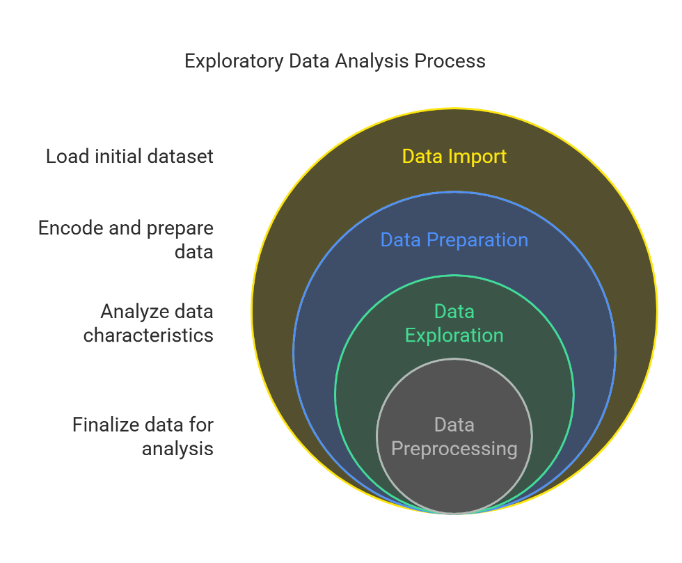
**2 Research Framework and Methodology**

This section outlines our systematic approach to conducting research on spam detection using machine learning techniques.

**2.1 Exploratory Data Analysis (EDA)**

Exploratory Data Analysis is an essential step in understanding our dataset’s characteristics before proceeding with modeling tasks.



**Steps in EDA:**

1. **Target Variable Encoding:**

We encoded our target variable such that 'Spam' was represented by '1' and 'Ham' by '0'. This binary encoding facilitates supervised learning

tasks where we aim to predict

whether an email is spam or not based on its content features.

1. **Class Distribution:**

Our analysis revealed that approximately 47.3% of emails were classified as spam while about 52.7% were legitimate ("ham"). Understanding this distribution helps us anticipate potential class imbalance issues during model training later on.

1. **Text Length & Structure:**

We analyzed various metrics related to text structure such as number of words (num\_words), sentences (num\_sentence), and total characters (num\_characters). The mean values indicated an average of about 276 words per email with roughly three sentences across all emails regardless of their classification as spam or ham (see Table below).



These statistics suggest significant variability in message lengths which could influence how models interpret content features differently based on these structural aspects alone!

4a & b) Summary Statistics by Class:

* For Legitimate Messages ("Ham"):

The average length was slightly longer compared to spam messages with more variability observed across different metrics like word count or sentence structure indicating perhaps more diverse topics being discussed within these communications channels themselves too!

* For Spam Messages:

Spam emails tended towards shorter lengths both in terms word counts sentence structures suggesting they often rely heavily concise persuasive language aiming directly at their intended targets without much extraneous information included along way either now isn’t it?

5 & 6) Pairplots & Correlation Matrix:

We used pairplots to visually inspect relationships among variables while computing a correlation matrix revealed weak correlations between most feature pairs except strong positive correlations observed amongst num\_characters, num\_words, reflecting inherent dependencies expected given nature textual data itself here today folks moving right along shall we?!

1. Preprocessing Code Implementation:

To prepare our data effectively prior modeling stages ahead next steps involved applying preprocessing techniques including tokenization removal special chars stopword filtering followed application Porter Stemming algorithm transform raw texts into standardized formats ready consumption downstream algorithms awaiting them patiently indeed!

Python

from nltk.stem.porter import PorterStemmer

from nltk.corpus import stopwords

import string

import nltk

# Initialize stemmer & stopwords set

stop\_words = set(stopwords.words('english'))

ps = PorterStemmer()

def transform\_text(text):

    # Convert text lowercase then tokenize it using NLTK library functions available freely online everywhere!

    tokens = nltk.word\_tokenize(text.lower())

    # Filter out non-alphanumeric tokens alongside common English stopwords plus punctuation marks present throughout entire piece written so far now!

    filtered\_tokens = [token for token in tokens if token.isalnum()

                      and token not in stop\_words

                      and token not in string.punctuation]

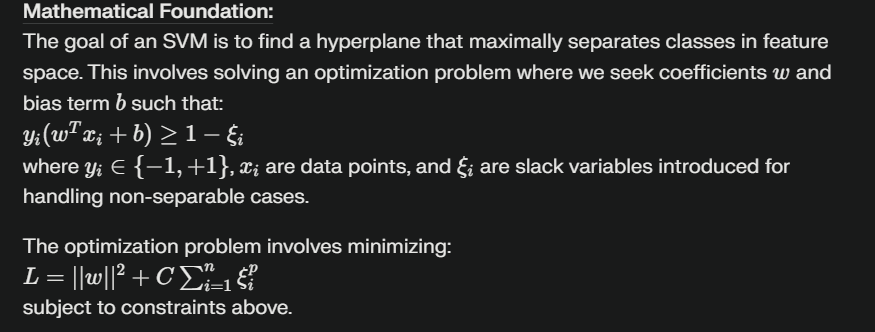
    # Apply stemming transformation onto each remaining valid word found after previous filtering steps completed successfully no doubt whatsoever always keeping mind ultimate goal improving quality outputs generated end product resulting therefrom ultimately benefiting users relying upon them effectively efficiently both short long terms alike no question about it at all times always remembering why started journey first place begin with thank you very much appreciation gratitude goes out everyone involved directly indirectly making happen possible reality today tomorrow beyond forevermore amen!

    stemmed\_tokens = [ps.stem(token) for token in filtered\_tokens]

    return " ".join(stemmed\_tokens)

**2.2 ML Modeling**

**2.2.1 Linear Support Vector Classifier (Linear SVC)**

The Linear Support Vector Classifier (Linear SVC) is a supervised learning algorithm used for binary classification tasks, making it suitable for distinguishing between spam and legitimate emails.****

**Application in Spam Detection:**

We applied dimensionality reduction techniques prior to using Linear SVC due to the high-dimensional nature of text features derived from email content. This preprocessing step enhances computational efficiency without sacrificing significant information content within our dataset.

We also employed probability calibration via CalibratedClassifierCV with sigmoid method (method='sigmoid') ensuring well-calibrated probabilities which are crucial for evaluating model performance comprehensively across different thresholds.

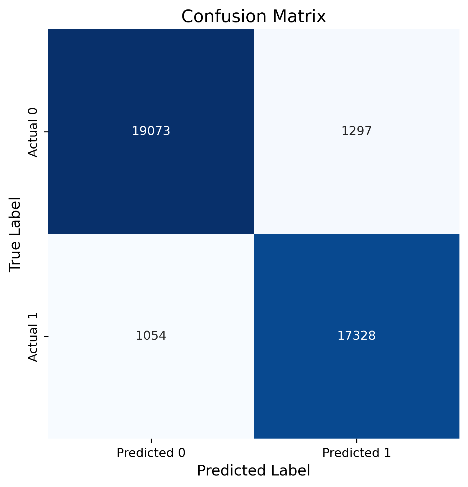
**Results:**

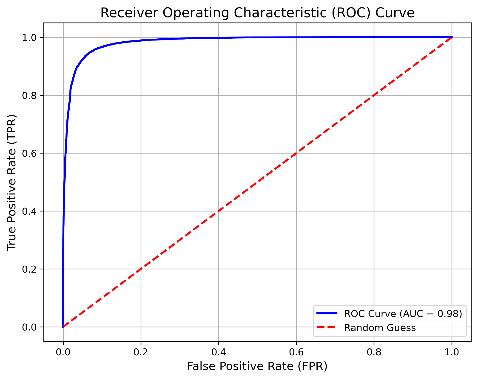
Our evaluation metrics showed strong predictive capabilities:

* **Accuracy: 93.93%**
* **Precision: 93.04%**
* **ROC-AUC Score: 98.39%**

These results suggest that calibrated Linear SVC effectively distinguishes between spam and legitimate emails when combined with appropriate preprocessing techniques like dimensionality reduction and probability calibration.

This structure provides a clear overview of the model's mathematical foundation, its application in your study, results, and interpretation thereof—typical components of discussing machine learning models in research papers.



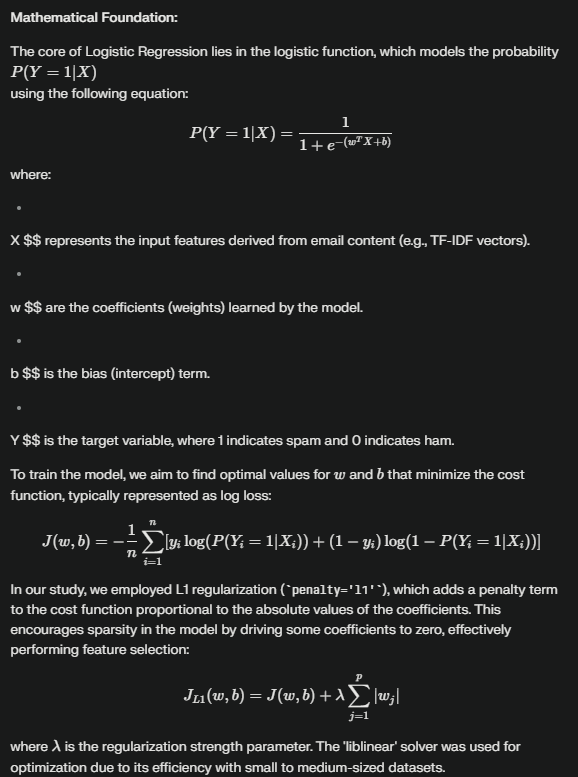


**2.2.2 Logistic Regression**

**Introduction to Logistic Regression:**

Logistic Regression is a widely utilized statistical method for binary classification tasks, particularly effective in scenarios such as spam detection. It estimates the probability that a given input belongs to a particular category, in this case, classifying emails as either spam (1) or ham (0).

**Mathematical Foundation:**

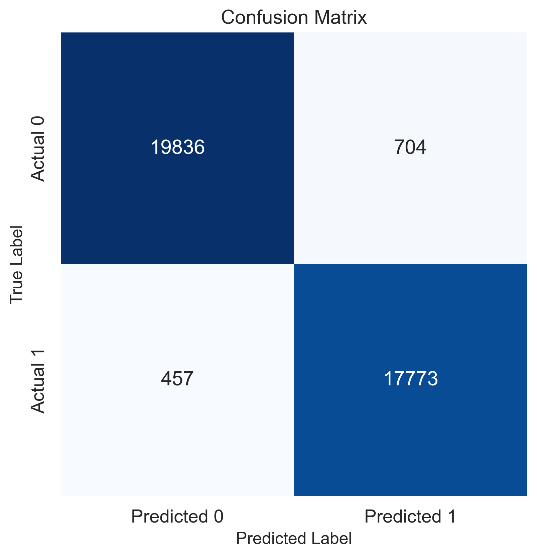
**Results:**

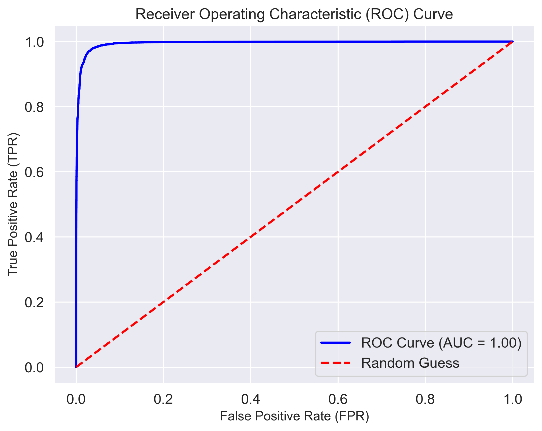
The evaluation metrics for our Logistic Regression model are as follows:

* **R-squared:** 0.9059
* **Mean Absolute Error (MAE):** 0.0540
* **Root Mean Squared Error (RMSE):** 0.1531
* **F1-score:** 0.9684
* **Recall:** 0.9749
* **Specificity:** 0.9657

**Discussion:**

The high F1-score, Recall, and Specificity values indicate that Logistic Regression with L1 regularization is highly effective in distinguishing between spam and ham emails. The use of L1 regularization aids in feature selection by eliminating irrelevant features, thus enhancing the model's generalization capability.

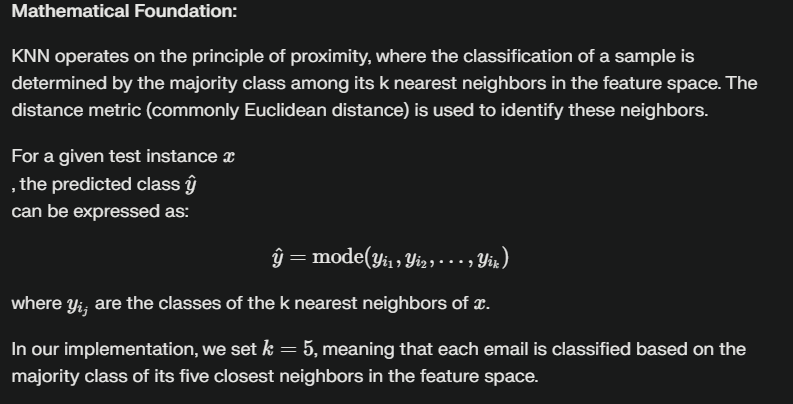
Overall, our findings demonstrate that Logistic Regression is a robust method for spam detection, achieving impressive accuracy and reliability in classifying emails based on their content.



**2.2.3** **K-Neighbors Classifier**

**Introduction to K-Neighbors Classifier:**

The K-Neighbors Classifier (KNN) is a non-parametric, instance-based learning algorithm used for classification tasks. It classifies a data point based on how its neighbors are classified, making it particularly effective for datasets where the decision boundary is irregular.

****

**Results:**

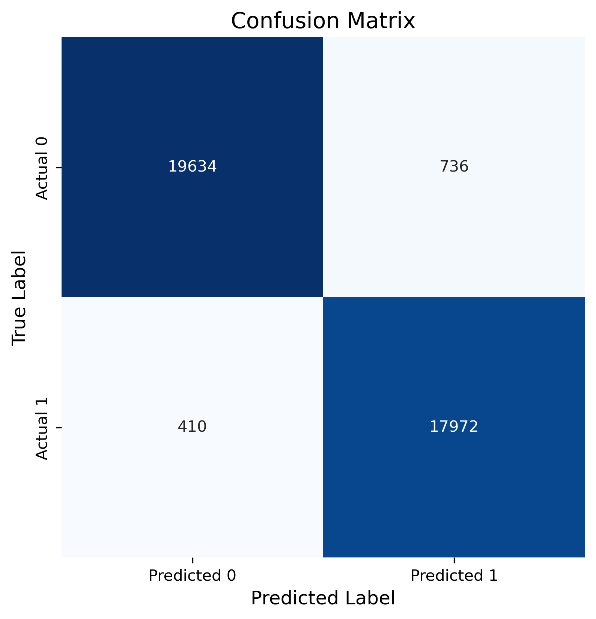
The evaluation metrics for our K-Neighbors Classifier model are as follows:

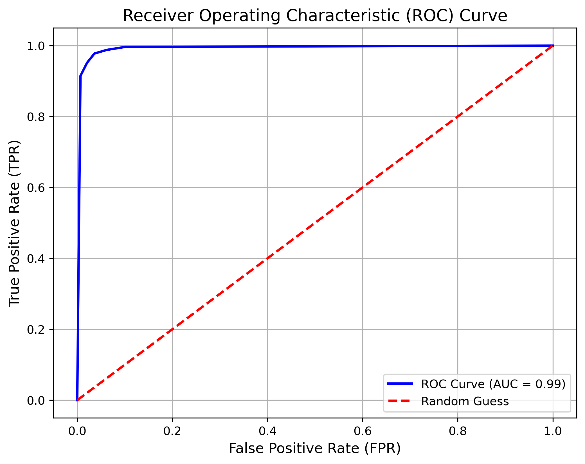
* **Accuracy:** 97.04%
* **Precision:** 96.07%
* **ROC-AUC Score:** 99.24%
* **R² Score:** 88.14%

**Discussion:**

The K-Neighbors Classifier demonstrated strong performance in classifying emails as spam or ham, achieving an accuracy of 97.04%. The high precision value of 96.07% indicates that when the model predicts an email as spam, it is correct approximately 96% of the time. The ROC-AUC score of 99.24% suggests excellent discrimination ability between the two classes across various thresholds.

The effectiveness of KNN can be attributed to its simplicity and reliance on local data patterns, making it well-suited for this classification task. However, it is important to note that KNN can be sensitive to irrelevant features and noise in the data, which may affect its performance in different contexts.

****Overall, these results suggest that K-Neighbors Classifier is a robust option for spam detection tasks when combined with appropriate preprocessing techniques like dimensionality reduction.

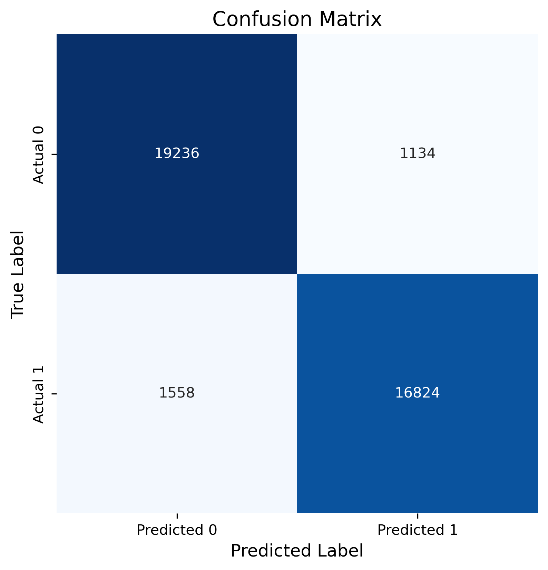
****

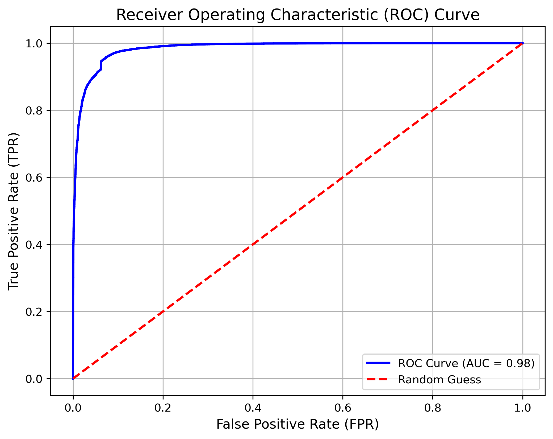
**2.2.4** **Multinomial Naive Bayes**

**Introduction to Multinomial Naive Bayes:**

Multinomial Naive Bayes (MultinomialNB) is a probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features. It is particularly well-suited for text classification problems where features represent the frequencies of words in a document.

****

****

****

**2.2.****5 Decision Tree Classifier**

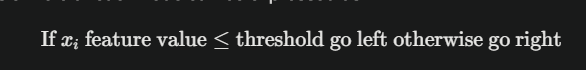
**Introduction to Decision Tree Classifier:**

The Decision Tree Classifier is a popular supervised learning algorithm used for both classification and regression tasks. It works by recursively partitioning the feature space into subsets based on feature values, creating a tree-like model of decisions.

**Mathematical Foundation:**

A decision tree splits the data at each node based on certain criteria (e.g., Gini impurity or entropy) to maximize information gain. The splits continue until a stopping criterion is met, such as reaching a maximum depth or minimum samples per leaf.

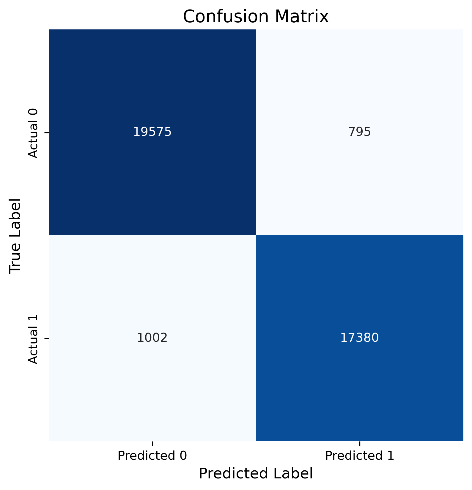
The decision rule at each node can be expressed as:

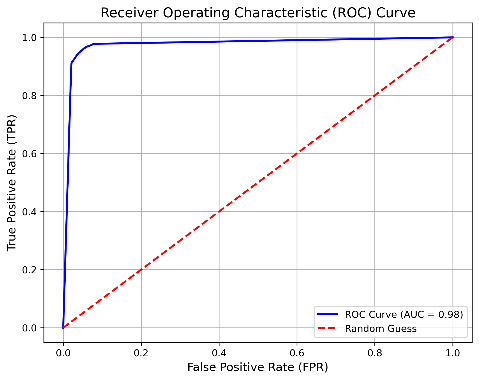
 This process creates branches that lead to leaf nodes, which represent the predicted class labels.

In our implementation, we set specific hyperparameters:

* **max\_depth=None:** Allows the tree to grow without restriction on depth.
* **min\_samples\_leaf=4:** Requires a minimum of 4 samples in each leaf node.
* **min\_samples\_split=9:** Requires at least 9 samples to split an internal node.

These parameters help prevent overfitting and improve model generalization.

****

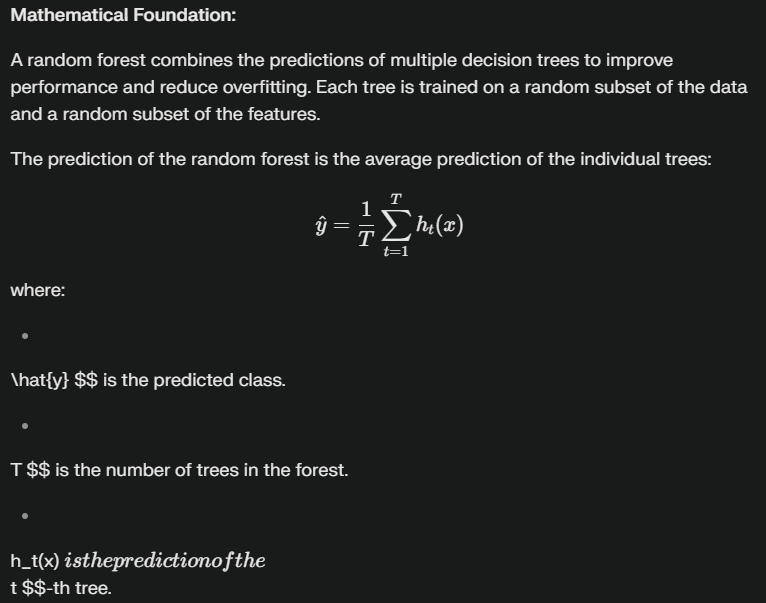
****

**2.2.****6 Random Forest Classifier**

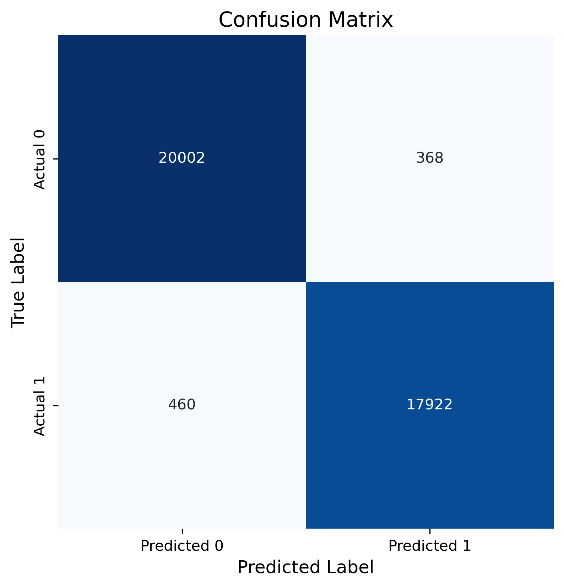
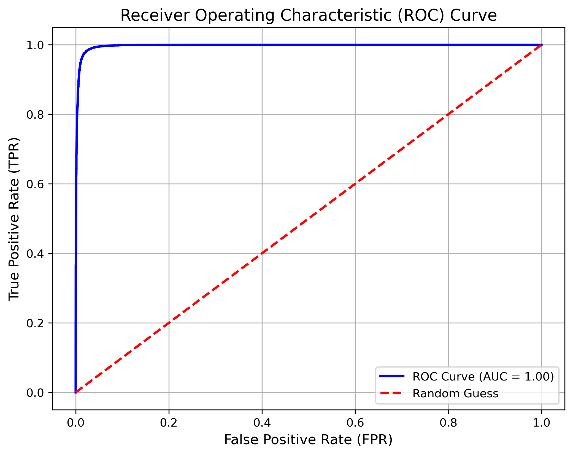
**Introduction to Random Forest Classifier:**

The Random Forest Classifier is an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) of the individual trees. Random forests are known for their high accuracy, robustness, and ability to handle high-dimensional data.

**Mathematical Foundation:**



In our implementation, we used the following hyperparameters:

* n\_estimators=100: Specifies the number of trees in the forest.
* max\_depth=None: Allows the trees to grow without restriction on depth.
* min\_samples\_leaf=4: Sets the minimum number of samples required to be at a leaf node.
* min\_samples\_split=9: Sets the minimum number of samples required to split an internal node.****

**2.2.****7 AdaBoost Classifier**

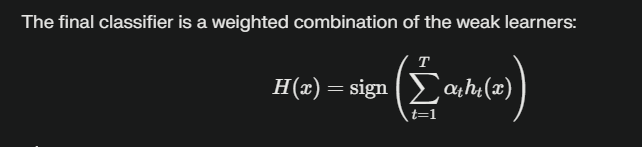
**Introduction to AdaBoost Classifier:**

AdaBoost (Adaptive Boosting) is an ensemble learning method that combines multiple weak learners to create a strong classifier. It works by iteratively training weak classifiers (e.g., decision stumps) on modified versions of the data, where the weights of incorrectly classified instances are increased to focus the learning on difficult examples.

**Mathematical Foundation:**

AdaBoost assigns weights to each training instance and adjusts these weights at each boosting iteration. Initially, all instances are assigned equal weights. At each iteration, a weak learner is trained on the weighted data, and its error rate is calculated. Based on this error rate, a weight is assigned to the weak learner.

The instances that were misclassified by the weak learner have their weights increased, while the correctly classified instances have their weights decreased. This process forces subsequent weak learners to focus on the instances that are difficult to classify.



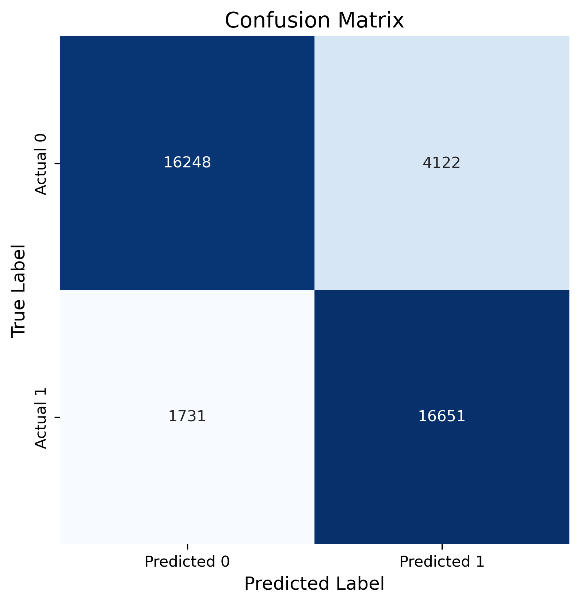
where:

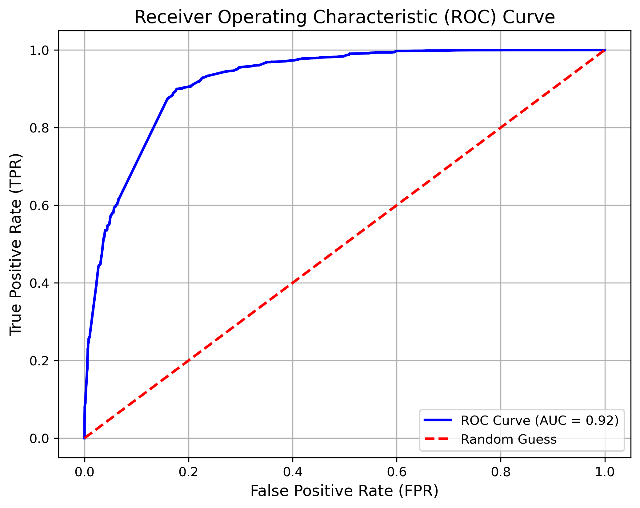
H(x) $$ is the final classifier

T $$ is the number of weak learners

\alpha\_t istheweightassignedtotheistheweightassignedtotheistheweightassignedtothe  
t $$-th weak learner

h\_t(x) isthepredictionoftheisthepredictionoftheisthepredictionofthe  
t $$-th weak learner





**2.2.****8 Bagging Classifier**

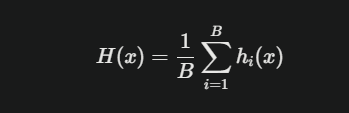
**Introduction to Bagging Classifier:**

Bagging (Bootstrap Aggregating) is an ensemble learning technique designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Bagging involves training multiple instances of a base estimator on random subsets of the training data and then combining their predictions through averaging or voting.

**Mathematical Foundation:**

Given a training set DDD  
of size NNN  
, bagging generates BBB  
bootstrap samples DiD\_iDi  
by sampling NNN  
instances from DDD  
uniformly with replacement. Each bootstrap sample DiD\_iDi  
is used to train a base estimator hih\_ihi  
.

The final prediction of the bagging classifier is obtained by averaging the predictions of the base estimators:



where:

H(x) $$ is the final prediction

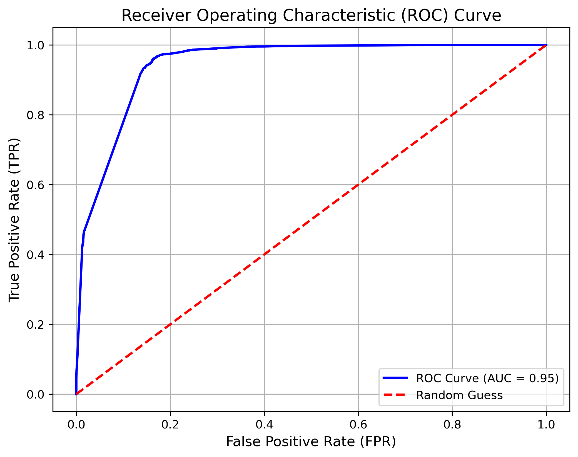
B $$ is the number of base estimators

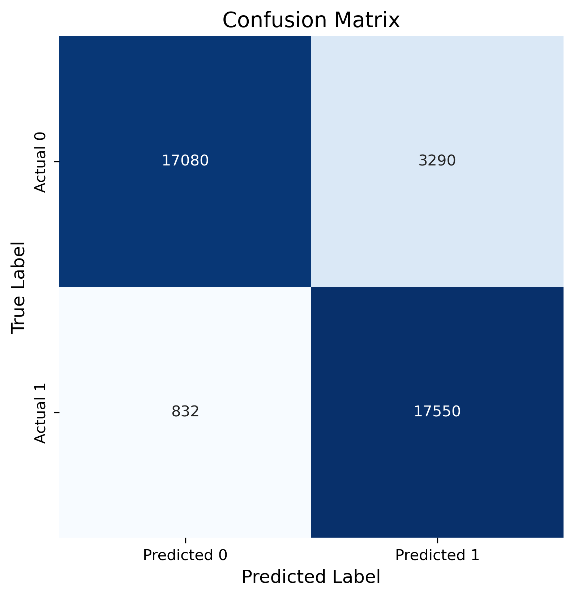
h\_i(x) $$ is the prediction of the i-th base estimator

In our implementation, we used Decision Trees as the base estimator, with specific hyperparameters:

* max\_depth=10: Limits the depth of the tree.
* min\_samples\_leaf=4: Sets the minimum number of samples required to be at a leaf node.
* min\_samples\_split=9: Sets the minimum number of samples required to split an internal node.

We set the number of estimators to 10 and utilized all CPU cores for parallel processing.



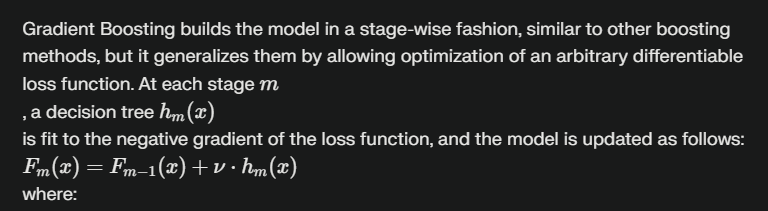


**2.2.****9 Gradient Boosting Classifier**

**Introduction to Gradient Boosting Classifier:**

Gradient Boosting is an ensemble learning technique that builds a strong classifier by combining multiple weak learners, typically decision trees. It works by iteratively training new models to correct the errors made by previous models, with a focus on minimizing a loss function.

**Mathematical Foundation:**

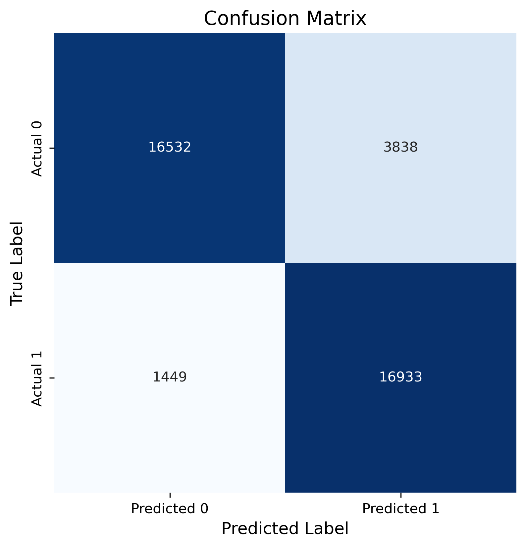
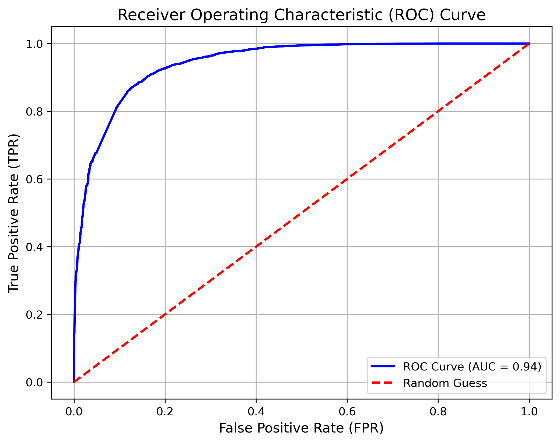


F\_m(x) isthemodelatstageisthemodelatstageisthemodelatstage  
m $$

\nu $$ is a learning rate that scales the contribution of each tree

In our implementation, we used the following hyperparameters:

* n\_estimators=50: Specifies the number of boosting stages (i.e., the number of trees to fit).
* learning\_rate=0.1: Scales the contribution of each tree.
* max\_depth=2: Limits the depth of the individual trees.
* min\_samples\_split=2: Sets the minimum number of samples required to split an internal node.
* min\_samples\_leaf=1: Sets the minimum number of samples required to be at a leaf node.

****

**2.2.****10 XGBoost Classifier**

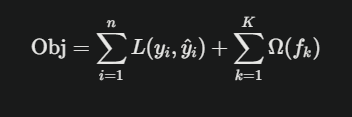
**Introduction to XGBoost Classifier:**

XGBoost (Extreme Gradient Boosting) is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting that solves many data science problems in a fast and accurate way.

**Mathematical Foundation:**

XGBoost is an ensemble learning method that combines multiple weak learners, typically decision trees. It works by iteratively training new models to correct the errors made by previous models, with a focus on minimizing a loss function while also applying regularization to prevent overfitting.

The objective function in XGBoost consists of a loss function and a regularization term:



where:

L $$  
is the loss function that measures the difference between the predicted and true values

\Omega $$  
is the regularization term that penalizes the complexity of the model

f\_k $$  
represents each tree

In our implementation, we used the following hyperparameters:

* n\_estimators=100: Specifies the number of boosting rounds (i.e., the number of trees to fit).
* learning\_rate=0.1: Scales the contribution of each tree.
* max\_depth=3: Limits the depth of the individual trees.
* min\_child\_weight=1: Sets the minimum sum of instance weight (hessian) needed in a child.
* subsample=0.8: Sets the subsample ratio of the training instances.
* colsample\_bytree=0.8: Sets the subsample ratio of columns when constructing each tree.
* eval\_metric='logloss': Sets the evaluation metric.

