Artificial Intelligence
Machine Learning

Fall 2024
Parsa Darban 810100141
This file contains the report and results of the simulations conducted.

Abstract

In this project, we focus on classifying hotels based on the following features.

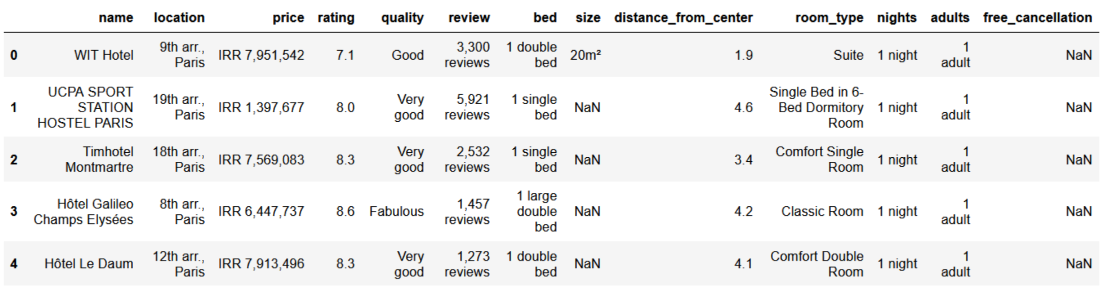
feature	explanation
name	Hotel name
location	Hotel Location (area and city)
price	Price of stay per night and for the specified number of people
rating	Overall hotel rating based on user feedback (a value between 0 and 10)
quality	Qualitative description of the hotel rating (e.g., "Fabulous", "Good", "Very Good")
review	Total number of reviews submitted by users about the hotel
bed	Type of bed available in the room (e.g., "single bed" or "double bed")
size	Room size (in square meters)
distance from center	Distance from the city center (in kilometers)
room type	Type of room reserved (e.g., "Suite", "Single Bed in 6-Bed Dormitory Room")
nights	Number of nights of stay
adults	Number of adults in the booking
free cancellation	Free cancellation availability

First, we need to prepare the data to train the models. Then, we classify them using algorithms and evaluate their performance.

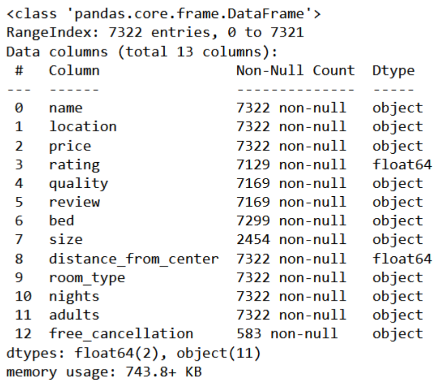
Data Preparation

In this section, we familiarize ourselves with the data and work on transforming it into a format suitable for training the model.

First, we examine the overall structure of the data.



	name	location	price	rating	quality	review	bed	size	distance_from_center	room_type	nights	adults	free_cancellation
0	WIT Hotel	9th arr., Paris	IRR 7,951,542	7.1	Good	3,300 reviews	1 double bed	20m²	1.9	Suite	1 night	1 adult	NaN
1	UCPA SPORT STATION HOSTEL PARIS	19th arr., Paris	IRR 1,397,677	8.0	Very good	5,921 reviews	1 single bed	NaN	4.6	Single Bed in 6-Bed Dormitory Room	1 night	1 adult	NaN
2	Timhotel Montmartre	18th arr., Paris	IRR 7,569,083	8.3	Very good	2,532 reviews	1 single bed	NaN	3.4	Comfort Single Room	1 night	1 adult	NaN
3	Hôtel Galileo Champs Elysées	8th arr., Paris	IRR 6,447,737	8.6	Fabulous	1,457 reviews	1 large double bed	NaN	4.2	Classic Room	1 night	1 adult	NaN
4	Hôtel Le Daum	12th arr., Paris	IRR 7,913,496	8.3	Very good	1,273 reviews	1 double bed	NaN	4.1	Comfort Double Room	1 night	1 adult	NaN



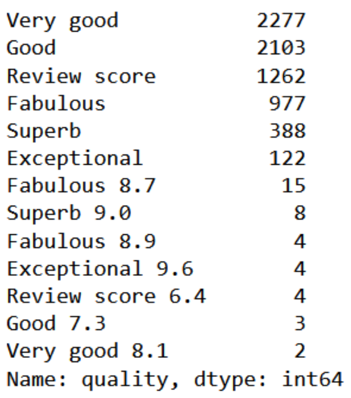
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7322 entries, 0 to 7321
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   name                  7322 non-null   object 
 1   location              7322 non-null   object 
 2   price                 7322 non-null   object 
 3   rating                7129 non-null   float64
 4   quality               7169 non-null   object 
 5   review                7169 non-null   object 
 6   bed                   7299 non-null   object 
 7   size                  2454 non-null   object 
 8   distance_from_center  7322 non-null   float64
 9   room_type             7322 non-null   object 
 10  nights                7322 non-null   object 
 11  adults                7322 non-null   object 
 12  free_cancellation     583 non-null    object 
dtypes: float64(2), object(11)
memory usage: 743.8+ KB
```

We observe that only two features in our dataset have numerical values. Therefore, one of the most important tasks is converting categorical data into numerical data.

Features like price, nights, adults, and review can be converted into numerical data without requiring special engineering.

Additionally, for the free_cancellation feature, if its value is NaN, it indicates the absence of this feature, and if it has the value free cancellation, it means the feature is present. Thus, we can convert this feature into numerical form using binary representation.

Now, let’s analyze the quality feature. Its values are as follows:



```
Very good           2277
Good                2103
Review score        1262
Fabulous             977
Superb               388
Exceptional          122
Fabulous 8.7          15
Superb 9.0             8
Fabulous 8.9           4
Exceptional 9.6        4
Review score 6.4       4
Good 7.3               3
Very good 8.1          2
Name: quality, dtype: int64
```

As we can see, not only does the quality feature have categorical values, but some of its values also include numerical characteristics. Upon further inspection, we find that the data with these conditions also has rating = NaN.

```
Count of rows where quality is in the list and rating is NaN: 40
            quality  rating
1220       Good 7.3     NaN
1636     Superb 9.0     NaN
1660     Superb 9.0     NaN
1679     Superb 9.0     NaN
1704     Superb 9.0     NaN
1893   Fabulous 8.9     NaN
1894   Fabulous 8.7     NaN
1897 Exceptional 9.6    NaN
1911   Fabulous 8.7     NaN
1914 Exceptional 9.6    NaN
1916   Fabulous 8.9     NaN
1931   Fabulous 8.7     NaN
1935 Exceptional 9.6    NaN
1938   Fabulous 8.9     NaN
1961   Fabulous 8.9     NaN
1962   Fabulous 8.7     NaN
```

Therefore, we can extract its numerical value and place it into the rating feature. After doing this, we notice that there are 153 entries where both quality and rating are NaN.

By examining the data, we find many duplicate entries with identical features. We keep only one of these duplicates and remove the rest. It is important to note that all their features were identical, which allowed us to safely remove the duplicates. This process reduced the number of entries with both quality and rating as NaN from 153 to 74. Additionally, the total dataset size decreased from 7322 to 2782 entries.

At this point, we can either fill the 74 missing values with the mean or median of the respective feature or simply remove them. Since the number of missing values is small, we choose to remove them.

We also remove small-size data as well as features like location and hotel name, as they do not significantly affect the model.

Another issue with the data is the Review score feature. Based on the rating, we map it to more distinguishable values. After analysis, we perform the mapping as follows:

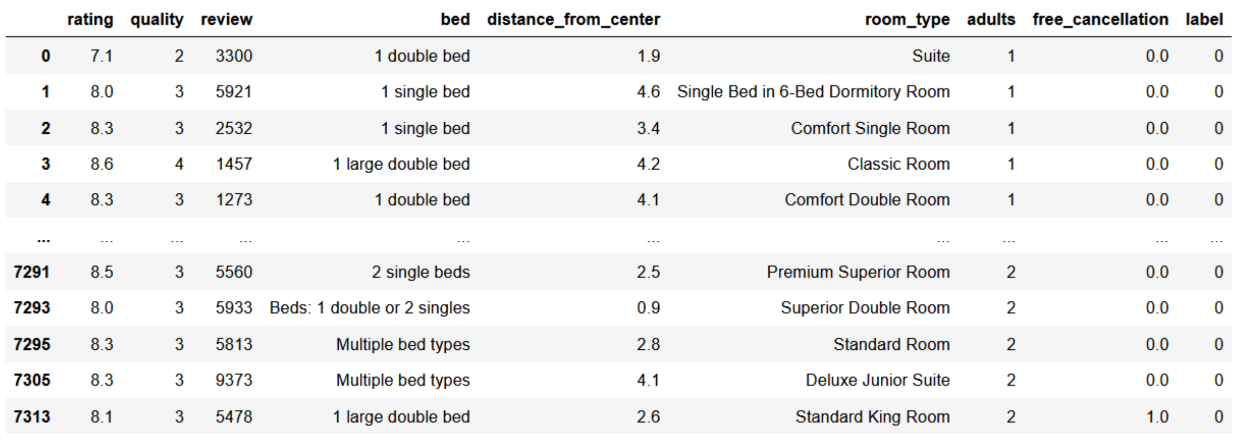
```
if rating < 7.0:
    return 'Bad'
elif 7.0 <= rating < 8.0:
    return 'Good'
elif 8.0 <= rating < 8.6:
    return 'Very good'
elif 8.6 <= rating < 9.0:
    return 'Fabulous'
elif 9.0 <= rating < 9.5:
    return 'Superb'
elif rating >= 9.5:
    return 'Exceptional'
else:
    return None
```

Now, we convert all features into numerical values.

The next step is to normalize the costs to a consistent scale. We divide the price by the number of nights to calculate the cost of the hotel per night. Then, we remove the nights and price features.

Additionally, we label the data based on the hotel price. If the price is greater than the median price, we assign a label of 1; otherwise, we assign a label of 0. Since price per night strongly influences the model due to its direct relation to the labels, we remove it as well.

Now, we arrive at the following dataset:



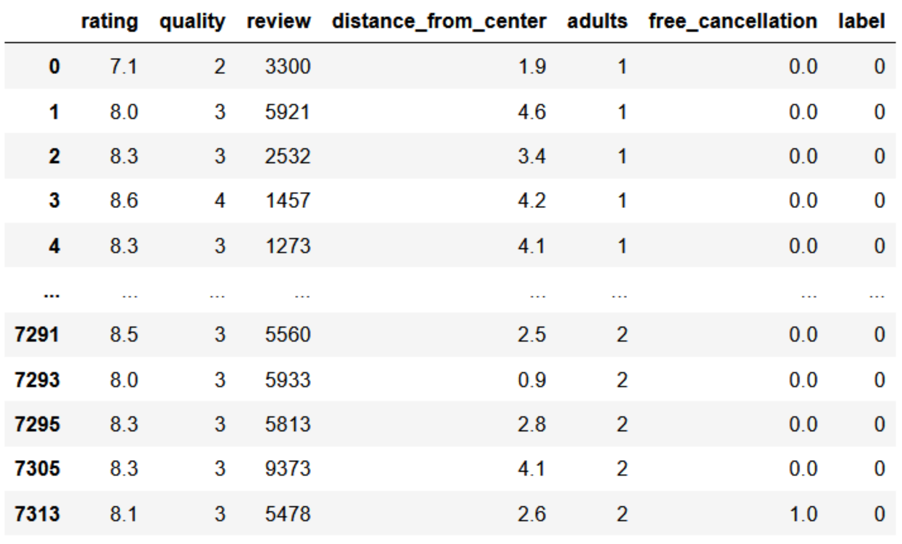
	rating	quality	review	bed	distance_from_center	room_type	adults	free_cancellation	label
0	7.1	2	3300	1 double bed	1.9	Suite	1	0.0	0
1	8.0	3	5921	1 single bed	4.6	Single Bed in 6-Bed Dormitory Room	1	0.0	0
2	8.3	3	2532	1 single bed	3.4	Comfort Single Room	1	0.0	0
3	8.6	4	1457	1 large double bed	4.2	Classic Room	1	0.0	0
4	8.3	3	1273	1 double bed	4.1	Comfort Double Room	1	0.0	0
...	...	...	...	...	...	...	...	...	...
7291	8.5	3	5560	2 single beds	2.5	Premium Superior Room	2	0.0	0
7293	8.0	3	5933	Beds: 1 double or 2 singles	0.9	Superior Double Room	2	0.0	0
7295	8.3	3	5813	Multiple bed types	2.8	Standard Room	2	0.0	0
7305	8.3	3	9373	Multiple bed types	4.1	Deluxe Junior Suite	2	0.0	0
7313	8.1	3	5478	1 large double bed	2.6	Standard King Room	2	1.0	0

We proceed as follows:

```
Number of rows with label 0: 1361
Number of rows with label 1: 1361
```

From this point onward, we take two actions.

First, we remove the room_type and bed features and evaluate the model.

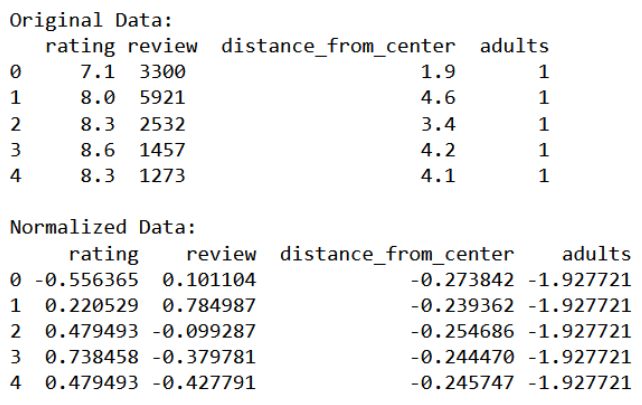


	rating	quality	review	distance_from_center	adults	free_cancellation	label
0	7.1	2	3300	1.9	1	0.0	0
1	8.0	3	5921	4.6	1	0.0	0
2	8.3	3	2532	3.4	1	0.0	0
3	8.6	4	1457	4.2	1	0.0	0
4	8.3	3	1273	4.1	1	0.0	0
...	...	...	...	...	...	...	...
7291	8.5	3	5560	2.5	2	0.0	0
7293	8.0	3	5933	0.9	2	0.0	0
7295	8.3	3	5813	2.8	2	0.0	0
7305	8.3	3	9373	4.1	2	0.0	0
7313	8.1	3	5478	2.6	2	1.0	0

Second, we found a way to represent the validity of the rating and the number of reviews by summing the ratings and multiplying each rating by the number of reviews. This result was then used for the model.

Normalization

The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.
In case of multivariate data, this is done feature-wise (in other words independently for each column of the data).
Given the distribution of the data, each value in the dataset will have the mean value subtracted, and then divided by the standard deviation of the whole dataset (or feature in the multivariate case).



```
Original Data:
   rating review  distance_from_center  adults
0     7.1  3300                    1.9       1
1     8.0  5921                    4.6       1
2     8.3  2532                    3.4       1
3     8.6  1457                    4.2       1
4     8.3  1273                    4.1       1

Normalized Data:
     rating    review  distance_from_center    adults
0 -0.556365  0.101104             -0.273842 -1.927721
1  0.220529  0.784987             -0.239362 -1.927721
2  0.479493 -0.099287             -0.254686 -1.927721
3  0.738458 -0.379781             -0.244470 -1.927721
4  0.479493 -0.427791             -0.245747 -1.927721
```

Train-Test Split

Using train_test_split from scikit-learn, we divide the data into 80% for training and 20% for testing.

Evaluation

The confusion matrix is a table used to evaluate the performance of a classification model by comparing its predicted labels with the actual labels. It consists of four key components: True Positives (TP), where the model correctly predicts the positive class; True Negatives (TN), where the model correctly predicts the negative class; False Positives (FP), where the model incorrectly predicts the positive class; and False Negatives (FN), where the model incorrectly predicts the negative class. The confusion matrix provides insights into the types of errors the model is making, which is critical for calculating metrics such as accuracy, precision, recall, and F1 score, helping to assess how well the model is performing, especially in cases of class imbalance.

Precision is a metric that measures the accuracy of positive predictions made by a classification model. It is calculated as the ratio of true positive predictions to the total number of positive predictions (both true positives and false positives). Precision answers the question: *"Of all the instances the model predicted as positive, how many were actually positive?"* A high precision means that the model is good at avoiding false positives, but it doesn't necessarily mean it is good at identifying all positive instances.

Recall, also known as sensitivity or true positive rate, measures the ability of a classification model to identify all relevant positive instances in a dataset. It is calculated as the ratio of true positive predictions to the total number of actual positive instances (true positives and false negatives). Recall answers the question: *"Of all the actual positive instances, how many did the model correctly identify?"* A high recall indicates that the model is good at capturing most of the positive cases, but it might also produce some false positives.

F1 Score is the harmonic mean of precision and recall, providing a single metric that balances both. It is especially useful when the class distribution is imbalanced, as it gives a better measure of the model's performance than accuracy alone. The F1 score ranges from 0 to 1, where a score of 1 indicates perfect precision and recall, and 0 indicates the worst performance. It is a good choice when both false positives and false negatives are costly, as it takes both into account equally.

Accuracy is the most straightforward metric, representing the proportion of correct predictions (both true positives and true negatives) to the total number of predictions. It is calculated as the ratio of correct predictions to the total number of predictions. Accuracy is useful when the class distribution is balanced, but it can be misleading in cases of imbalanced datasets, as it might not reflect the model's ability to identify the minority class effectively. For example, if a dataset contains 95% negative samples and 5% positive, a model that always predicts negative will still have 95% accuracy, despite performing poorly on the minority class.

Micro, macro, and weighted averages are methods used to aggregate performance metrics in multi-class classification. Micro average calculates metrics globally by considering the total number of true positives, false positives, and false negatives across all classes, treating all instances equally, which is useful for imbalanced datasets. Macro average computes metrics for each class individually and then averages them, giving equal weight to each class regardless of size, making it suitable when all classes should be treated equally. Weighted average also calculates metrics for each class but averages them using the proportion of each class in the dataset as the weight, accounting for class imbalances by giving larger classes more influence in the final result.

Models

Given that the data is randomly split into training and test datasets, this process was performed once and the results were stored. Due to this randomness, the accuracy may vary. In this report, the maximum accuracy is calculated, but the results in the code may differ slightly.

Naïve Bayesian

Naive Bayes is a classification technique based on Bayes' Theorem, with the assumption that all the features predicting the target value are independent of each other. It calculates the probability of each class and then selects the one with the highest probability.

As we know, the Bayesian classifier can be expressed as follows if our features are $\{x_1, x_2, ..., x_n\}$ and Y is the label:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

The conditional probability can be written as:

$$P(X|Y) = P(x_1|x_2, x_3, \dots, x_n, y) \times P(x_2|x_3, \dots, x_n, y) \times \dots \times P(x_n|y)$$

However, since this algorithm assumes that the features are independent of each other, the above conditional probability simplifies to:

$$P(X|Y) = P(x_1|y) \times P(x_2|y) \times \dots \times P(x_n|y)$$

Thus, we know that the following relation holds:

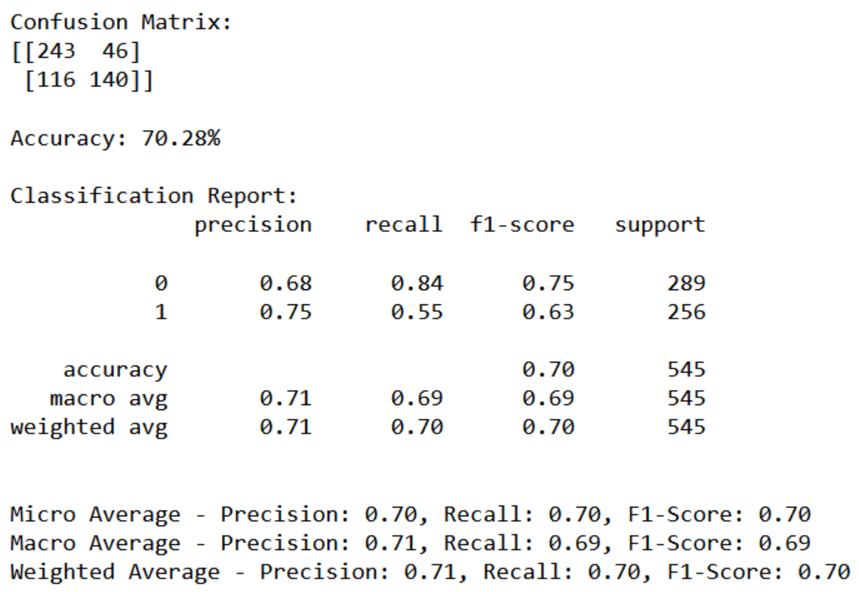
$$P(Y|X) \propto P(Y) \prod_{i=1}^{n} P(x_i|y)$$

A common rule is to select the hypothesis with the highest probability, which is known as the Maximum a Posteriori (MAP) decision rule. Therefore, we have:

$$Y = \underset{y}{\operatorname{argmax}} P(Y) \prod_{i=1}^{n} P(x_i|y)$$

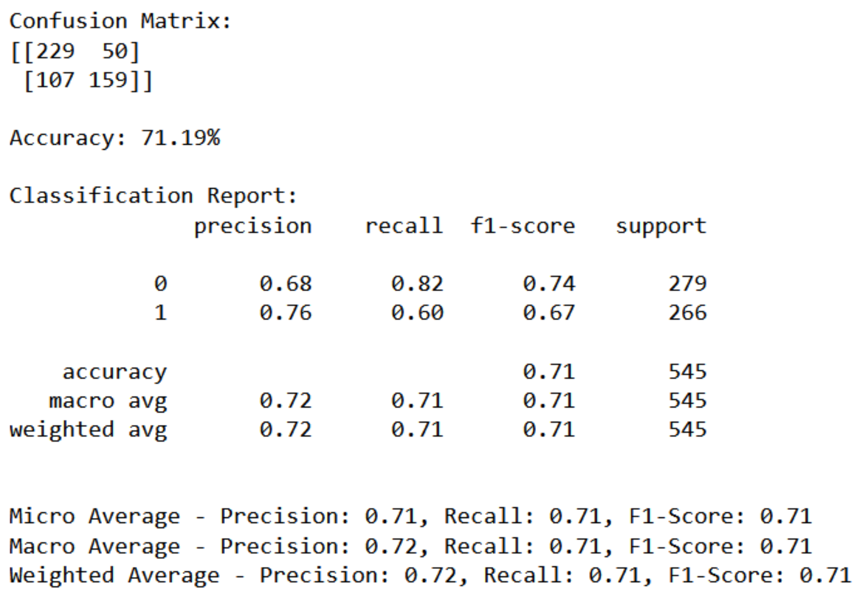
Hence, the algorithm calculates the probability for each class and selects the one with the maximum value.

The outcome of this from the initial feature extraction attempt is:



```
Confusion Matrix:
[[243  46]
 [116 140]]

Accuracy: 70.28%

Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.84      0.75       289
           1       0.75      0.55      0.63       256

    accuracy                           0.70       545
   macro avg       0.71      0.69      0.69       545
weighted avg       0.71      0.70      0.70       545


Micro Average - Precision: 0.70, Recall: 0.70, F1-Score: 0.70
Macro Average - Precision: 0.71, Recall: 0.69, F1-Score: 0.69
Weighted Average - Precision: 0.71, Recall: 0.70, F1-Score: 0.70
```

The outcome of this from the second feature extraction attempt is:



```
Confusion Matrix:
[[229  50]
 [107 159]]

Accuracy: 71.19%

Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.82      0.74       279
           1       0.76      0.60      0.67       266

    accuracy                           0.71       545
   macro avg       0.72      0.71      0.71       545
weighted avg       0.72      0.71      0.71       545


Micro Average - Precision: 0.71, Recall: 0.71, F1-Score: 0.71
Macro Average - Precision: 0.72, Recall: 0.71, F1-Score: 0.71
Weighted Average - Precision: 0.72, Recall: 0.71, F1-Score: 0.71
```

Decision Tree

It is a supervised machine learning algorithm, used for both classification and regression task. It is a model that uses set of rules to classify something.

A decision tree algorithm is used to split the features of a dataset through a cost function. Before optimization and pruning of unnecessary branches, this algorithm grows in such a way that it includes features unrelated to the problem. Therefore, a pruning operation is performed to remove these unnecessary branches. In a decision tree algorithm, parameters such as the depth of the tree can be adjusted to prevent overfitting or overly complex trees as much as possible.

In a decision tree, to predict the classes of the dataset's problem, the algorithm starts from the root node of the tree. It compares the root feature values with the data features and, based on this comparison, follows the branches and moves to the next node. One of its most important tasks is Attribute Selection Measure (ASM). There are two methods for it, but they all provide the purity of the feature selection result, though their mathematics are different.

Entropy is one of the ASM (Attribute Selection Measure) that measure the randomness of the information being processed. The lower it is, the purer the feature is. After calculating the entropy, it is used in the calculation of information gain (IG). The lower the entropy, the higher the information gain, the better the feature, and ultimately a more optimized tree is formed.

$$Entropy = -\sum_i p_i \log(p_i)$$

$$IG(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} Entropy(S_v)$$

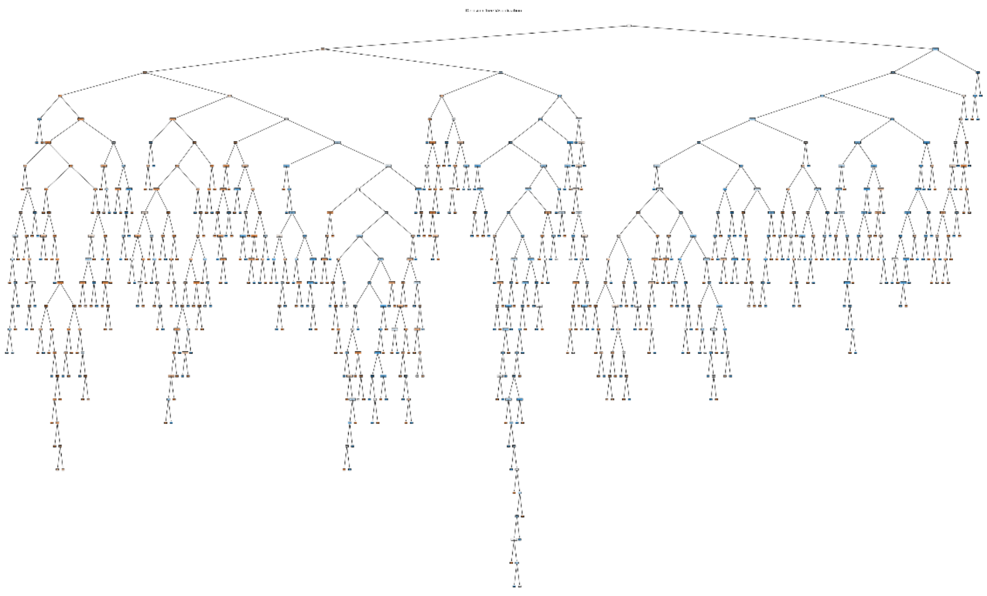
Values(A) is the all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

Another criterion is Gini index. It is similar to Entropy but faster and more efficient, making it preferred in large datasets or real-time systems.

$$Gini = 1 - \sum_i P_i^2$$

Optimization in this algorithm means minimizing Gini and entropy (maximizing information gain). Additionally, the decision tree itself has different algorithms such as CART, ID3, etc. Since Scikit-learn uses CART, we will explain it and its related parameters.

CART is a binary algorithm, meaning it creates two child nodes from each parent node. Even if the features have different values, it divides them by setting a threshold. It also uses Gini by default and improves the tree recursively at each stage. Additionally, it has the ability to prune the tree when branches no longer affect the model's accuracy. On the other hand, due to having specific stopping criteria for the algorithm, such as reaching the maximum depth, it is considered better than ID3. It is also evident in the diagram that each node has only two children.



criterion: A criterion for splitting nodes ('Gini' or 'entropy').
max_depth: The maximum depth of the decision tree (the number of node splits).
min_samples_split: The minimum number of samples required to split a node into two subgroups.
min_samples_leaf: The minimum number of samples required in each leaf node of the tree.

The outcome of this from the initial feature extraction attempt is:

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2}

Confusion Matrix:
[[232  34]
 [ 35 244]]

Accuracy: 87.34%

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       266
           1       0.88      0.87      0.88       279

    accuracy                           0.87       545
   macro avg       0.87      0.87      0.87       545
weighted avg       0.87      0.87      0.87       545


Micro Average - Precision: 0.87, Recall: 0.87, F1-Score: 0.87
Macro Average - Precision: 0.87, Recall: 0.87, F1-Score: 0.87
Weighted Average - Precision: 0.87, Recall: 0.87, F1-Score: 0.87
```

The outcome of this from the second feature extraction attempt is:

```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

Confusion Matrix:
[[237  35]
 [ 17 254]]

Accuracy: 90.42%

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       272
           1       0.88      0.94      0.91       271

    accuracy                           0.90       543
   macro avg       0.91      0.90      0.90       543
weighted avg       0.91      0.90      0.90       543


Micro Average - Precision: 0.90, Recall: 0.90, F1-Score: 0.90
Macro Average - Precision: 0.91, Recall: 0.90, F1-Score: 0.90
Weighted Average - Precision: 0.91, Recall: 0.90, F1-Score: 0.90
```

Random Forest

One of the issues with decision trees is their sensitivity to features. To address this problem, we use an algorithm called Random Forest. This algorithm employs two random processes. One of them is Bootstrapping, which means creating a new dataset of the same size as the original dataset by sampling with replacement. For each of these datasets, features are randomly selected, and a decision tree is built for each.

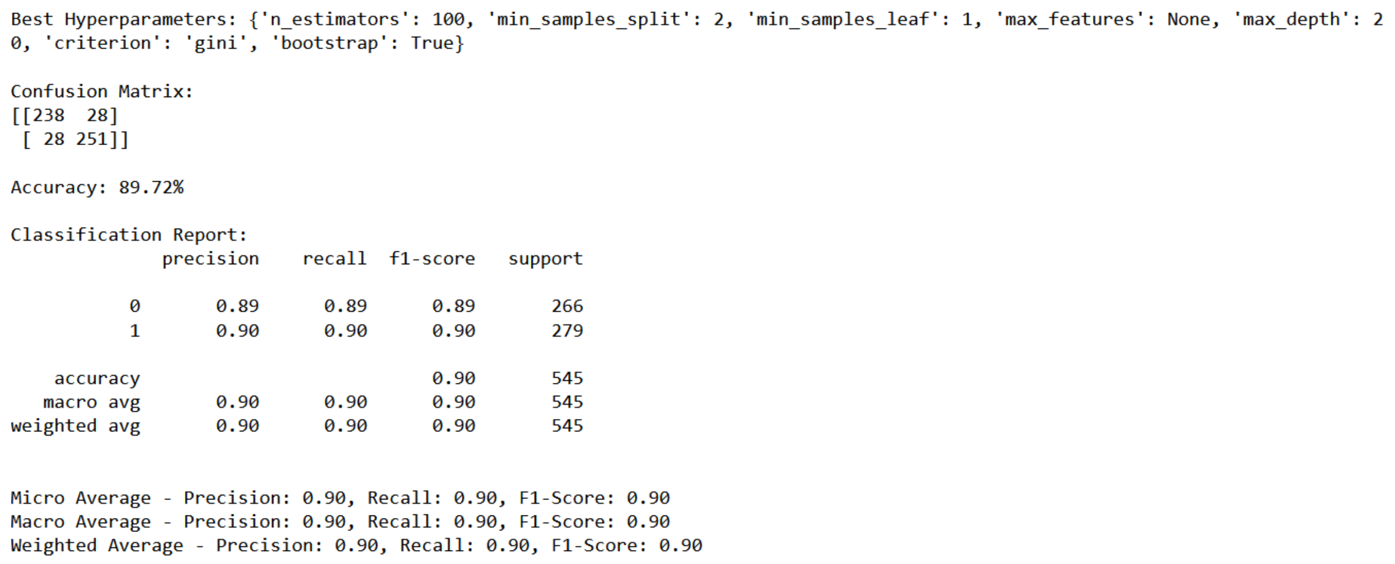
At this stage, we have a large number of trees based on random data from the original dataset. When new data is input into the algorithm, it is passed through each tree, and each tree predicts a specific class. Finally, a majority vote is taken. This entire process is known as Bagging.

Thus, we have:

$$Bootstrap + Aggregating = Bagging$$

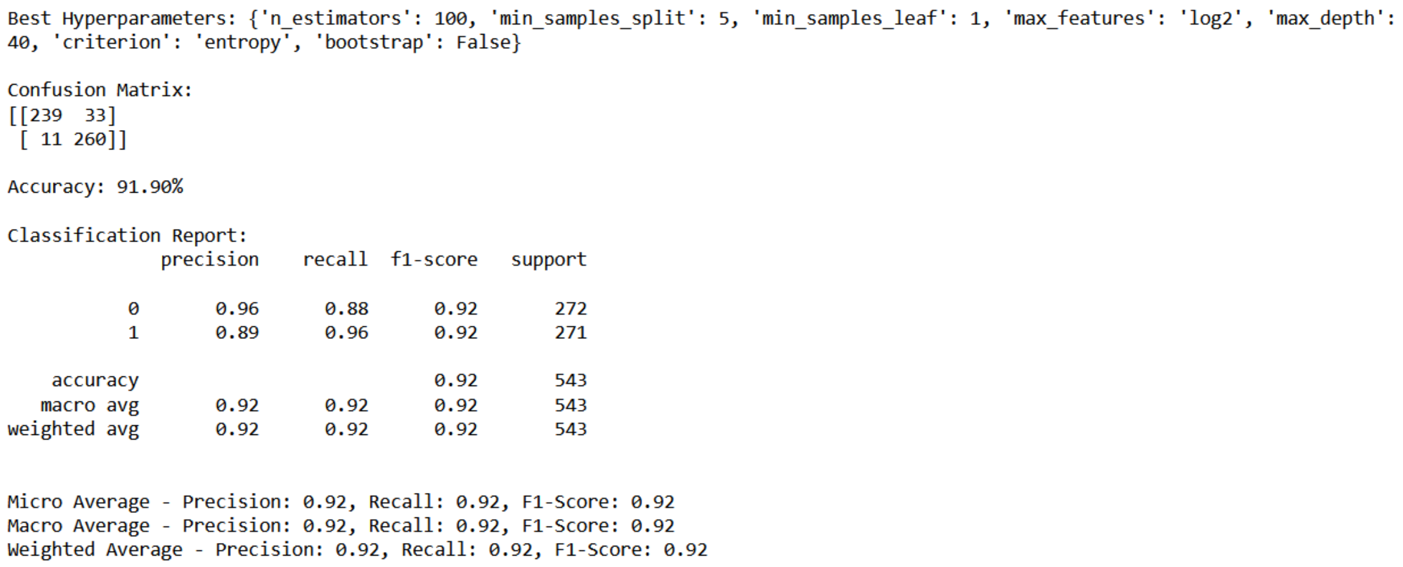
This algorithm is less sensitive to data and also reduces the variance of models.

The outcome of this from the initial feature extraction attempt is:



```
Best Hyperparameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': None, 'max_depth': 2
0, 'criterion': 'gini', 'bootstrap': True}

Confusion Matrix:
[[238  28]
 [ 28 251]]

Accuracy: 89.72%

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.89      0.89       266
           1       0.90      0.90      0.90       279

    accuracy                           0.90       545
   macro avg       0.90      0.90      0.90       545
weighted avg       0.90      0.90      0.90       545


Micro Average - Precision: 0.90, Recall: 0.90, F1-Score: 0.90
Macro Average - Precision: 0.90, Recall: 0.90, F1-Score: 0.90
Weighted Average - Precision: 0.90, Recall: 0.90, F1-Score: 0.90
```

The outcome of this from the second feature extraction attempt is:



```
Best Hyperparameters: {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth':
40, 'criterion': 'entropy', 'bootstrap': False}

Confusion Matrix:
[[239  33]
 [ 11 260]]

Accuracy: 91.90%

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.88      0.92       272
           1       0.89      0.96      0.92       271

    accuracy                           0.92       543
   macro avg       0.92      0.92      0.92       543
weighted avg       0.92      0.92      0.92       543


Micro Average - Precision: 0.92, Recall: 0.92, F1-Score: 0.92
Macro Average - Precision: 0.92, Recall: 0.92, F1-Score: 0.92
Weighted Average - Precision: 0.92, Recall: 0.92, F1-Score: 0.92
```

Adaptive Boosting

Boosting algorithms, unlike the previous Bagging algorithm, assign weights to the original dataset at each stage to improve the model and ultimately make predictions. The models used in boosting are referred to as Decision Stumps, which consist of only one node and one leaf.

Initially, the algorithm calculates the Gini index with equal weights and selects the best stump. Then, it computes the error using the following formula and updates the weights of both the classifier and the features.

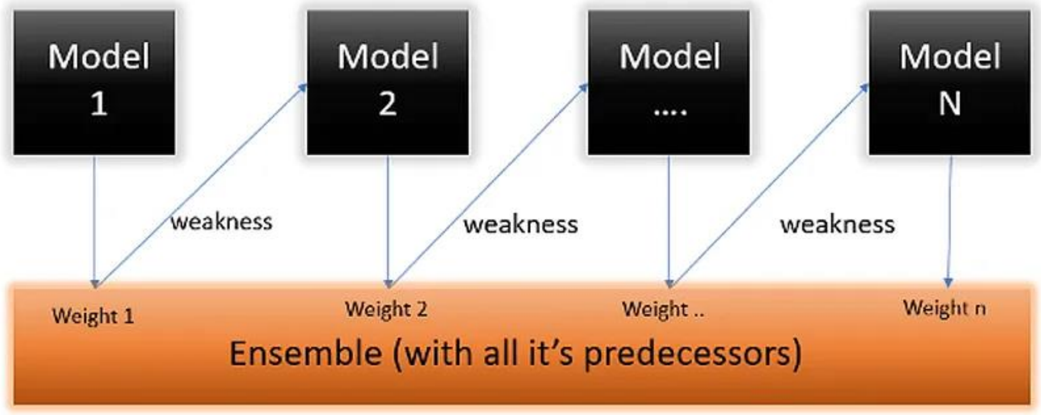
$$\epsilon_t = \frac{\sum_{i=1}^{n} w_i^{(t)} . I[h_t(x_i) \neq y_i]}{\sum_{i=1}^{n} w_i^{(t)}}$$

$$\alpha_t = \eta \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$

$$w_i^{(t+1)} = w_i^{(t)} . \exp\left(\pm \alpha_t . y_i . h_{(t)}(x_i)\right)$$

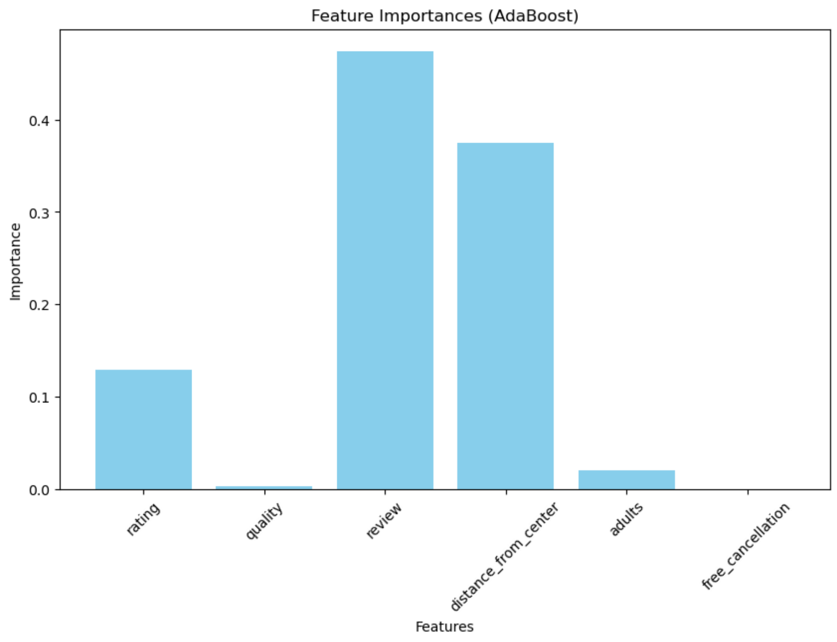
η is learning rate.

The weights are adjusted as follows: positive for increasing the weight of misclassified data and negative for correctly classified data.



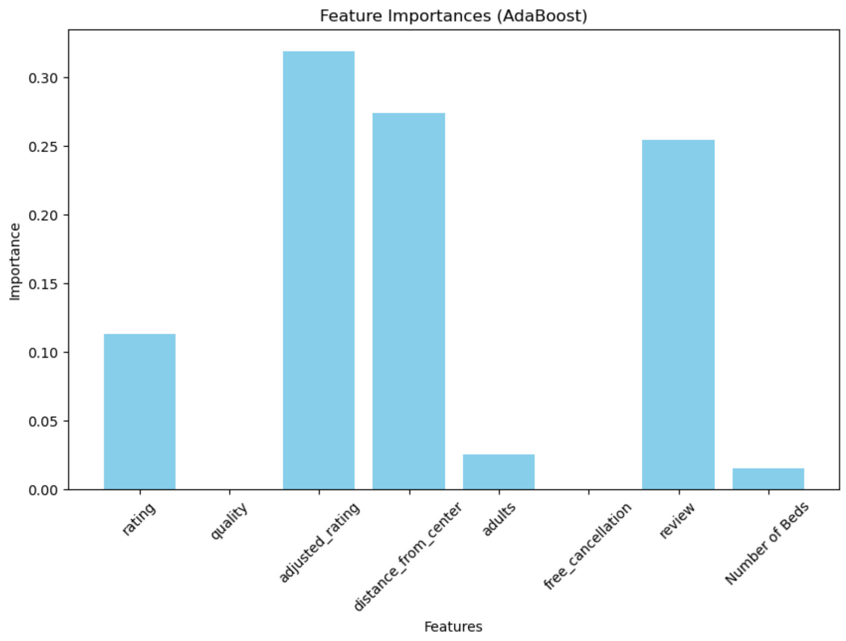
The outcome of this from the initial feature extraction attempt is:

```
Best Hyperparameters: {'n_estimators': 350, 'learning_rate': 1.0}

Confusion Matrix:
[[217  49]
 [ 64 215]]

Accuracy: 79.27%

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.82      0.79       266
           1       0.81      0.77      0.79       279

    accuracy                           0.79       545
   macro avg       0.79      0.79      0.79       545
weighted avg       0.79      0.79      0.79       545


Micro Average - Precision: 0.79, Recall: 0.79, F1-Score: 0.79
Macro Average - Precision: 0.79, Recall: 0.79, F1-Score: 0.79
Weighted Average - Precision: 0.79, Recall: 0.79, F1-Score: 0.79
```



The outcome of this from the second feature extraction attempt is:

```
Best Hyperparameters: {'n_estimators': 1000, 'learning_rate': 0.5}

Confusion Matrix:
[[211  61]
 [ 43 228]]

Accuracy: 80.85%

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.78      0.80       272
           1       0.79      0.84      0.81       271

    accuracy                           0.81       543
   macro avg       0.81      0.81      0.81       543
weighted avg       0.81      0.81      0.81       543


Micro Average - Precision: 0.81, Recall: 0.81, F1-Score: 0.81
Macro Average - Precision: 0.81, Recall: 0.81, F1-Score: 0.81
Weighted Average - Precision: 0.81, Recall: 0.81, F1-Score: 0.81
```



XGBoost

XGBoost (extreme gradient boosting) is an advanced implementation of the gradient boosting algorithm. It is designed for speed and performance and is widely used for structured/tabular data in machine learning competitions and real-world applications.

XGBoost is a powerful gradient boosting framework designed for speed and efficiency, building an ensemble of weak learners, typically decision trees, sequentially to minimize errors. It incorporates advanced engineering features like parallelized tree building and efficient handling of missing data, ensuring faster training. Regularization techniques, including L1L_1 (Lasso) and L2L_2 (Ridge), help reduce overfitting, while support for both predefined and customizable loss functions add flexibility for various machine learning tasks. These features make XGBoost a robust choice for optimized performance in predictive modeling.

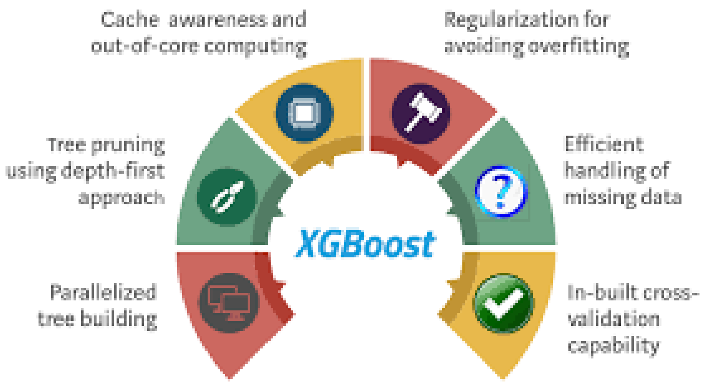
The algorithm minimizes the following objective function:

$$\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{T} \Omega(f_k)$$

L represents the loss function, Ω denotes the regularization term, and f corresponds to each decision tree added to the ensemble.

The predicted values $\hat{y}_i$ are updated as:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} + \eta . f_t(x_i)$$


Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

learning_rate: The rate of change for weights at each step.

n_estimators: The number of base models.

min_samples_split: The minimum number of samples required to split a node.

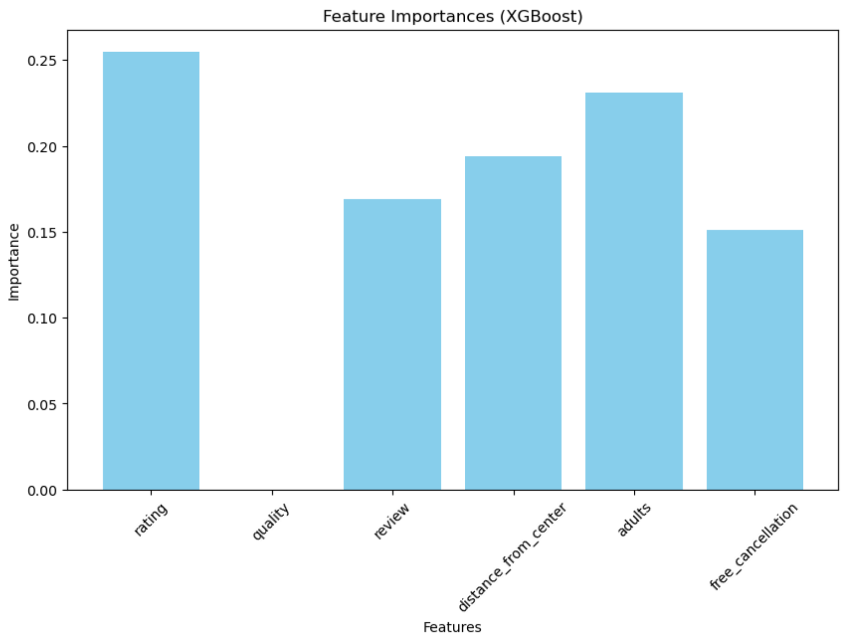
min_samples_leaf: The minimum number of samples required for each leaf node.

max_depth: The maximum depth of the decision trees.

max_features: The maximum number of features considered for splitting a node.

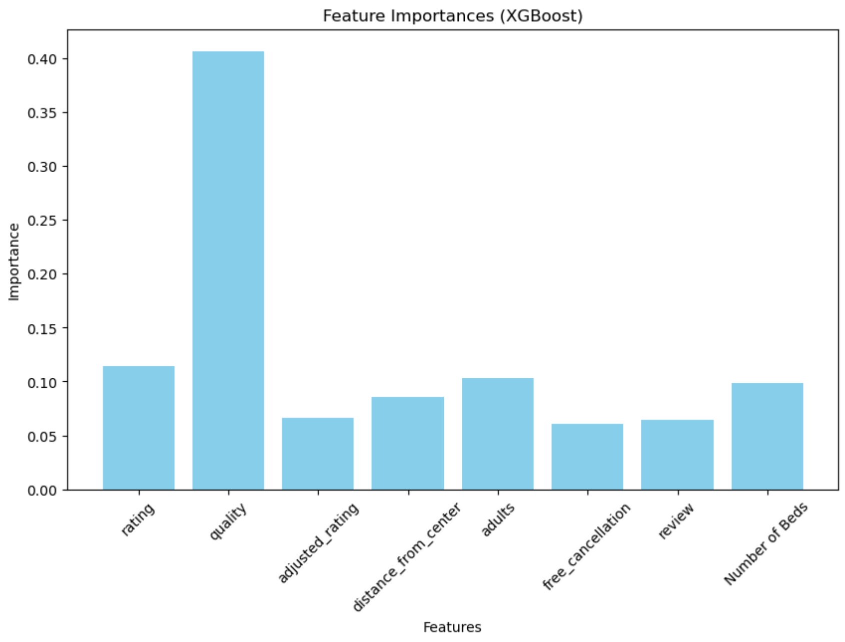
The outcome of this from the initial feature extraction attempt is: (max Accuracy = 89.72)

```
Best Hyperparameters: {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 200, 'subsample': 0.8}

Confusion Matrix:
[[230  36]
 [ 25 254]]

Accuracy: 88.81%

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.86      0.88       266
           1       0.88      0.91      0.89       279

    accuracy                           0.89       545
   macro avg       0.89      0.89      0.89       545
weighted avg       0.89      0.89      0.89       545


Micro Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
Macro Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
Weighted Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
```



The outcome of this from the second feature extraction attempt is:

```
Best Hyperparameters: {'colsample_bytree': 0.8, 'learning_rate': 0.5, 'max_depth': 3, 'n_estimators': 400, 'subsample': 1.0}

Confusion Matrix:
[[238  34]
 [ 16 255]]

Accuracy: 90.79%

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.88      0.90       272
           1       0.88      0.94      0.91       271

    accuracy                           0.91       543
   macro avg       0.91      0.91      0.91       543
weighted avg       0.91      0.91      0.91       543


Micro Average - Precision: 0.91, Recall: 0.91, F1-Score: 0.91
Macro Average - Precision: 0.91, Recall: 0.91, F1-Score: 0.91
Weighted Average - Precision: 0.91, Recall: 0.91, F1-Score: 0.91
```


Feature Importances (XGBoost)
Importance
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
rating
quality
adjusted_rating
distance_from_center
adults
free_cancellation
review
Number of Beds
Features

SAMME

Boosting begins by assigning equal weights to all training examples. In each iteration, a weak learner (e.g., decision stump) is trained using the current weights, and its error is calculated. The learner's contribution to the ensemble is determined based on its error, with more accurate learners contributing more. The weights of misclassified examples are increased to emphasize their importance in the next iteration, while the weights of correctly classified examples are reduced. Each learner's predictions are combined using their calculated ensemble weights to form the final prediction. This process is repeated for a predefined number of iterations or until a stopping criterion is met, and the final output is based on the weighted majority vote of the ensemble.

The primary difference between AdaBoost and SAMME lies in their weight computation for weak learners, especially in the context of multi-class classification. In AdaBoost, the weight of a weak learner (α_t) computed as mentioned. In SAMME, the weight is adjusted for multi-class classification by adding a term that accounts for the number of classes K, given by $\alpha_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) + \ln(K-1)$. This additional term allows SAMME to handle multiple classes directly, whereas AdaBoost relies on binary classifiers or other strategies to extend to multi-class problems.

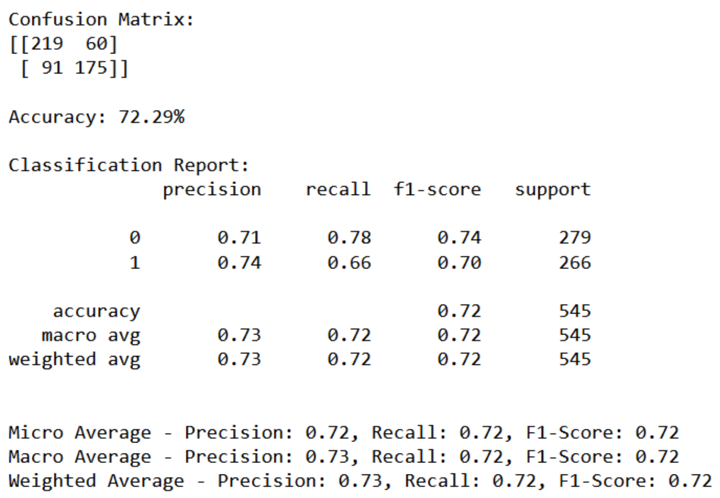
The outcome of this from the initial feature extraction attempt is: (Custom SAMME Implementation)

```
Confusion Matrix:
[[207  63]
 [104 171]]

Accuracy: 69.36%

Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.77      0.71       270
           1       0.73      0.62      0.67       275

    accuracy                           0.69       545
   macro avg       0.70      0.69      0.69       545
weighted avg       0.70      0.69      0.69       545


Micro Average - Precision: 0.69, Recall: 0.69, F1-Score: 0.69
Macro Average - Precision: 0.70, Recall: 0.69, F1-Score: 0.69
Weighted Average - Precision: 0.70, Recall: 0.69, F1-Score: 0.69
```

The outcome of this from the initial feature extraction attempt is: (Library SAMME Implementation)

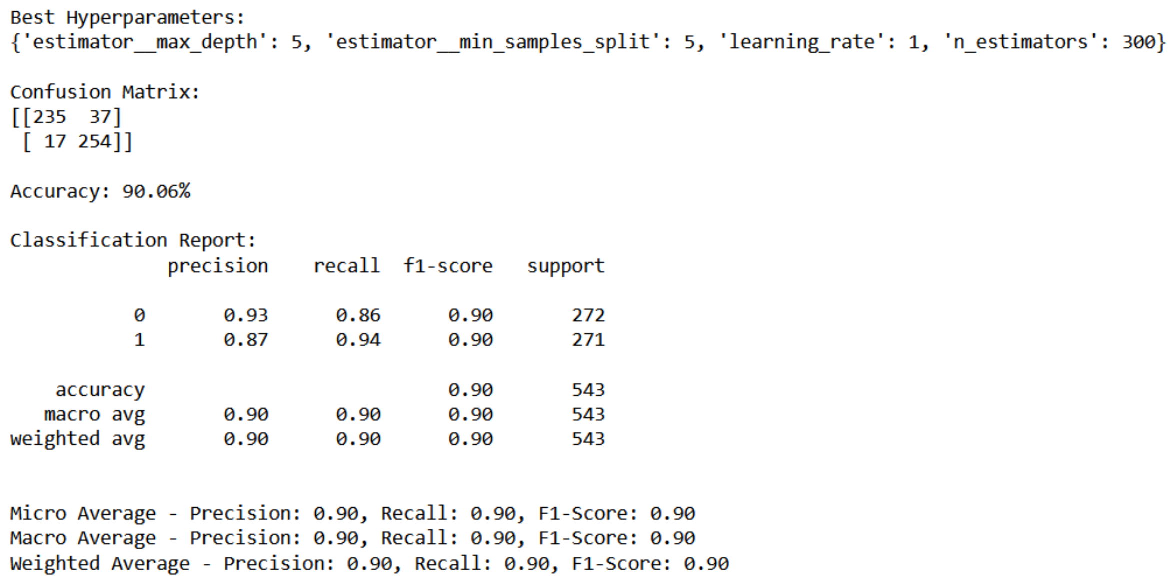
```
Best Hyperparameters:
{'estimator__max_depth': 5, 'estimator__min_samples_split': 2, 'learning_rate': 1, 'n_estimators': 100}

Confusion Matrix:
[[240  26]
 [ 33 246]]

Accuracy: 89.17%

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.90      0.89       266
           1       0.90      0.88      0.89       279

    accuracy                           0.89       545
   macro avg       0.89      0.89      0.89       545
weighted avg       0.89      0.89      0.89       545


Micro Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
Macro Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
Weighted Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
```

The outcome of this from the second feature extraction attempt is: (Custom SAMME Implementation)



```
Confusion Matrix:
[[219  60]
 [ 91 175]]

Accuracy: 72.29%

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.78      0.74       279
           1       0.74      0.66      0.70       266

    accuracy                           0.72       545
   macro avg       0.73      0.72      0.72       545
weighted avg       0.73      0.72      0.72       545


Micro Average - Precision: 0.72, Recall: 0.72, F1-Score: 0.72
Macro Average - Precision: 0.73, Recall: 0.72, F1-Score: 0.72
Weighted Average - Precision: 0.73, Recall: 0.72, F1-Score: 0.72
```

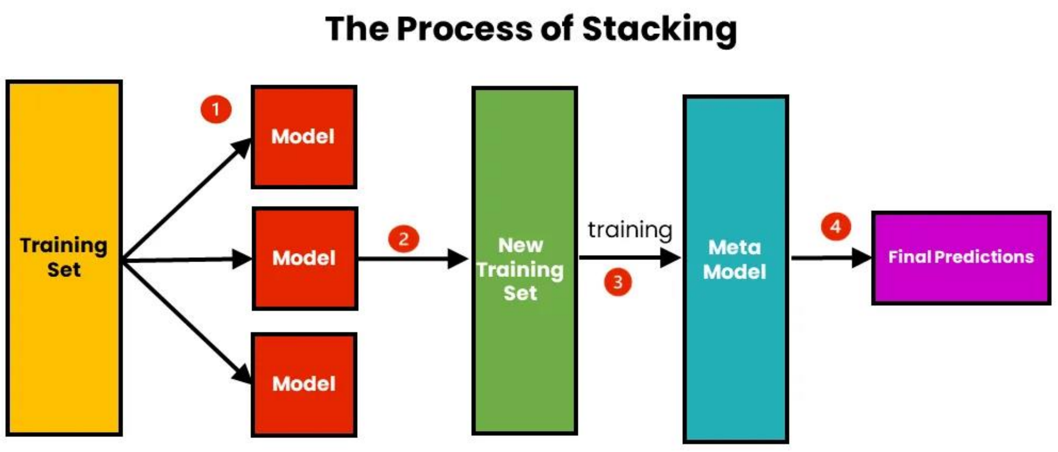
The outcome of this from the second feature extraction attempt is: (Library SAMME Implementation)



```
Best Hyperparameters:
{'estimator__max_depth': 5, 'estimator__min_samples_split': 5, 'learning_rate': 1, 'n_estimators': 300}

Confusion Matrix:
[[235  37]
 [ 17 254]]

Accuracy: 90.06%

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.86      0.90       272
           1       0.87      0.94      0.90       271

    accuracy                           0.90       543
   macro avg       0.90      0.90      0.90       543
weighted avg       0.90      0.90      0.90       543


Micro Average - Precision: 0.90, Recall: 0.90, F1-Score: 0.90
Macro Average - Precision: 0.90, Recall: 0.90, F1-Score: 0.90
Weighted Average - Precision: 0.90, Recall: 0.90, F1-Score: 0.90
```

Stacking

The process of building a stacking ensemble involves several key steps: preparing the data by identifying relevant features, cleaning, and splitting it into training and validation sets; selecting diverse base models to ensure complementary errors; training these models on the training set with different algorithms or hyperparameters; using the trained models to make predictions on the validation set; developing and training a meta-model, which takes the base models' predictions as input and makes the final prediction; making predictions on the test set using the meta-model; and finally evaluating the performance of the ensemble using metrics like accuracy, precision, recall, and F1 score to assess its effectiveness.

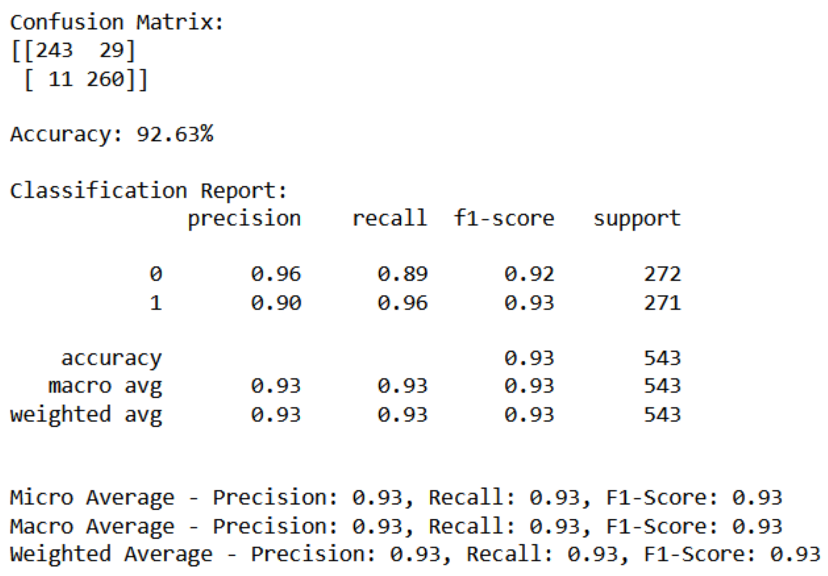


In the end, the goal of stacking is to combine the strengths of various base models by feeding them into a meta-model, which learns how to weigh and combine their forecasts to generate the final prediction. This can frequently result in higher performance than utilizing a single model alone.

The outcome of this from the initial feature extraction attempt is:

```
Confusion Matrix:
[[234  32]
 [ 29 250]]

Accuracy: 88.81%

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       266
           1       0.89      0.90      0.89       279

    accuracy                           0.89       545
   macro avg       0.89      0.89      0.89       545
weighted avg       0.89      0.89      0.89       545


Micro Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
Macro Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
Weighted Average - Precision: 0.89, Recall: 0.89, F1-Score: 0.89
```

The outcome of this from the second feature extraction attempt is:



```
Confusion Matrix:
[[243  29]
 [ 11 260]]

Accuracy: 92.63%

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.89      0.92       272
           1       0.90      0.96      0.93       271

    accuracy                           0.93       543
   macro avg       0.93      0.93      0.93       543
weighted avg       0.93      0.93      0.93       543


Micro Average - Precision: 0.93, Recall: 0.93, F1-Score: 0.93
Macro Average - Precision: 0.93, Recall: 0.93, F1-Score: 0.93
Weighted Average - Precision: 0.93, Recall: 0.93, F1-Score: 0.93
```