**Faculty of Engineering & Technology Electrical & Computer Engineering Department**

**ENCS5341**

**Machine Learning**

# Assignment 3

**Prepared by:**

| | |
|---|---|
| Alhasan Manasra | 1211705 |
| Mohammad Khdour | 1212517 |

**Instructor**: Dr.Ismail Khater

**Section:** 2 & 3

Date: December 28, 2024

## Table Of Contents

Table Of Contents ..... II

Table Of Figures ..... III

Table Of Tables ..... IV

Dataset and Attributes Description ..... 1

Methodology ..... 2

Part 1: K-Nearest Neighbors (KNN) ..... 2

Part 2: Logistic Regression ..... 4

Part 3: Support Vector Machines (SVM) ..... 6

Part 4: Ensemble Methods ..... 7

Conclusion ..... 10

## Table Of Figures

Figure 1 - No Missing Values .......... 2

Figure 2 - The evaluation of the KNN model based on the test dataset using several metrics, a confusion matrix, and an ROC curve. .......... 3

Figure 3 - The performance of the K-Nearest Neighbors (KNN) algorithm .......... 3

Figure 4 - number of neighbors (K) and the cross-validated accuracy of the KNN model. .......... 4

Figure 5 - Confusion Matrix and a ROC Curve, accompanied by evaluation metrics .......... 5

Figure 6 - Performance metrics of the Logistic Regression .......... 5

Figure 7 - Comparison of metrics for each kernel. .......... 7

Figure 8 - Performance Comparison of Ensemble Methods (AdaBoost vs. Random Forest) .......... 9

## Table Of Tables

Table 1 - The confusion matrix ........ 3

Table 2 - The confusion matrix results part 2 ........ 5

Table 3 - Performance Metrics of Logistic Regression with L1 and L2 Regularization ........ 6

Table 4 - Compare the performance using classification metrics ........ 6

Table 5 - AdaBoost Model Performance Metrics ........ 8

Table 6 - Random Forest Model Performance Metrics ........ 8

Table 7 - Performance of AdaBoost and Random Forest ........ 9

## Dataset and Attributes Description

According to the mushroom dataset. It was cleaned using various techniques such as Modal imputation, one-hot encoding, z-score normalization, and feature selection. It contains 54034 samples and 9 features distributed in the following form:

1. **cap-diameter**: The size of the mushroom cap, measured numerically in millimeters.
2. **cap-shape:** The shape of the mushroom cap, represented as categorical values encoded by one how encoder.
3. **gill-attachment**: Describes how the gills are attached to the stem, encoded as numerical values.
4. **gill-color:** The color of the gills, represented by categorical values encoded as numbers.
5. **stem-height:** The height of the mushroom stem, measured as a continuous numerical value in centimeters.
6. **stem-width:** The width of the mushroom stem, measured as a continuous numerical value in millimeters.
7. **stem-color:** The color of the mushroom stem, encoded as numerical categories.
8. **season**: The season in which the mushroom is found, encoded numerically (e.g., 1 for spring, 2 for summer, etc.).
9. **class:** The target variable, indicating the mushroom's classification 0 for edible, 1 for poisonous.

## Methodology

In the first we selected dataset from Kaggle website and we choose the mushroom.csv file. After that we start working on our assignment parts.

## Part 1: K-Nearest Neighbors (KNN)

### Description

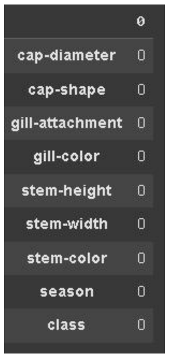
K-Nearest Neighbors (KNN) is a simple, non-parametric, and supervised machine learning algorithm which is a classification problem. It operates on the principle that similar data points are close to each other in the feature space. KNN does not teach a model, it just remembers the entire group of training data and uses that information to predict by finding the K-nearest neighbors to the input based on the chosen distance metric (like Euclidean distance). Must be labeled under class. After that, the classifications (first K neighboring groups majority voting) would set the most frequent set, which is KNN.

### implementation

We start of the data preparation phase, which is the initial step before applying machine learning algorithms. This step ensures the dataset is clean, ready for analysis, and free of missing values.

We imported required libraries for machine learning we implement KNN using KNeighborsClassifier. Then Loaded the dataset from the CSV file and Checked for missing values to validate data integrity.

So we ensures the dataset is free of missing values, which is important for reliable KNN performance because KNN relies on distance calculations.



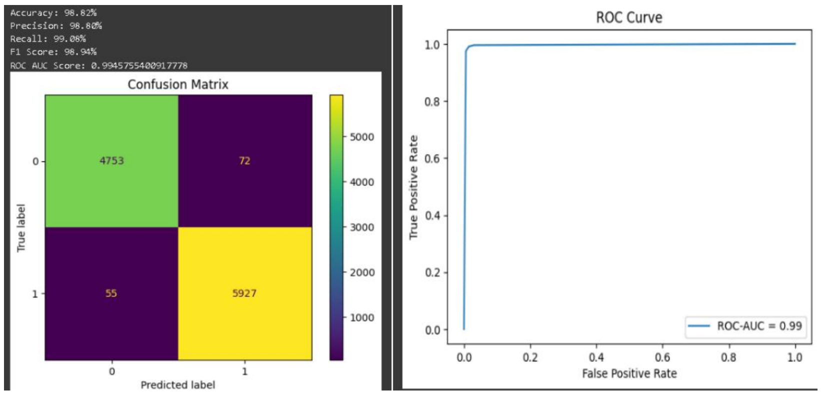
| | 0 |
|---|---|
| cap-diameter | 0 |
| cap-shape | 0 |
| gill-attachment | 0 |
| gill-color | 0 |
| stem-height | 0 |
| stem-width | 0 |
| stem-color | 0 |
| season | 0 |
| class | 0 |

*Figure 1 - No Missing Values*

After that, the data were split into features, X, and target labels, y. To make sure the evaluation process is robust, split the data into an 80% training set and a 20% test set using train_test_split.

Before training the KNN model, features were normalized using StandardScaler. It standardized the data- a very important step. KNN relies on computation with distances, and thus unscaled features have different weights. The KNN classifier is instantiated with K=3, which means for the classification of any sample three nearest neighbors will be considered. The model will now be trained on the training dataset using the fit method.

Model performance was conducted on the test dataset, giving both class labels, namely y_pred, and the probabilities of the positive class-y_prob. Performance was checked using accuracy, precision, recall, F1-Score, and ROC-AUC, all standard performance metrics. Accuracy determined how well overall predictions were. Meanwhile, precision and recall show how well a model does in terms of properly classifying positive instances of a class. In turn, F1-Score measured a proper balance between Precision and Recall, while ROC-AUC evaluated the performance regarding distinguishing between classes.



```
Accuracy: 98.82%
Precision: 98.80%
Recall: 99.08%
F1 Score: 98.94%
ROC AUC Score: 0.9945755400917778
```

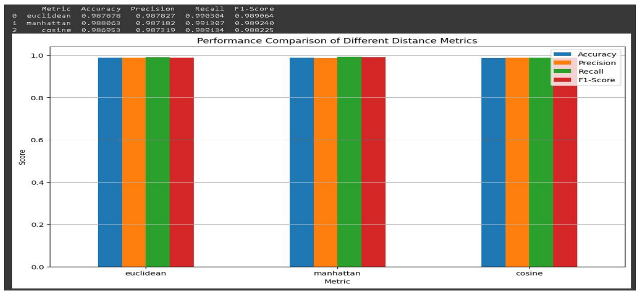
*Figure 2 - The evaluation of the KNN model based on the test dataset using several metrics, a confusion matrix, and an ROC curve.*

Therefore, the KNN model applied on a test dataset showed outstanding results due to several metrics. Indeed, the model with its gained accuracy achieved about 98.82%, demonstrating correctness for nearly all instances during tests. With 98.80%, the precision proved that nearly all foreseen positive instances were just so. Recall with the rate of 99.08% underlines that among actual positive ones, only some were missed by a model. The F1-Score of 98.94% balances Precision and Recall, confirming the strong classification capability of the model. Moreover, the ROC-AUC score of 0.99 underlines an excellent capability of discrimination between the two classes, as further visualized in the ROC curve, which is close to the top-left corner, showing an optimal trade-off between TPR and FPR.

| Actual / Predicted | Negative | Positive |
|---|---|---|
| Negative | 4753 (TN) | 72 (FP) |
| Positive | 55 (FN) | 5927 (TP) |

*Table 1 - The confusion matrix*

This step aims at trying different metrics, namely Euclidean, Manhattan, and Cosine, to run the KNN algorithm. To that end, the KNN model was trained for every metric using k=5 neighbors on the test set to make the predictions. In this model, the metrics used are Accuracy, Precision, Recall, and F1-Score, each being computed for every metric. These results were then stored and converted into a DataFrame for better organization. Finally, their performances were visualized as a bar plot, and indeed it shows how various distance metrics influence the KNN model performance. In actual sense, this is where one will identify an appropriate choice of distance metric providing optimum classification results.

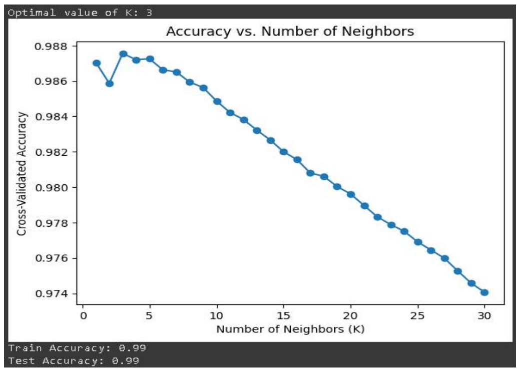


```
      Metric  Accuracy  Precision    Recall  F1-Score
0  euclidean  0.987878   0.987827  0.990304  0.989064
1  manhattan  0.988063   0.987182  0.991307  0.989240
2     cosine  0.986953   0.987319  0.989134  0.988225
```

*Figure 3 - The performance of the K-Nearest Neighbors (KNN) algorithm*

The results show the performance of KNN using three distance metrics: Euclidean, Manhattan, and Cosine. The Manhattan metric achieved the highest performance with an Accuracy of 98.86% and slightly better Precision, Recall, and F1-Score compared to the others. Euclidean performed similarly well with an Accuracy of 98.78%, while Cosine had slightly lower results at 98.69%. All metrics performed strongly, with Manhattan being the most effective for this dataset.

Then, using GridSearchCV, the optimum K was determined within the range for the KNN algorithm. In this case, 5-fold cross-validation ran the model for different ranges of K, starting from 1 and going all the way to 30. Its best K was 3, with the highest accuracy achieved through cross-validation. To represent the relationship between K and performance, the line plot carried out represents that performance decreases gradually in the case of increased K higher than the optimum. Lastly, the best model was re-trained by using optimal K, and it achieved both accuracies of training and testing equivalent to 99%, which also proved the effectiveness of chosen parameters.



```
Optimal value of K: 3
```

```
Train Accuracy: 0.99
Test Accuracy: 0.99
```

*Figure 4 - number of neighbors (K) and the cross-validated accuracy of the KNN model.*

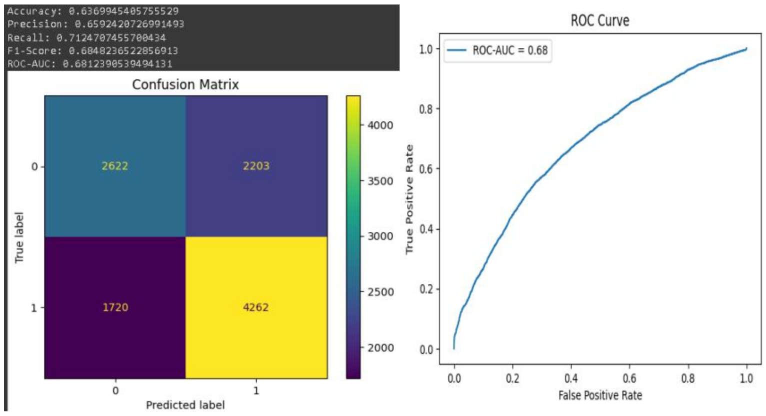
## Part 2: Logistic Regression

### Description

Logistic regression is a supervised machine learning algorithm used for binary classification where we use sigmod function, that takes input as independent variables and produces a probability value between 0 and 1. if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 otherwise it belongs to Class 0. Logistic regression is a statistical algorithm which analyze the relationship between two data factors.

### implementation

We implement and trained a model of Logistic Regression by using API LogisticRegression in the train dataset. In the final stage, prediction was carried out on the test set to make an estimation of how the model should perform and by using several metrics for this purpose like Accuracy, Precision, Recall, F1-Score, ROC-AUC, confusion matrix representation, ROC curve with TPR, and FPR. This will be the default implementation of Logistic Regression, which will be used as a basis for further experimentation with regularization.



```
Accuracy: 0.6369945405755529
Precision: 0.6592420726991493
Recall: 0.7124707455700434
F1-Score: 0.6848236522856913
ROC-AUC: 0.6812390539494131
```

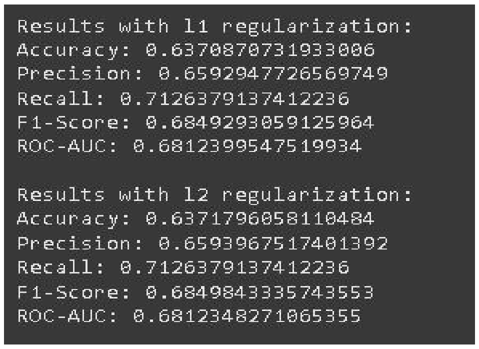
*Figure 5 - Confusion Matrix and a ROC Curve, accompanied by evaluation metrics*

The model achieved an accuracy of 63.7%, indicating the proportion of correct predictions overall. Precision was 65.9%, meaning 65.9% of the predicted positives were actually positive. Recall was 71.2%, reflecting the model's ability to correctly identify positive samples. The F1-Score, balancing precision and recall, was 68.5%. The ROC curve, with an AUC of 68.1%, shows the model's moderate capability in distinguishing between the two classes.

| Actual / Predicted | Negative | Positive |
|---|---|---|
| Negative | 2622 (TN) | 2203 (FP) |
| Positive | 1720 (FN) | 4262 (TP) |

*Table 2 - The confusion matrix results part 2*

After that at this stage, the impact of L1 regularization on Logistic Regression will be discussed. First, the model is trained, with each type of regularization being set by the penalty parameters, either 'l1' or 'l2', along with solver-'liblinear'. Then, using both, it predicts on the test set and computes the metrics Accuracy, Precision, Recall, F1-Score, and ROC-AUC. The comparison of the model's performance with different regularization techniques that will follow was supposed to shed light on how the technique influences the results.



```
Results with l1 regularization:
Accuracy: 0.6370870731933006
Precision: 0.6592947726569749
Recall: 0.7126379137412236
F1-Score: 0.6849293059125964
ROC-AUC: 0.6812399547519934

Results with l2 regularization:
Accuracy: 0.6371796058110484
Precision: 0.6593967517401392
Recall: 0.7126379137412236
F1-Score: 0.6849843335743553
ROC-AUC: 0.6812348271065355
```

*Figure 6 - Performance metrics of the Logistic Regression*

| Regularization | Accuracy | Precision | Recall | F1-Score | ROC-AUC |
|---|---|---|---|---|---|
| L1 (Lasso) | 63.71% | 65.93% | 71.26% | 68.49% | 68.12% |
| L2 (Lasso) | 63.72% | 65.94% | 71.26% | 68.50% | 68.12% |

*Table 3 - Performance Metrics of Logistic Regression with L1 and L2 Regularization*

These metrics show that both regularization techniques return very similar values. Accuracy, Precision, Recall, and F1-Score are overall measures of classification performance, while ROC-AUC reflects the capability of a model to tell classes apart. Both of them give similar results, hence the type of regularization does not strongly influence this dataset.

## Part 3: Support Vector Machines (SVM)

### Description

Support Vector Machine or SVM is a Supervised Learning algorithm, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

### Implementation

By explaining with different kernels and observing the results showed us how kernel functions influence the performance of SVMs. The better performance of the RBF kernel signifies that this kernel is more capable of dealing with complex, nonlinear datasets and, hence, the best choice for the considered problem. We initialize the SVM model with the default kernel, the Radial Basis Function (RBF), using the class SVC from the sklearn.svm module. This dataset is used to train the classification model, and it has been divided into X_train and y_train. It's the first step toward the performance evaluation of SVM.

To see the impact of various kernels, we will iteratively train the SVM model three times using three different types of kernels, namely linear, poly, and rbf. For each kind of kernel, we used the trained model to train on the training set and predict the outcomes on the test set (X_test).

For each kernel, it calculates the standard metrics of classification performance: Accuracy, Precision, Recall, and F1-Score. After getting the result of each kernel, it was stored and later transformed into a DataFrame to compare them better. At the end, the output was the summary of metrics over all three kernels, as seen in the table:

| Kernal | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| Linear | 64.19% | 66.91% | 69.86% | 68.35% |
| Polynomial | 76.82% | 78.73% | 79.64% | 79.18% |
| RBF | 90.07% | 91.57% | 90.34% | 90.97% |

*Table 4 - Compare the performance using classification metrics*

The results indicate that the choice of the kernel significantly impacts model performance. The RBF kernel outperformed the linear and polynomial kernels across all metrics, with the highest accuracy (90.07%) and F1-Score (90.97%). This suggests that the RBF kernel was better suited to handle the dataset's complexity and non-linear decision boundaries. The linear kernel showed the weakest performance, likely due to its inability to capture non-linear relationships in the data, while the polynomial kernel demonstrated moderate performance.

So after that we plotted as a grouped bar chart using matplotlib. The bar chart compares the performance metrics (Accuracy, Precision, Recall, and F1-Score) of the SVM model using three kernels: linear, polynomial, and rbf. It clearly shows that the rbf kernel achieves the highest performance across all metrics, followed by the polynomial kernel, while the linear kernel performs the least effectively. This indicates that the rbf kernel is the most suitable for handling the dataset's complexity, as it captures non-linear patterns more effectively than the other kernels.

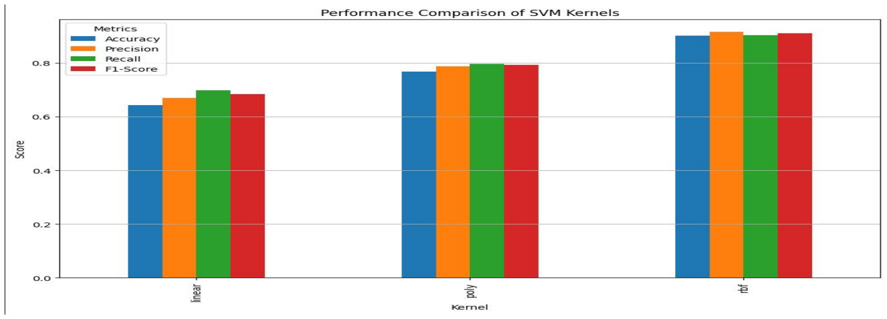


*Figure 7 - Comparison of metrics for each kernel.*

## Part 4: Ensemble Methods

### Description

### Boosting

is a sequential process where models are trained one after another, with each subsequent model focusing on the errors made by its predecessor. The idea is to weigh data points such that misclassified instances get more attention in the next iteration. Popular algorithms like AdaBoost, Gradient and Boosting are based on this approach. Boosting tends to reduce bias and is particularly effective for improving weak models, but it can be prone to overfitting if not carefully tuned.

### Bagging

Bootstrap Aggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting It is usually applied to decision tree method. Bagging is a special case of the model averaging approach.

### Implementation

We started this part by implementing the Boosting technique using the AdaBoost algorithm. First, instantiate an AdaBoostClassifier with 50 estimators and fit the robust ensemble model on the training dataset of X_train and y_train. After that, predict on the test dataset to get predictions and probabilities, namely y_pred_boost and y_prob_boost, respectively. At this stage, test the performance of the model based on the standard metrics of Accuracy, Precision, Recall, F1-Score, and ROC-AUC. These results provided some insight into the performance of Boosting in model performance enhancement using ensemble techniques, with metrics of moderately strong classification ability. This will provide a foundation for comparing the performance of AdaBoost against other ensemble and individual models.

| Metric | Value |
|---|---|
| Accuracy | 0.69 |
| Precision | 0.71 |
| Recall | 0.74 |
| F1-Score | 0.72 |
| ROC-AUC | 0.75 |

*Table 5 - AdaBoost Model Performance Metrics*

The AdaBoost model had an output class accuracy of 69% on most the correctly classified test samples. The precision was 71%, which is the rate at which true positive predictions are made out of all positive predictions. Recall equaled 74%, standing for the model's actual capability to identify positive samples. It balanced precision and recall using the F1-Score at 72%. It showed an ROC-AUC score of 75% because this model moderately distinguishes classes. This therefore indicates that AdaBoost does well on this dataset with reasonable classification accuracy and balance.

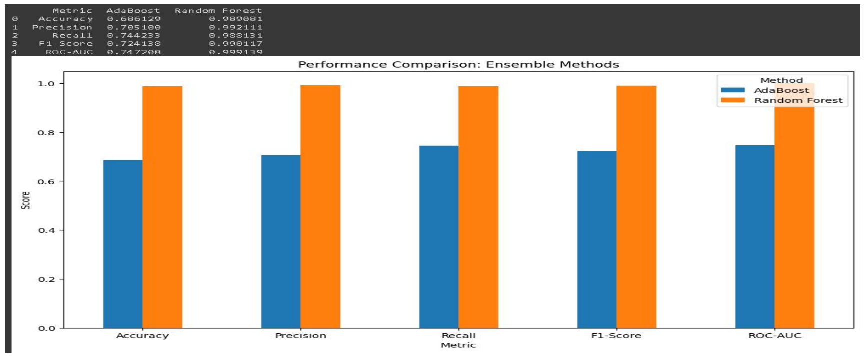
In the second task we do it by choosing the Random Forest algorithm, employing a bagging technique. The model is instantiated using API sklearn.ensemble.RandomForestClassifier. Some of the parameters defined included 50 estimators hence, the model will be an ensemble of decision trees. Now, we had to train our model on the training dataset to have it learn from the inherent patterns of the data. Afterwards, the trained model was tested on the test dataset to predict labels and estimate probabilities with respect to the evaluation metrics. The main classification metrics (Accuracy, Precision, Recall, F1-Score, and ROC-AUC) were calculated for model evaluation. This implementation is pointing out the strength of various bagging methods for controlling overfitting and how ensembling over diverse decision trees could result in improved classification.

| Metric | Value |
|---|---|
| Accuracy | 0.99 |
| Precision | 0.99 |
| Recall | 0.99 |
| F1-Score | 0.99 |
| ROC-AUC | 1 |

*Table 6 - Random Forest Model Performance Metrics*

This output is telling how the Random Forest model very efficiently classified the data with high accuracy. The 99% Accuracy means that nearly all the predictions made by the model were correct. Precision and Recall, at 99%, stand on the model's impressive capability of correctly classifying the positive samples while keeping false positives and false negatives as low as possible. The F1-score is the proper balance between precision and recall, showing the robustness of this model against the imbalance in datasets. Finally, the ROC-AUC score is 1.00, showing that the model has perfect discrimination capability between classes of positive and negative, hence showing an excellent performance. These results underpin the strength of ensemble methods such as Random Forest for high accuracy and reliability in predictions.

In this analysis, we will further discuss both ensemble methods with the comparative point of view on the performance measures they attained. Random Forest was outperforming AdaBoost on each metric: it reached a considerably higher accuracy, precision, recall, F1-score, and ROC AUC (0.99 vs. 0.68, 0.99 vs. 0.71, 0.99 vs. 0.74, 0.99 vs. 0.72, 1.00 vs. 0.75). This superior performance by RF signifies that it generalizes better on this dataset, probably due to its ability to handle high variance data and to capture complex patterns due to bagging. These results from the ensemble compare with individual models such as KNN, logistic regression, and SVM, indicating once again the strength of ensembles in enhancing predictive accuracy and robustness through various decision trees.



```
      Metric  AdaBoost  Random Forest
0   Accuracy  0.686129       0.989081
1  Precision  0.705100       0.992111
2     Recall  0.744233       0.988131
3   F1-Score  0.724138       0.990117
4    ROC-AUC  0.747208       0.999139
```

*Figure 8 - Performance Comparison of Ensemble Methods (AdaBoost vs. Random Forest)*

| Metric | AdaBoost | Random Forest |
|---|---|---|
| Accuracy | 0.69 | 0.99 |
| Precision | 0.71 | 0.99 |
| Recall | 0.74 | 0.99 |
| F1-Score | 0.72 | 0.99 |
| ROC-AUC | 0.75 | 1 |

*Table 7 - Performance of AdaBoost and Random Forest*

Random Forest clearly outperformed AdaBoost across all evaluation metrics. This demonstrates that Random Forest's bagging technique, which reduces variance and avoids overfitting, makes it better suited for this dataset. Furthermore, ensemble methods like Random Forest and AdaBoost showed significant improvements over individual models like (KNN, Logistic Regression, SVM), reinforcing the effectiveness of ensemble strategies for predictive accuracy and robustness.

## Conclusion

In this assignment, we explored various machine learning models, including KNN, Logistic Regression, SVM, and ensemble methods like AdaBoost and Random Forest. By evaluating their performance using key metrics, we observed that ensemble methods, particularly Random Forest, provided the best results due to their ability to combine multiple decision trees effectively. This highlights the importance of model selection and experimentation in achieving optimal performance for classification tasks.

- ### Working on project

We collaborated as a team to complete this assignment, helping each other finalize and edit our work. Initially, we divided the tasks: Alhasan worked on the first and third parts, while Khdour handled the second and fourth parts. For the report, we had extensive discussions to ensure we fully understood each other's work and explained it effectively. This teamwork allowed us to produce a comprehensive report explaining the parts in it.

The first part of our report take page.

Methodology part take 8 pages, I added tables for confusion matrix additionally.