MACHINE LEARNING
FINAL PROJECT
RESTAURANT REVIEW ANALYSIS

Devi Vara Prasad Rongala - **11565998**

Table of Contents:

Introduction :

There are several methods to analyze customer reviews for a restaurant, such as Naive Bayes, Regression, and Random Forest. We will be using these three models to analyze restaurant reviews and will explain each model in subsequent slides. Afterwards, we will compare the accuracy scores of each model to determine which one is best suited for the chosen dataset. Finally, we will identify and present the best model for the given dataset

Problem Statement :

Restaurant Review Analysis

We aim to build a Machine Learning model that can accurately detect the various sentiments expressed in a collection of English sentences or large paragraphs, such as reviews. The model's objective is to correctly identify the sentiment of users' reviews, which can be either positive or negative. Therefore, we will classify reviews as either positive or negative based on their sentiment.

Dataset :

We have used the Data set from kaggle and below is the hyperlink to the data set

https://www.kaggle.com/datasets/hj5992/restaurantreviews?resource=download

Data Preprocessing :

- We created a data frame review_df that has the columns Review and liked
- Download the data and libraries required for textual data preparation. It begins by loading the necessary libraries, including the CountVectorizer module from Scikit-learn, regular expression, string, and NLTK (Natural Language Toolkit), a toolkit for NLP (Natural Language Processing).The method then sets the English stopwords as a variable and downloads lemmatizer and stopword data from NLTK and Open Multilingual Wordnet (OMW). The WordNetLemmatizer, which is used to break down words into their root or base form, is then initialized.

In [1]:
```
import pandas as pd
# load data from csv file
filename = r"C:\Users\DevRo\Downloads\output.csv"
#convert csv to dataframe
reviews_df= pd.read_csv(filename)
```

In [2]:
```
reviews_df
```

Out[2]:

	Review	Liked
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 of...	1
4	The selection on the menu was great and so wer...	1
...	...	...
995	I think food should have flavor and texture an...	0
996	Appetite instantly gone.	0
997	Overall I was not impressed and would not go b...	0
998	The whole experience was underwhelming, and I ...	0
999	Then, as if I hadn't wasted enough of my life ...	0

1000 rows × 2 columns

In [3]:
```
print(reviews_df.columns)
```

In [23]:
```
#download nltk libraries for data preprocessing

import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
# Download stopwords and lemmatizer data from NLTK
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('omw-1.4')
print(nltk.data.find('corpora/omw-1.4.zip'))
# Download Open Multilingual Wordnet (OMW)
nltk.download('omw')

# Load the English stopwords
nltk.download('stopwords')
stopwords = stopwords.words('english')

# Initialize the WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

```
C:\Users\DevRo\AppData\Roaming\nltk_data\corpora\omw-1.4.zip
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\DevRo\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\DevRo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\DevRo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\DevRo\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package omw to
[nltk_data]     C:\Users\DevRo\AppData\Roaming\nltk_data...
[nltk_data]   Package omw is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\DevRo\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

- These code lines perform text preprocessing and vectorization on the Review column of the reviews_df dataframe.The first line removes all punctuation from the text using regular expression.The second line converts all text to lowercase using the .lower() method. This is done to standardize the text and ensure that uppercase and lowercase letters are treated as the same.The third and fourth lines vectorize the text data using a bag-of-words model. This converts the text data into a matrix of word counts, where each row corresponds to a review and

each column corresponds to a unique word in the corpus. The resulting matrix can be used as input for various machine learning models.The figure shows output after this step.

In [6]:
```
# Remove punctuation
reviews_df['Review'] = reviews_df['Review'].str.replace('[^\w\s]','')

# Convert all text to lowercase
reviews_df['Review'] = reviews_df['Review'].str.lower()

# Vectorize the text data using a bag-of-words model
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(reviews_df['Review'])
y = reviews_df['Liked']
```

Out[29]:

	Review	Liked
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 of...	1
4	the selection on the menu was great and so wer...	1
...	...	...
995	i think food should have flavor and texture an...	0
996	appetite instantly gone	0
997	overall i was not impressed and would not go back	0
998	the whole experience was underwhelming and i t...	0
999	then as if i hadnt wasted enough of my life th...	0

1000 rows × 2 columns

MODEL 1 - Naive Bayes:

The Naive Bayes classifier is a supervised machine learning algorithm, which is used for classification tasks, like text classification. It is also part of a family of generative learning algorithms, meaning that it seeks to model the distribution of inputs of a given class or category

Some best examples of the Naive Bayes Algorithm are sentimental analysis, classifying new articles, and spam filtration. Classification algorithms are used for categorizing new observations into predefined classes for the uninitiated data.

In [68]:
```
import csv
import random

class NaiveBayesClassifier:

    def __init__(self):
        self.positive_word_counts = {}
        self.negative_word_counts = {}
        self.total_positive_words = 0
        self.total_negative_words = 0

    def train(self, train_data):
        for review, label in train_data:
            words = review.split()
            for word in words:
                if label == 1:
                    self.positive_word_counts[word] = self.positive_word_counts.get(word, 0) + 1
                    self.total_positive_words += 1
                else:
                    self.negative_word_counts[word] = self.negative_word_counts.get(word, 0) + 1
                    self.total_negative_words += 1

    def classify(self, test_data):
        predictions = []
        for review, _ in test_data:
            words = review.split()
            positive_prob = 1
            negative_prob = 1
            for word in words:
                positive_prob *= (self.positive_word_counts.get(word, 0) + 1) / (self.total_positive_words + len(self.positive_w
                negative_prob *= (self.negative_word_counts.get(word, 0) + 1) / (self.total_negative_words + len(self.negative_w
            if positive_prob > negative_prob:
                predictions.append(1)
            else:
                predictions.append(0)
        return predictions

    def train_test_split(self, csv_file_path, test_size=0.2):
        with open(csv_file_path, 'r') as csv_file:
            csv_reader = csv.reader(csv_file)
            next(csv_reader) # Skip the header row
            data = [(row[0], int(row[1])) for row in csv_reader]
            random.shuffle(data)
            split_index = int(len(data) * (1 - test_size))
            train_data = data[:split_index]
            test_data = data[split_index:]
        return train_data, test_data

    def evaluate(self, test_data):
        predictions = self.classify(test_data)
        correct_predictions = 0
        for i in range(len(test_data)):
            if predictions[i] == test_data[i][1]:
                correct_predictions += 1
        accuracy = correct_predictions / len(test_data)
        return accuracy
```

init(self): Initializes the class by creating two empty dictionaries (positive_word_counts and negative_word_counts) and setting the counts of positive and negative words to 0.

train(self, train_data): Trains the Naive Bayes model on the given train_data. For each review in the train_data, it splits the review into words and updates the counts of each word in either the positive_word_counts or negative_word_counts dictionary depending on the label of the review.

classify(self, test_data): Classifies each review in the test_data using the trained Naive Bayes model. For each review, it splits the review into words and calculates the probability of the review being positive or negative based on the counts of each word in the positive_word_counts and negative_word_counts dictionaries. It then assigns a label of 1 if the probability of the review being positive is greater than the probability of the review being negative, otherwise it assigns a label of 0. The predicted labels are returned as a list.

train_test_split(self, csv_file_path, test_size=0.2): Splits the dataset in the csv_file_path into training and testing data. It reads the data from the csv file, skips the header row, shuffles the data, and splits it into training and testing data based on the test_size parameter. The training and testing data are returned as two separate lists.

evaluate(self, test_data): Evaluates the performance of the Naive Bayes model on the given test_data. It uses the classify method to predict the labels for the reviews in the test_data and compares them to the actual labels to calculate the accuracy of the model. The accuracy is returned as a float between 0 and 1.

Results :

This code creates an instance of the NaiveBayesClassifier class, trains it on a dataset (80% as training data), evaluates its performance on a test set(20% test data), and prints the accuracy and predicted labels for the test set.We could see Accuracy is 72%

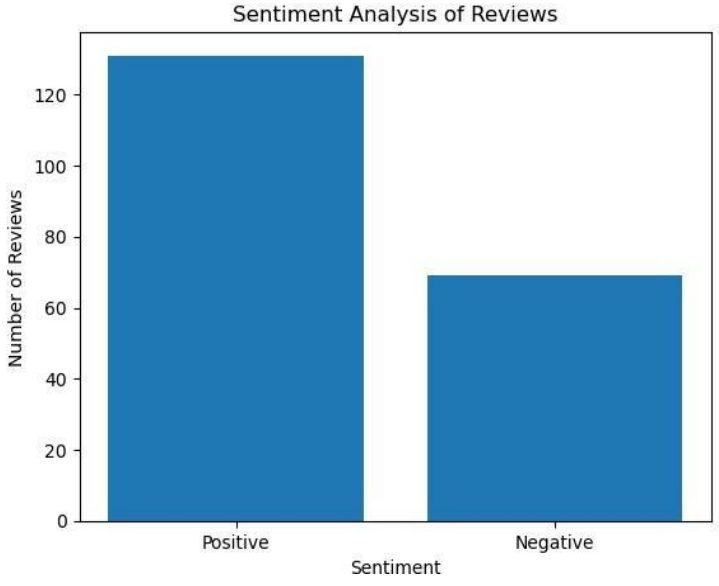
In [70]:
```
classifier1 = NaiveBayesClassifier()
train_data, test_data = classifier1.train_test_split(filename, test_size=0.2)
classifier1.train(train_data)
accuracy = classifier1.evaluate(test_data)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.725
```

In [71]:
```
predictions = classifier1.classify(test_data)
print(predictions)
```

```
[1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1,
1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0,
0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1]
```

In [72]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Convert test data to a Pandas DataFrame
test_data_df = pd.DataFrame({'review': test_data})

# Make predictions on test data and add predictions to DataFrame
predictions = classifier1.classify(test_data_df['review'])
test_data_df['sentiment'] = predictions

# Count the number of positive and negative reviews
positive_reviews = test_data_df[test_data_df['sentiment'] == 1]['review'].count()
negative_reviews = test_data_df[test_data_df['sentiment'] == 0]['review'].count()

# Visualize the counts using a bar chart
plt.bar(['Positive', 'Negative'], [positive_reviews, negative_reviews])
plt.title('Sentiment Analysis of Reviews')
plt.xlabel('Sentiment')
plt.ylabel('Number of Reviews')
plt.show()
```



```
In [73]: print(negative_reviews)
         69

In [74]: print(positive_reviews)
         131
```

Negative reviews = 69

Positive reviews =131

Model 2- Logistic Regression:

The type of statistical model is often used for classification and predictive analytic. Logistic regression estimates the probability of an event occurring such as voted or didn't vote, based on a given dateset of independent variables.

```
In [75]: import csv
         import math
         import numpy as np
         from sklearn.feature_extraction.text import CountVectorizer

         class LogisticRegression:
             def __init__(self, learning_rate=0.01, num_iterations=1000, print_loss=False):
                 self.learning_rate = learning_rate
                 self.num_iterations = num_iterations
                 self.print_loss = print_loss
                 self.weights = None
                 self.bias = None

             def fit(self, X, y):
                 # initialize weights and bias
                 n_samples, n_features = X.shape
                 self.weights = np.zeros(n_features)
                 self.bias = 0

                 # gradient descent
                 for i in range(self.num_iterations):
                     # calculate predicted values and gradient
                     y_pred = self.sigmoid(np.dot(X, self.weights) + self.bias)
                     dw = (1 / n_samples) * np.dot(X.T, (y_pred - y))
                     db = (1 / n_samples) * np.sum(y_pred - y)

                     # update weights and bias
                     self.weights -= self.learning_rate * dw
                     self.bias -= self.learning_rate * db

                     # print loss for each 100 iterations
                     if self.print_loss and i % 100 == 0:
                         loss = self.compute_loss(X, y)
                         print(f"Iteration {i}: Loss={loss}")

             def predict(self, X):
                 # predict binary values (0 or 1)
                 y_pred = np.round(self.sigmoid(np.dot(X, self.weights) + self.bias))
                 return y_pred.astype(int)

             def sigmoid(self, z):
                 # sigmoid activation function
                 return 1 / (1 + np.exp(-z))

             def compute_loss(self, X, y):
                 # compute cross-entropy loss
                 y_pred = self.sigmoid(np.dot(X, self.weights) + self.bias)
                 loss = -(1 / len(y)) * np.sum(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))
                 return loss

             def load_data(self, csv_file):
                 # load data from csv file
                 X, y = [], []
                 with open(csv_file, 'r') as f:
                     reader = csv.reader(f)
                     next(reader)
                     for row in reader:
                         X.append(row[0])
                         y.append(int(row[1]))
                 vectorizer = CountVectorizer(lowercase=True, stop_words='english')
                 X = vectorizer.fit_transform(X)
                 y = np.array(y)
                 return X.toarray(), y
```

The **__init__**method initializes the learning rate, number of iterations, and print loss parameters. The weights and bias are set to None initially.

The **fit** method performs gradient descent to train the logistic regression model. It takes two parameters, X and y, where X is a matrix of shape (n_samples, n_features) containing the input

features and y is a vector of shape (n_samples,) containing the binary labels. The method initializes the weights and bias to 0 and then iteratively updates them based on the gradient of the loss function. The loss function used is the cross-entropy loss.

The **predict** method takes a matrix X of shape (n_samples, n_features) and returns a binary vector of shape (n_samples,) containing the predicted labels for each input sample.

The **sigmoid** method takes a vector z and applies the sigmoid activation function element-wise to produce a vector of the same shape.

The **compute_loss** method takes a matrix X of shape (n_samples, n_features) and a vector y of shape (n_samples,) and computes the cross-entropy loss for the current weights and bias.

The **load_data** method takes a CSV file path and loads the data into memory as input features X and binary labels y. The data is loaded using the csv module, and the input features are transformed using the CountVectorizer from sklearn.feature_extraction.text, which converts text into a bag-of-words representation.

Results :

Trained a logistic regression model on our dataset loaded from a CSV file using the load_data method of the LogisticRegression class. The data is split into training and testing sets using the train_test_split function from the sklearn.model_selection module. The logistic regression model is trained on the training set using the fit method of the LogisticRegression class. The predict method is then used to make predictions on the test set, and the accuracy of the model is calculated as the proportion of correct predictions using the np.mean function. Finally, the accuracy is printed.We could see **Accuracy is 74%**

In [76]:
```
import csv
import numpy as np
from sklearn.model_selection import train_test_split


lr = LogisticRegression()
X, y = lr.load_data(filename)
```

In [77]:
```
# split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# train the model
lr.fit(X_train, y_train)

# make predictions on test set
y_pred = lr.predict(X_test)

# calculate accuracy
accuracy = np.mean(y_pred == y_test)
print(f"Accuracy: {accuracy}")
```

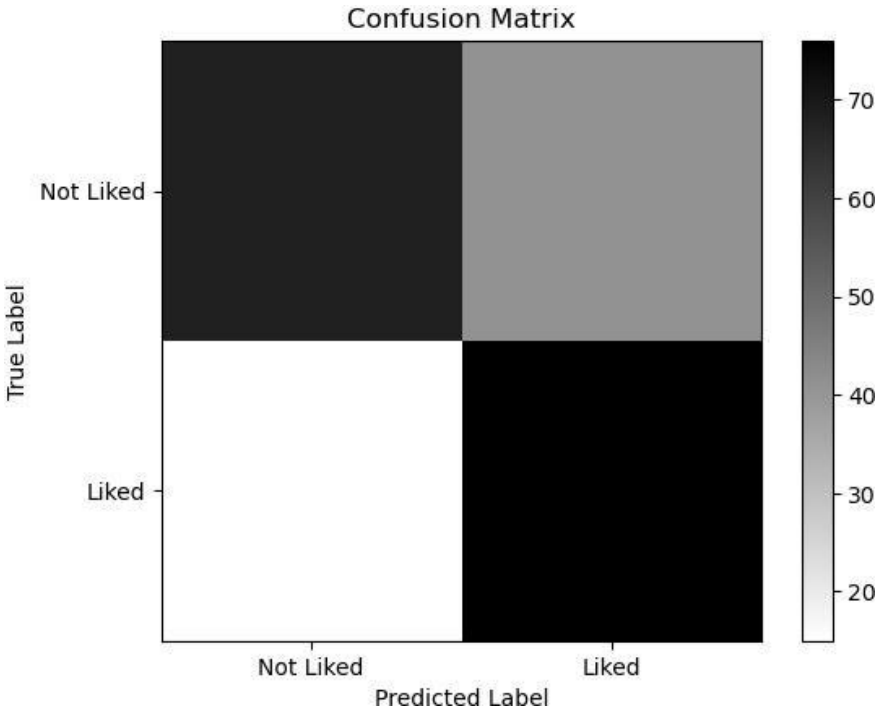
```
Accuracy: 0.74
```

In [78]:
```
y_pred
```

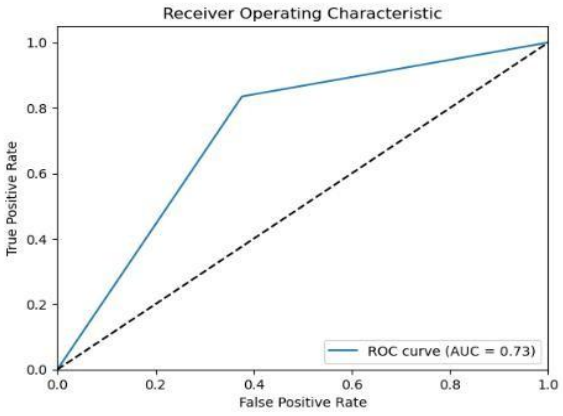
Out[78]:
```
array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 1])
```

Visualization：

- calculating and visualizing a confusion matrix for the binary classification problem using the confusion_matrix function from the metrics module of scikit-learn. The confusion matrix is a table that compares predicted labels to actual labels to assess how well a classification algorithm performed. The matrix's diagonal members reflect the number of examples that were successfully identified, whereas its off-diagonal elements represent the number of instances that were incorrectly classified.
- A binary classification model's effectiveness is graphically depicted by the ROC (Receiver Operating Characteristic) curve. Plotting the true positive rate (TPR) vs the false positive rate (FPR) at different threshold values results in it..



In [58]:
```
# plot ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



In [57]:
```
from sklearn.metrics import confusion_matrix,roc_curve,auc
# plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, cmap='binary', interpolation='None')
plt.colorbar()
plt.xticks([0,1], ['Not Liked', 'Liked'])
plt.yticks([0,1], ['Not Liked', 'Liked'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Word Cloud :

- The magnitude of each word in a word cloud, a graphical representation of text data, shows its frequency or significance. The words that appear the most frequently in a word cloud are displayed with larger font sizes, while the words that appear the least frequently are displayed with smaller font sizes. Word clouds are widely used in spotting patterns, trends, or insights that are not always obvious from a texts.The code generates a word cloud visualization from a collection of text reviews using the wordcloud library. The reviews_df variable is pandas DataFrame containing a column named 'Review' that contains the text of each review.
- The reviews are concatenated into a single string using the join function. The image's height and width, the color of the backdrop, and the smallest font size are then initialized with the WordCloud object. To create the word cloud, the concatenated text is passed into the generate method of the WordCloud object.
- The word cloud image is displayed using the matplotlib imshow function, and the axes function is run with the option "off" to remove the axes from the plot. The resulting image displays the terms that were used the most frequently in the reviews, with the magnitude of each word reflecting how often it was used.

```
. wordcloud
```

```
: already satisfied: wordcloud in c:\users\devro\anaconda3\lib\site-packages (1.9.1
: already satisfied: numpy>=1.6.1 in c:\users\devro\anaconda3\lib\site-packages (fr
: already satisfied: matplotlib in c:\users\devro\anaconda3\lib\site-packages (from
: already satisfied: pillow in c:\users\devro\anaconda3\lib\site-packages (from wor
: already satisfied: fonttools>=4.22.0 in c:\users\devro\anaconda3\lib\site-package

: already satisfied: cycler>=0.10 in c:\users\devro\anaconda3\lib\site-packages (fr
: already satisfied: kiwisolver>=1.0.1 in c:\users\devro\anaconda3\lib\site-package

: already satisfied: pyparsing>=2.2.1 in c:\users\devro\anaconda3\lib\site-packages

: already satisfied: packaging>=20.0 in c:\users\devro\anaconda3\lib\site-packages

: already satisfied: python-dateutil>=2.7 in c:\users\devro\anaconda3\lib\site-pack

: already satisfied: six>=1.5 in c:\users\devro\anaconda3\lib\site-packages (from p
1.16.0)
ay need to restart the kernel to use updated packages.
```

```
las as pd
.oud import WordCloud
lotlib.pyplot as plt

te all the reviews into a single string
  '.join(reviews_df['Review'])

word cloud
 WordCloud(width=800, height=800, background_color='white', min_font_size=10).gene

word cloud
figsize=(8,8))
wordcloud)
ff")
```

Model 3 : Random Forest :

A sort of ensemble learning technique called the Random Forest Classifier combines numerous decision trees to produce greater predicting performance than a single decision tree. This approach for supervised learning can be applied to both classification and regression tasks. By picking subsets of the data and features at random, the method builds a decision tree on each subset to produce a forest of them. Each decision tree in the prediction forest makes a forecast during the process, and the overall prediction is the average of all the predictions made by the trees. The data and feature randomization contributes to lowering over fitting and improving the generalization of the model.

In [61]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

class DecisionTree:
    def __init__(self, max_depth=10):
        self.max_depth = max_depth
        self.feature = None
        self.threshold = None
        self.left = None
        self.right = None
        self.label = None

    def fit(self, X, y):
        if len(set(y)) == 1:
            self.label = y[0]
        elif self.max_depth == 0:
            self.label = np.bincount(y).argmax()
        else:
            n_features = X.shape[1]
            best_gain = 0
            for f in range(n_features):
                feature_values = X[:, f]
                thresholds = np.unique(feature_values)
                for t in thresholds:
                    gain = self._information_gain(X, y, f, t)
                    if gain > best_gain:
                        best_gain = gain
                        self.feature = f
                        self.threshold = t
            if self.feature is not None:
                left_idxs = X[:, self.feature] <= self.threshold
                X_left, y_left = X[left_idxs], y[left_idxs]
                X_right, y_right = X[~left_idxs], y[~left_idxs]
                self.left = DecisionTree(max_depth=self.max_depth-1)
                self.right = DecisionTree(max_depth=self.max_depth-1)
                self.left.fit(X_left, y_left)
                self.right.fit(X_right, y_right)
            else:
                self.label = np.bincount(y).argmax()

    def predict(self, X):
        if self.label is not None:
            return np.full(X.shape[0], self.label)
        else:
            left_idxs = X[:, self.feature] <= self.threshold
            y = np.zeros(X.shape[0])
            y[left_idxs] = self.left.predict(X[left_idxs])
            y[~left_idxs] = self.right.predict(X[~left_idxs])
            return y

    def _entropy(self, y):
        _, counts = np.unique(y, return_counts=True)
        p = counts / len(y)
        return -np.sum(p * np.log2(p))

    def _information_gain(self, X, y, feature, threshold):
        left_idxs = X[:, feature] <= threshold
        H_parent = self._entropy(y)
        H_left = self._entropy(y[left_idxs])
        H_right = self._entropy(y[~left_idxs])
        n_left = len(y[left_idxs])
        n_right = len(y[~left_idxs])
        H_children = (n_left / len(y)) * H_left + (n_right / len(y)) * H_right
        return H_parent - H_children
```

```
class RandomForest:
    def __init__(self, n_estimators=100, max_depth=10, min_samples_split=5, max_features=None):
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.max_features = max_features
        self.trees = []

    def fit(self, X, y):
        n_samples = X.shape[0]
        n_features = X.shape[1]
        if self.max_features is None:
            self.max_features = int(np.sqrt(n_features))
        for i in range(self.n_estimators):
            tree = DecisionTree(max_depth=self.max_depth)
            idxs = np.random.choice(n_samples, n_samples, replace=True)
            X_sub = X[idxs]
            y_sub = y[idxs]
            tree.fit(X_sub, y_sub)
            self.trees.append(tree)

    def predict(self, X):
        y_preds = []
        for tree in self.trees:
            y_preds.append(tree.predict(X))
        y_preds = np.array(y_preds)
        y_pred = []
        for i in range(X.shape[0]):
            labels, counts = np.unique(y_preds[:, i], return_counts=True)
            y_pred.append(labels[np.argmax(counts)])
        return np.array(y_pred)

    def score(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)
```

In [62]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

class RestaurantReviewClassifier:
    def __init__(self, n_estimators=100, max_depth=10, min_samples_split=4, max_features=None):
        self.vectorizer = CountVectorizer(stop_words='english')
        self.rf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, min_samples_split=min_samples_split, ma

    def preprocess_data(self, filename):
        data = pd.read_csv(filename)
        data['Review'] = data['Review'].str.lower()
        X = self.vectorizer.fit_transform(data['Review'])
        y = data['Liked']
        return X, y

    def fit(self, X, y):
        self.rf.fit(X, y)

    def predict(self, X):
        return self.rf.predict(X)
```

- The constructor function of the DecisionTree class initializes the maximum tree depth (max_depth), the feature index (feature), the feature threshold (threshold), the left and right child nodes (left and right), and the label (label) for leaf nodes.The decision tree is trained using the input data (X) and labels (Y) using the fit method. The node becomes a leaf node with the same label as the input if all of the labels in y are identical. The node changes into a leaf node with the most popular label in y if the maximum depth is achieved. In the absence of it, the algorithm attempts to divide the data using the feature that offers the greatest information gain.
- It calculates the information gain, loops through each feature and each distinct value for that feature, and then notes the best feature and threshold that maximizes information gain. The data is split into left and right child nodes and the fit procedure is recursively run on each child node once the best feature and threshold have been identified.Using the trained decision tree, the predict method takes in new data X and provides the expected labels. It returns the label for that node if it is a leaf node. If not, the data is divided according to the feature and threshold, and the predict algorithm is repeatedly called on the left and right child nodes.
- The input labels' entropy is determined by the _entropy technique. The information gain of splitting the data on a specific feature and threshold is determined using the _information_gain method. The number of examples in each child node, n_left and n_right, as well as the entropies of the parent node H_parent, the left and right child nodes H_left and H_right, are all calculated. The information gain is represented as the difference between the entropies of the parent and child nodes, weighted by the number of examples in each child node.
- The fit function initializes and fits several instances of the DecisionTree class to create a Random Forest. Using the max_features hyperparameter, each tree is trained using a random subset of the training data. Predict produces an array of predicted class labels for each test sample after receiving a matrix of test data X. A majority vote of the anticipated labels for each tree in the forest is used to determine this. The scoring technique then determines the predictive accuracy using the supplied test data (X) and true labels (Y).
- The DecisionTree class, which is described in the preceding slide, is used in the implementation.
- Restaurant Review Classifier is a class that classifies restaurant reviews as positive or negative depending on whether the reviewer enjoyed the establishment or not using a random forest method. The class employs various techniques:The random forest classifier's hyperparameters are specified via the __init__function, which also initializes the class.
- preprocess_data: Reads data from a CSV file and preprocesses it by making all text lowercase and turning it into a word count matrix using a CountVectorizer.
- Fit: Adjusts the input data to the random forest classifier.
- utilizes the fitted random forest classifier to predict the labels for the input data.

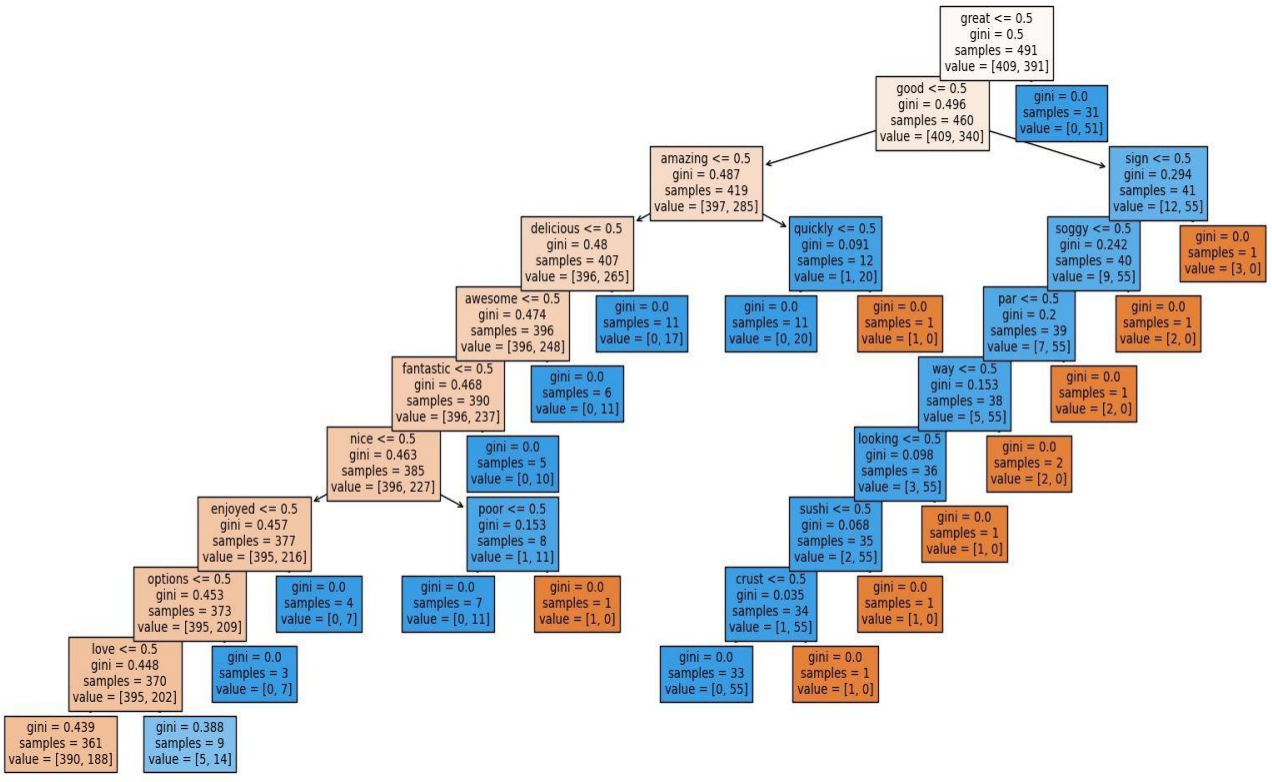
Results :

```
In [85]: clf = RestaurantReviewClassifier()
         X, y = clf.preprocess_data(filename)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         print('Accuracy:', (y_pred == y_test).mean())

         Accuracy: 0.685
```

```
In [64]: y_pred

Out[64]: array([0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
                1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
                1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
                0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
                0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
                0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
                0, 1], dtype=int64)
```

We could see that accuracy is 68%

Visualization :




```
In [65]: import matplotlib.pyplot as plt
         from sklearn.tree import plot_tree

         # assuming `clf` is your trained classifier
         plt.figure(figsize=(20,10))
         plot_tree(clf.rf.estimators_[0], feature_names=clf.vectorizer.get_feature_names(), filled=True)
         plt.show()
```

Accuracy Score for Three Models :

Because the restaurant review dataset is very small and straightforward and because the elements crucial for identifying whether a review is positive or negative are easily observable, all three models exhibit comparable performance. All three models are also reasonably potent machine learning algorithms that excel at a variety of tasks, including sentiment analysis.

	Model 1 - Navie Bayes	**Model 2 - Logistic Regression**	**Model 3 - Random Forest**
Accuracy Score	0.72	0.74	0.684

Applications :

Our models can be applied to any of the restaurant reviews to find out the positive and negative reviews and the most used words in the reviews, this can help the management to figure out their strong and weak areas and can use it for the development of the restaurant by enhancing the strong points and improving the weak point. We categorize the reviews and show the best category of the restaurant using the bag of words while processing the reviews, and show the best category the restaurant is reviewed for (like most of the reviews have written about hygiene or food, etc.)

Contribution:

Devi Vara Prasad Rongala:

Data Preprocessing

Logistic Regression

Decision Tree

Random Forest

NaiveBayesClassifier

Word Cloud