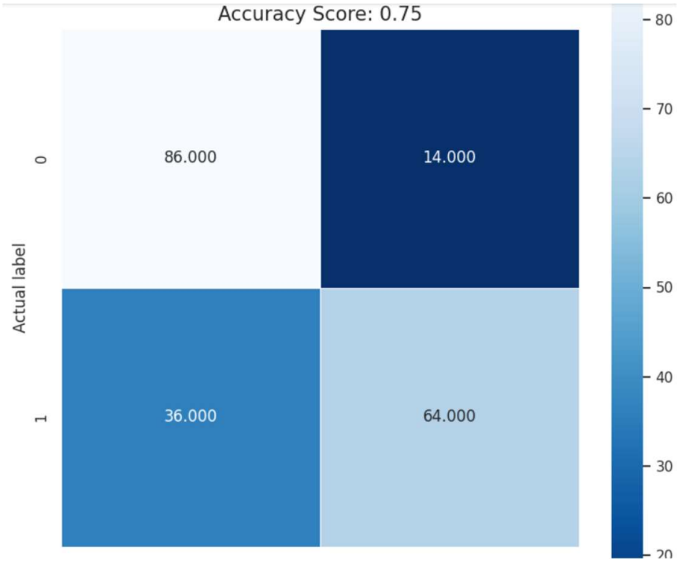



Figure. *Confusion Matrix for Random Forest Classifier after applying TfidfVectorizer and CountVectorizer*

Why RandomForestClassifier is Used Here:

High Accuracy: Random forests are known for their high accuracy and ability to handle complex relationships in data.

Robustness: They are less prone to overfitting compared to individual decision trees.

Feature Importance: Random forests can provide insights into which features are most important for making predictions.

## 5. GRADIENT BOOSTING CLASSIFIER

Uses a pipeline to train a GradientBoostingClassifier for sentiment classification. It vectorizes text data using TF-IDF and then applies the gradient boosting algorithm to create an ensemble of weak learners that sequentially improve the model's performance.

```
Accuracy: 0.735
F1 score: 0.7307936507936509
Recall: 0.735
Precision: 0.7506666666666666
Jaccard_ Score:  0.5810276679841897

 confusion matrix:
 [[86 14]
 [39 61]]

 clasification report:
              precision    recall  f1-score   support

           0       0.69      0.86      0.76       100
           1       0.81      0.61      0.70       100

    accuracy                           0.73       200
   macro avg       0.75      0.73      0.73       200
weighted avg       0.75      0.73      0.73       200
```

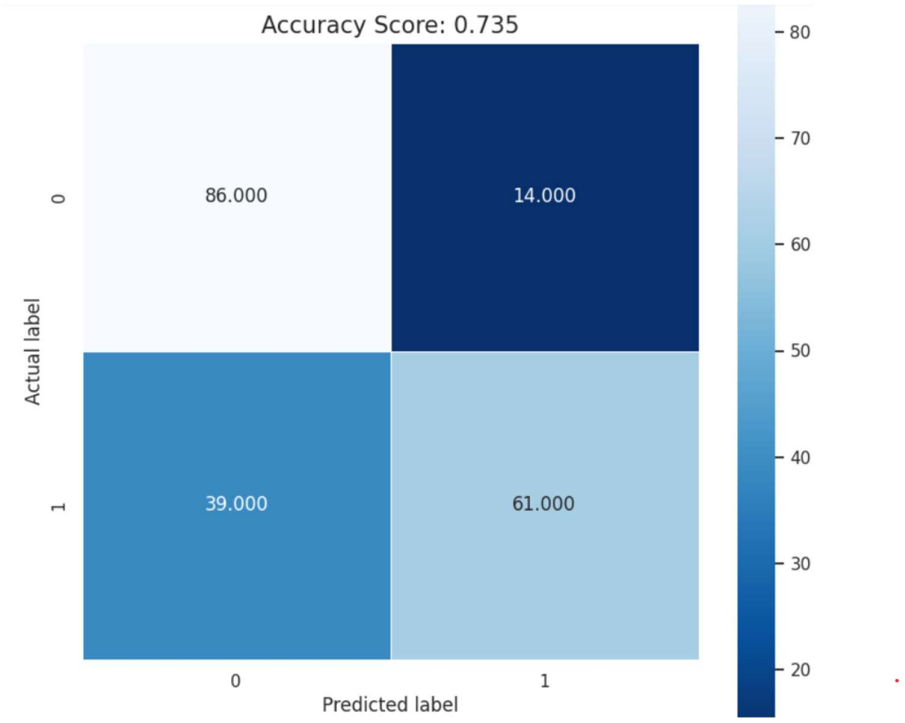



Figure. *Confusion Matrix for Gradient Boosting Classifier after applying TfidfVectorizer*

*Other classification model tried is *K Neighbors Classifier* applying TfidfVectorizer

## MODEL PERFORMANCE COMPARISON:

| Classifier Model | Accuracy (%) | F1 Score | Recall | Precision | Jaccard Score |
|---|---|---|---|---|---|
| Logistic Regression | 77.00 | 0.77 | 0.77 | 0.78 | 0.62 |
| Naïve Bais (BernoulliNB) | 75.50 | 0.75 | 0.75 | 0.75 | 0.60 |
| ***Support Vector Machine*** | ***78.00*** | ***0.78*** | ***0.78*** | ***0.78*** | ***0.64*** |
| Random Forest Classifier | 75.00 | 0.74 | 0.75 | 0.76 | 0.60 |
| Gradient Boosting Classifier | 73.50 | 0.73 | 0.73 | 0.75 | 0.58 |
| K-Neighbors Classifier | 71.50 | 0.71 | 0.71 | 0.72 | 0.55 |

Table. Comparative Analysis of *Performance of the Classification models using the various metrices*

**Key Observations / Results Comparison:**

**SVM** stands out as the top-performing model, showing the highest accuracy and Jaccard score, making it the best choice for balanced performance across all metrics.

**Logistic Regression** closely follows **SVM** with high precision and comparable accuracy, making it a reliable alternative if interpretability and lower computational cost are desired.

**Naïve Bayes** performs reasonably well across metrics, making it a good choice for speed and efficiency but lacking the robustness of **SVM and Logistic Regression**.

**Random Forest** and **Gradient Boosting** show potential with decent precision and recall but lag behind in accuracy, suggesting they might be suitable for scenarios requiring robustness to overfitting or complex data structures.

**KNN** has the lowest metrics, indicating it may not be ideal for this problem, particularly in larger datasets or high-dimensional spaces.

**Recommendation**: Based on this comparison, SVM and Logistic Regression are likely the best choices for maximizing accuracy and balanced metric performance.

*Best Performing Models*: Both SVM and Logistic Regression are the top-performing models for this dataset, with SVM being slightly superior in overall performance.

*Worst Performing Model*: KNN is the least effective choice in this comparison, as it lags behind in all key metrics, suggesting it may not be suitable for this particular classification task..

*Balanced Performance*: Logistic Regression is the most balanced model for this classification task, offering reliable predictive performance across all key metrics, making it suitable for tasks where both precision and recall are critical.

**Hyperparameter tuning** was also conducted to identify the optimal parameters for few models, ultimately enhancing the performance of the machine learning models. This tuning process was carried out using both **GridSearchCV** which systematically search through a range of parameter values, with GridSearchCV exhaustively exploring all possible combinations.

It has been found that GridSearchCV provides better accuracy but at a higher computational cost. This is due to the fact GridSearchCV is an exhaustive search method for hyperparameter tuning. It evaluates all possible combinations of the specified hyperparameters in a grid.

There is other tuning technique like **RandomizedSearchCV, Hyperband** that can be tried which claims to be more efficient hyperparameter optimization technique designed to address some of the limitations of

GridSearchCV, particularly regarding computational cost and time efficiency. It is especially useful when dealing with large hyperparameter spaces and expensive models.

## VII. BERT-Based Model for Classification

**Modelling BERT**:
The pre-trained BERT model is loaded as a Keras layer from Tensorflow. We have created a very simple fine-tuned model, with the preprocessing model, the selected BERT model, one Dense and a Dropout layer.
The code utilizes a pre-trained BERT model (small_bert/bert_en_uncased_L-4_H-512_A-8 in this case) and fine-tunes it for sentiment classification. Here's a breakdown:

Preprocessing: A BERT-specific preprocessing model is used to transform text into numerical token IDs and create input masks.
Classifier: A simple classifier (Dense layer with Dropout) is added on top of BERT's pooled output to predict sentiment.
Fine-tuning: The entire model (BERT + classifier) is trained on your dataset, adjusting BERT's weights to better capture sentiment-related patterns in your data.

```
BERT model selected           : https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1
Preprocess model auto-selected: https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3
```

***AdamW*** is commonly used in sentiment analysis for training deep learning models like BERT. These models have a large number of parameters, and AdamW helps to train them efficiently and effectively, leading to improved sentiment classification accuracy.

```
[ ] tf.keras.utils.plot_model(classifier_model)
```

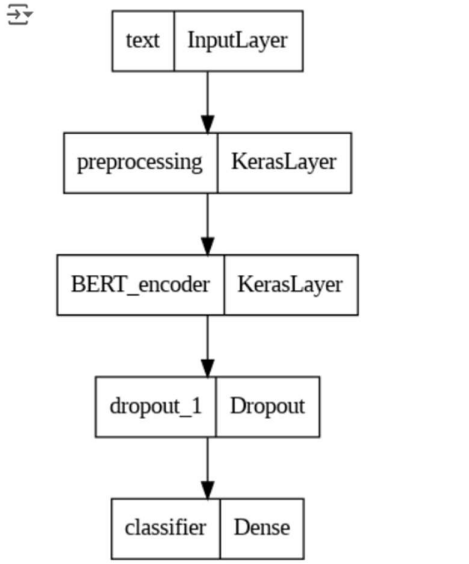



Figure: *Showing steps in Modelling BERT*

Loss function: Since this is a binary classification problem and the model outputs a probability (a single-unit layer), we'll use losses.BinaryCrossentropy loss function.

```
classifier_model.summary()
```

```
Model: "model_1"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 text (InputLayer)           [(None,)]                 0         []

 preprocessing (KerasLayer)  {'input_mask': (None, 128)  0        ['text[0][0]']
                             , 'input_type_ids': (None,
                              128),
                              'input_word_ids': (None,
                             128)}

 BERT_encoder (KerasLayer)   {'pooled_output': (None, 5  2876364   ['preprocessing[0][0]',
                             12),                        9          'preprocessing[0][1]',
                              'encoder_outputs': [(None              'preprocessing[0][2]']
                             , 128, 512),
                              (None, 128, 512),
                              (None, 128, 512),
                              (None, 128, 512)],
                              'default': (None, 512),
                              'sequence_output': (None,
                              128, 512)}

 dropout_1 (Dropout)         (None, 512)               0         ['BERT_encoder[0][5]']

 classifier (Dense)          (None, 1)                 513       ['dropout_1[0][0]']

=================================================================================================
Total params: 28764162 (109.73 MB)
Trainable params: 28764161 (109.73 MB)
Non-trainable params: 1 (1.00 Byte)
```

Figure: *Showing BERT model Summary*

```
[ ] tf.keras.utils.plot_model(model=classifier_model,show_shapes=True,dpi=76)
```

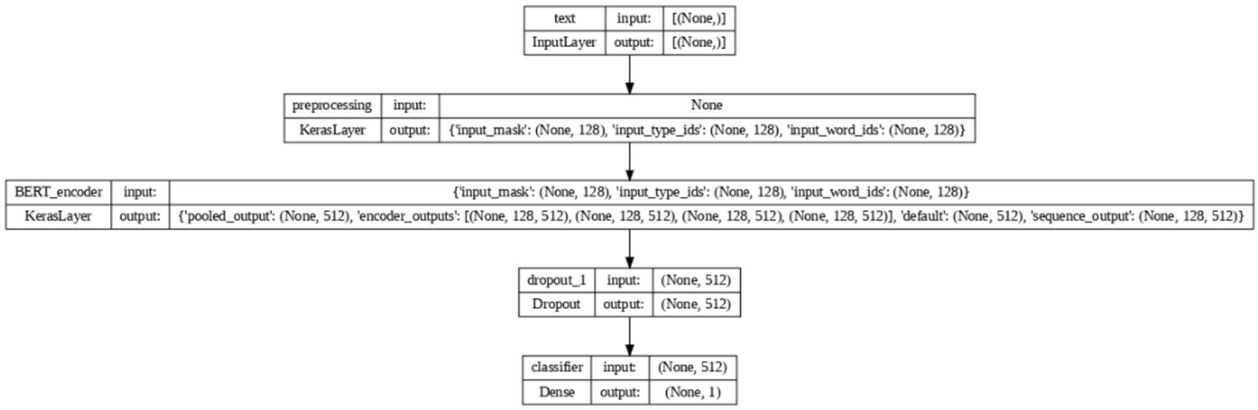



Figure: *Plotting steps in Modelling BERT*

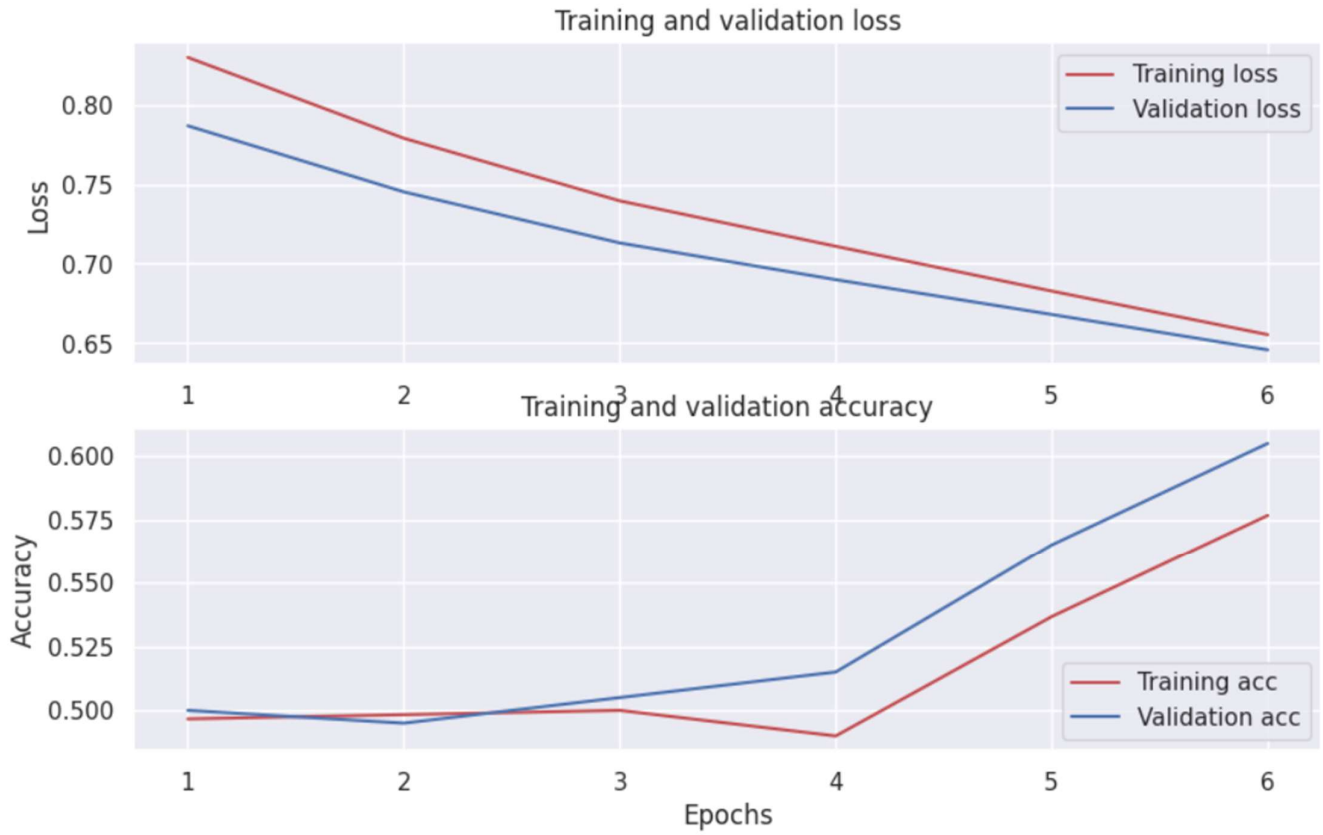



Figure: Graph *showing Training vs Loss and Accuracy*

**Evaluate the model**:

```
[ ]  loss, accuracy = classifier_model.evaluate(X_test, y_test)

     print(f'Loss: {loss}')
     print(f'Accuracy: {accuracy}')
```

```
7/7 [==============================] - 15s 2s/step - loss: 0.6378 - binary_accuracy: 0.6200
Loss: 0.6378467082977295
Accuracy: 0.6200000047683716
```

```
[[97  3]
 [73 27]]

Mean Absolute Error: 0.38
Mean Squared Error: 0.38
Root Mean Squared Error: 0.6164414002968976

              precision    recall  f1-score   support

           0       0.57      0.97      0.72       100
           1       0.90      0.27      0.42       100

    accuracy                           0.62       200
   macro avg       0.74      0.62      0.57       200
weighted avg       0.74      0.62      0.57       200

Accuracy: 0.62
AUC: 0.62
```

Figure: *Showing Confusion Matrix and Classification Report*

The **BERT** model in this case seems to have a **good grasp on negative sentiment** but needs improvement in identifying positive sentiment. Further investigation and potential adjustments to fine-tuning, data, or model architecture might be needed to enhance its performance, especially for positive samples.

**Comparative Analysis of BERT Model vs Other Classification models**

Based on accuracy, the BERT model's performance (62%) appears to be lower than other traditional machine learning models like Logistic Regression, Naive Bayes, SVC, Random Forest, and Gradient Boosting (all around 76-77%).

***Possible Explanations***:

Fine-tuning: BERT models require careful fine-tuning for optimal performance. It's possible that the fine-tuning process for your BERT model was not fully optimized, leading to lower accuracy compared to other models.

Data: BERT models often benefit from large datasets. If your dataset is relatively small, the other models might have been able to learn better representations, leading to higher accuracy.

Complexity: BERT models are highly complex and can be sensitive to hyperparameter settings. It's possible that the chosen hyperparameters were not ideal for your dataset, affecting the model's performance.

***Considerations***:

-Accuracy alone might not be the best metric for comparison. Consider other metrics like F1-score, precision, recall, and AUC to get a more comprehensive view of each model's performance.

-BERT models are known for capturing contextual information, which might not be fully reflected in accuracy alone. Evaluate the models on qualitative aspects like understanding nuances in sentiment.

-Computational resources can be a factor. BERT models require significant computational power for training, while other models are generally more lightweight.

***Recommendations***:

-Further fine-tune the BERT model by experimenting with different hyperparameters and fine-tuning -strategies.

-Consider using a larger dataset if possible, to improve BERT's performance.

-Evaluate models based on a wider range of metrics to get a more complete picture of their strengths and weaknesses.

-Explore other BERT variations or pre-trained models that might be better suited for your specific task.

# VIII. PERFROMING TOPIC DETECTION

***LDA Model Setup:***

Topic detection has been performed using Latent Dirichlet Allocation (LDA) from genism NLP Package which may create and query corpus. It operates by constructing word embeddings or vectors, which are then used to model topics.

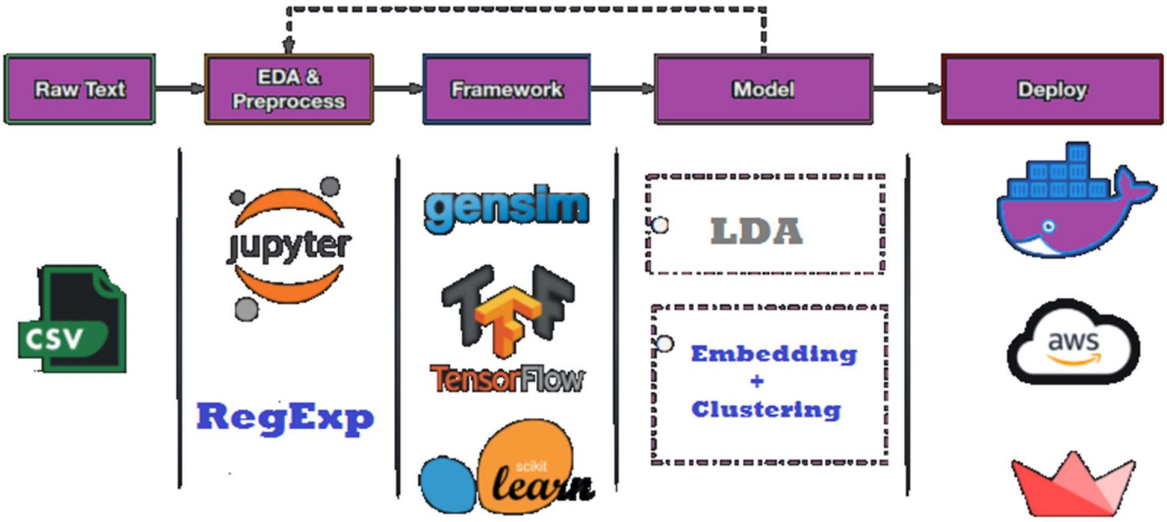



Figure. *Typic flow in Topic detection*

```
# number of topics
num_topics = 10
```

```
    # Build LDA model
    lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                           id2word=id2word,
                                           num_topics=num_topics)

    # Print the Keyword in the 10 topics
    pprint(lda_model.print_topics())
    #doc_lda = lda_model[corpus]
```

```
WARNING:gensim.models.ldamulticore:too few updates, training might not converge; consider increasin
[(0,
  '0.039*"place" + 0.013*"really" + 0.011*"service" + 0.011*"ever" + '
  '0.011*"good" + 0.010*"like" + 0.010*"chicken" + 0.009*"sushi" + '
  '0.007*"back" + 0.007*"would"'),
 (1,
  '0.025*"great" + 0.021*"service" + 0.020*"place" + 0.015*"food" + '
  '0.010*"back" + 0.009*"go" + 0.007*"good" + 0.007*"one" + 0.007*"get" + '
  '0.007*"time"'),
 (2,
  '0.022*"place" + 0.019*"great" + 0.018*"food" + 0.008*"would" + 0.008*"eat" '
  '+ 0.008*"got" + 0.008*"never" + 0.008*"service" + 0.007*"minute" + '
  '0.007*"came"'),
 (3,
  '0.015*"experience" + 0.014*"good" + 0.013*"like" + 0.009*"great" + '
  '0.009*"go" + 0.008*"service" + 0.008*"burger" + 0.008*"absolutely" + '
  '0.008*"food" + 0.006*"one"'),
```

***Topic Interpretation***:

Each topic is represented by a set of keywords, with each word's weight indicating its importance within the topic. This LDA model has identified **10 distinct topics** that are primarily focused on themes related to food, service, and dining experiences, likely from reviews or customer feedback. Below is a breakdown of each topic:

1. **Topic 0**: This topic centers on places, with a focus on service and specific items like *chicken* and *sushi*. Keywords like *really*, *ever*, and *back* suggest a mixed sentiment, possibly reflecting first-time or returning customer experiences.
2. **Topic 1**: This topic emphasizes *great* service and a *great place*, highlighting positive experiences. Keywords like *food*, *good*, *go*, and *get* indicate a general satisfaction with the food and service, as well as a desire to return.

3. **Topic 2**: Here, the focus is on *place*, *great*, and *food*, suggesting overall customer satisfaction. Words like *eat*, *got*, and *minute* might point to details on waiting times, specific meals, or food quality.
4. **Topic 3**: This topic is oriented around *experience* and *like*, possibly capturing subjective opinions. Words like *burger*, *absolutely*, and *go* hint at strong opinions, potentially highlighting memorable or standout experiences.
5. **Topic 4**: Food and service are prominent, but the inclusion of *bad* and *love* implies mixed or contrasting reviews, where both positive and negative aspects of food and service are noted.
6. **Topic 5**: This topic focuses on the *quality* of food and service with keywords like *wait*, *selection*, and *server*, which may indicate customer feedback on wait times, food variety, and staff interaction.
7. **Topic 6**: Emphasis on *back*, *time*, and *go* suggests customer willingness to revisit the place. Words like *many*, *probably*, and *restaurant* indicate decisions or expectations about returning.
8. **Topic 7**: This topic emphasizes *service*, *friendly*, *staff*, and *atmosphere*, suggesting a focus on the ambiance and friendliness of the service, which are key factors in customer satisfaction.
9. **Topic 8**: Keywords like *good*, *food*, *great*, and *like* depict positive general sentiments about the dining experience, while *meal*, *pretty*, and *well* indicate approval of both the food and presentation.
10. **Topic 9**: Words such as *place*, *best*, *time*, *back*, and *worst* show contrasting sentiments, with some customers considering it the "best" and others the "worst." Terms like *steak* and *vega* might indicate specific menu items or locations that polarized opinion.

*Summary*: These topics highlight common themes in dining experiences, such as food quality, service, ambiance, and specific memorable aspects (like chicken, steak, and sushi). The model captures both positive (e.g., great, good, best) and negative sentiments (e.g., bad, worst), indicating a diverse range of customer feedback. This information can be useful for identifying key drivers of satisfaction and dissatisfaction within dining reviews.

***Topic model visualization using pyLDAvis:***

Further we tried visualizing the topics discovered by your LDA model using an interactive visualization provided by pyLDAvis

```
# Visualize the topics
pyLDAvis.enable_notebook()

LDAvis = pyLDAvis.gensim_models.prepare(lda_model, corpus, id2word)
```

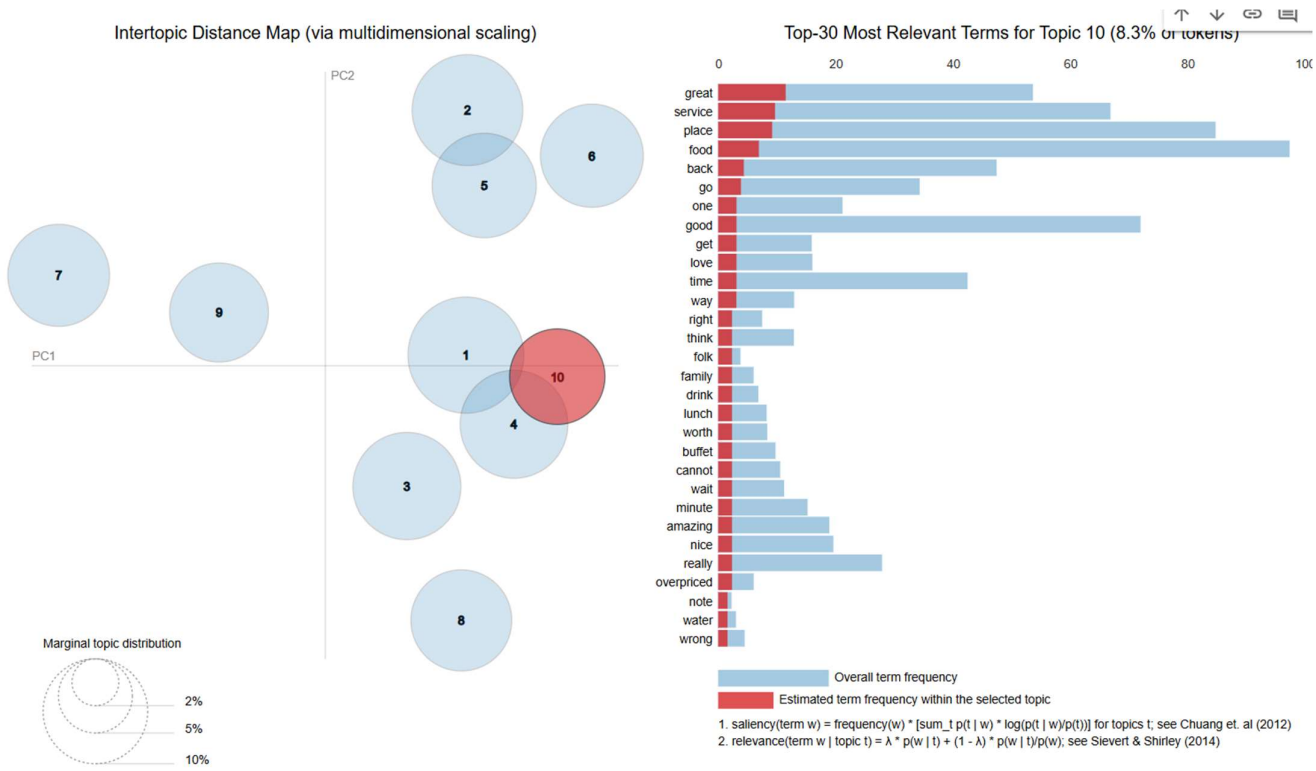



Figure. *The pyLDAvis visualization provides an interactive way to understand the topics discovered by your LDA model.*

*Here's a breakdown of the key elements:*

**Intertopic Distance Map (Left Panel):** This scatterplot shows the relationship between the different topics in your model. Topics closer together are more similar, while topics farther apart are more distinct. The size of the circles represents the prevalence of each topic in the corpus.

**Top Terms (Right Panel):** When you select a topic in the Intertopic Distance Map, this panel updates to show the most relevant terms for that topic. This helps you understand the semantic meaning of each topic.

**Relevance Metric (Slider):** You can adjust the relevance metric (lambda) slider to balance the prominence of salient terms (high lambda) and overall frequency (low lambda). This can reveal different aspects of the topics.

This visualization helps you explore the topics, understand their relationships, assess their quality, and potentially gain insights for model tuning. By examining the intertopic distance map, term relevance, and document view (if available), you can gain a deeper understanding of the themes and patterns in your text data.

# IX. CONCLUSION

This study explored various machine learning techniques for sentiment analysis of customer reviews. Traditional algorithms like Logistic Regression, Naive Bayes, and Support Vector Machines demonstrated good performance, achieving accuracies around 76-77%. While the BERT model achieved lower accuracy (62%), its potential for capturing contextual information warrants further investigation and fine-tuning. Comparing these models revealed trade-offs between accuracy, complexity, and computational resources. Overall, this project highlights the effectiveness of machine learning for understanding customer sentiment, with opportunities for further refinement and exploration of advanced techniques like BERT. By leveraging

these insights, businesses can gain valuable understanding of customer feedback to improve products and services.

## X. REFERENCES


Matloff, N. Statistical Regression and Classification: From Linear Models to Machine Learning; Chapman and Hall/CRC: Boca Raton, FL, USA, 2017

Olivas, E.S. Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques; IGI Global: Hershey, PA, USA, 2009.

Webb G.I. (2011) Naïve Bayes. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_576

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

neptune.ai/blog/pyldavis-topic-modelling-exploration-tool-that-every-nlp-data-scientist-should-know
medium.com/@mandoiwanaga08/topic-modeling-visualization-f1181d82fc44

https://www.tensorflow.org/text/tutorials/classify_text_with_bert

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

https://www.iguazio.com/blog/mastering-ml-model-performance-best-practices-for-optimal-results/


## XI. KEY TERMS AND DEFINITIONS

*Accuracy* - The number of correct predictions divided by the total number of predictions.
*F1 Score* - harmonic mean of the recall and precision.
*Recall* - number of true positives divided by the sum of the number of true positives & number of false negatives.
*Precision* - number of true positives divided by sum of the number of true positives & the number of false positives.
*Jaccard score* (or Jaccard index) measures similarity between the predicted and actual classes. A higher Jaccard score indicates better performance.
*AUC (Area Under the Curve):* A measure of the model's ability to distinguish between classes (often used for binary classification).