Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ENCS5341

MACHINE LEARNING AND DATA SCIENCE

Assignment3 – Report

Prepared by:

Manar Shawahni, 1201086.

Layan Abuershaid, 1200098.

Instructor's Name: Dr. Yazan Abu Farha

Section: 1

28 December, 2024

Contents

1. Introduction and our Approach

This assignment explores machine learning algorithms to understand their performance and applications.

1.1. K-Nearest Neighbors (KNN)

KNN is a simple, supervised algorithm used for classification and regression. It classifies data points based on the majority class of their closest neighbors, measured using metrics like Euclidean, Manhattan, or Cosine. The number of neighbors (K) is a key parameter that impacts performance. KNN works well for small datasets but can be computationally expensive for large ones due to distance calculations for every query. [1]

In our approach, we experimented with three distance metrics: Euclidean, Manhattan, and Cosine by training a KNN classifier and performing predictions. Using grid search with cross-validation, we determined the optimal K for each metric by testing a range of values from 1 to 20, and the highest cross-validation accuracy is used to select the best k.

1.2. Logistic Regression

It is a statistical method for binary classification. It used a sigmoid function to predict classes and minimizes a loss function like cross-entropy during training. To avoid overfitting, it applies L1 (Lasso) and L2 (Ridge) regularization, making it a strong choice for classification tasks. [2]

We trained Logistic Regression models with both L1 and L2 regularization types to prevent overfitting and by using grid search with cross-validation, we tuned the hyperparameter C which controls the regularization strength and retrained the models with the optimal value. Finally, we compared the models' performance before and after hyperparameter tuning.

1.3. Support Vector Machines (SVM)

SVM is a flexible classification algorithm that separates data into classes by finding the best hyperplane and maximizing the margin between classes. It performs well in high-dimensional spaces and can handle non-linear data using kernel like linear, polynomial, and RBF. Using support vectors, it is memory efficient and less likely to overfit, making it suitable for datasets with many features. [3]

We trained SVM models using three kernels: Linear, Polynomial, and RBF. The Linear kernel is best for linearly separable data, while the Polynomial kernel adds flexibility for non-linear data by introducing polynomial terms. The RBF kernel handles more complex non-linear relationships. After training, we compared the performance of all kernels in a table to identify the best one for the dataset.

1.4. Ensemble Methods

These methods combine the predictions of multiple models to improve performance. In this assignment, we focused on two main techniques: **Boosting** and **Bagging**. Boosting, such as AdaBoost, trains weak models sequentially, giving more weight to errors to focus on harder cases. Bagging, including Random Forest, trains multiple base models in parallel and uses randomness in feature selection to improve accuracy and reduce overfitting. [4]

We trained models using AdaBoost, Bagging, and Random Forest to compare their performance. For AdaBoost, we used a Decision Tree with a maximum depth of 1 as the base estimator and set the number of estimators to 50. Bagging was implemented with the same number of estimators, and Random Forest, an extension of Bagging, included randomness in feature selection. Each model was trained on the dataset.

For each model of the 4, we evaluated performance using accuracy, precision, recall, F1-score, and ROC-AUC, classification reports, ROC curves and confusion matrices.

2. Results and Analysis

To start our analysis, we used the Iris dataset with 150 samples, each having four features [sepal length, sepal width, petal length and petal width] with a target variable representing the species.

```
Null values in the dataset:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64

Duplicate rows in the dataset:
1

 check duplicate rows after drop:
0
```

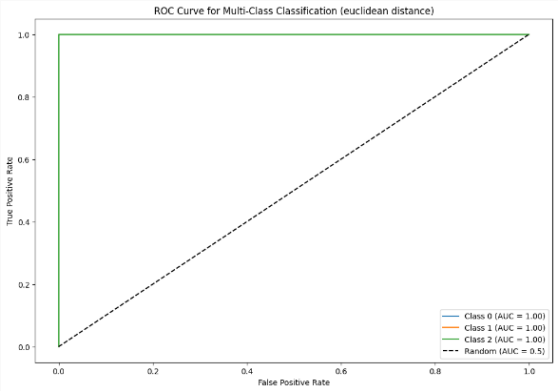
```
Dataset after standardize:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0

Shape of Training and Testing Sets:
X_train: (119, 4), y_train: (119,)
X_test: (30, 4), y_test: (30,)
```

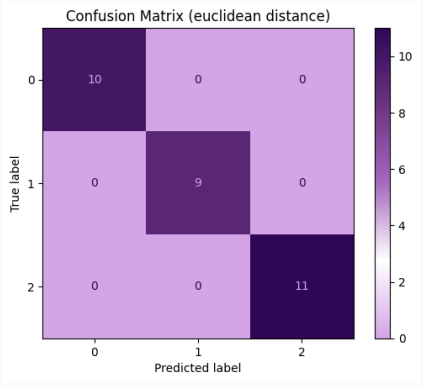
After checking the data, we found no missing values and one duplicate then removing one duplicate, we standardized the features and split the data into 119 samples for training and 30 for testing.

2.1. Part 1: KNN Euclidean and Manhattan distance:

```
Results for euclidean:
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

The results using **both Euclidean and Manhattan distance** metric show perfect classification performance, achieving an accuracy of 1.0. All classes (0, 1, and 2) have a precision, recall, and F1-score of 1.00, indicating that the model correctly identified all samples with no errors.





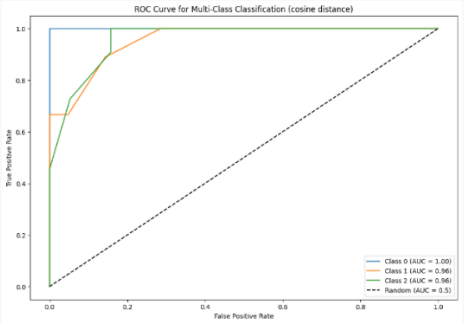
This ROC curve shows the performance classification, with all classes achieving an Area Under the Curve (AUC) of 1.0. The confusion matrix confirms this also, as all samples (10 for Class 0, 9 for Class 1, and 11 for Class 2) were correctly classified without errors in any category.

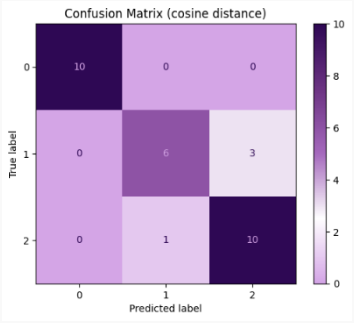
Cosine Distance

```
Results for cosine:
Accuracy: 0.8666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.86      0.67      0.75         9
           2       0.77      0.91      0.83        11

    accuracy                           0.87        30
   macro avg       0.88      0.86      0.86        30
weighted avg       0.87      0.87      0.86        30
```

The results differ significantly from those obtained with Euclidean and Manhattan distances. The overall accuracy is 86.67%.

Class 0 achieved perfect precision, recall, and F1-scores of 1.00, but performance was weaker for Classes 1 and 2, with recall at 0.67 and 0.91, and F1-scores at 0.75 and 0.83, respectively. So, cosine distance is less effective for this dataset compared to Euclidean distance. This shows that cosine distance is less effective for this dataset due to its variability in classification performance across classes.



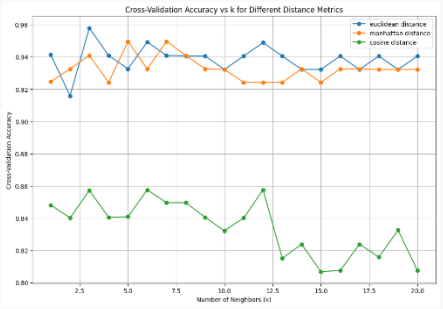


As shown above, Class 0 achieves an AUC of 1.0 which shows perfect separability, while classes 1 and 2 have lower AUCs 0.96, showing reduced distinction. The confusion matrix shows Class 0 is perfectly classified, but three Class 1 samples are misclassified as

Class 2, and one Class 2 sample is misclassified as Class 1, reflecting weaker performance for these classes.

Comparison of Distance Metrics:

```
Comparison table to find optimal k for each metric:
           Best k   CV Mean  Accuracy  Precision  Recall  F1-Score  \
euclidean     3.0  0.957971       1.0   1.000000     1.0  1.000000
manhattan     5.0  0.949638       1.0   1.000000     1.0  1.000000
cosine        6.0  0.857609       0.9   0.901389     0.9  0.899233

           Average ROC-AUC
euclidean         1.000000
manhattan         1.000000
cosine            0.973131
```



Both Euclidean and Manhattan distances achieved perfect classification performance (accuracy, precision, recall, F1-score all at 1.0) with optimal K values of 3 and 5, respectively. Their Average ROC-AUC of 1.0 confirms perfect separability of classes. Cosine distance had a lower overall accuracy 0.9 compared to Euclidean and Manhattan and its Average ROC-AUC of 0.973131 shows good but not perfect, so it's less effective.

The graph shows that Euclidean and Manhattan distances have higher and more steady accuracy, while Cosine distance is lower and less stable.

```
Best Model Results:
Metric: euclidean
Best k: 3
Cross-Validation value: 0.9579710144927537
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1-Score: 1.0
```

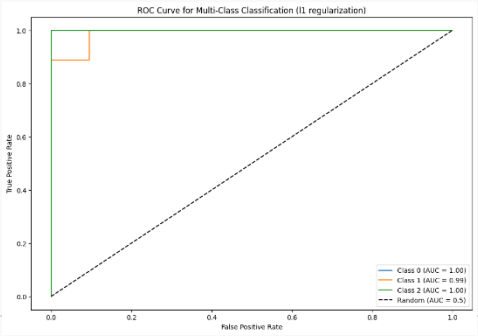
For the dataset, the Euclidean metric with K=3 is the best choice, because it achieves perfect performance across all metrics. Also, lower K value reduces computational costs and captures local patterns without over-smoothing.

2.2. Part 2: Logistic Regression

Results for l1 regularization (Before Hyperparameter Tuning):

```
Results for l1 regularization (Before Hyperparameter Tuning):
Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

This model achieved 96.67% accuracy. Class 0 and Class 2 had excellent performance, with perfect precision, recall, and F1-scores for Class 0 and almost perfect scores for Class 2. Class 1 had a lower recall at 0.89 but maintained high precision and F1-scores.

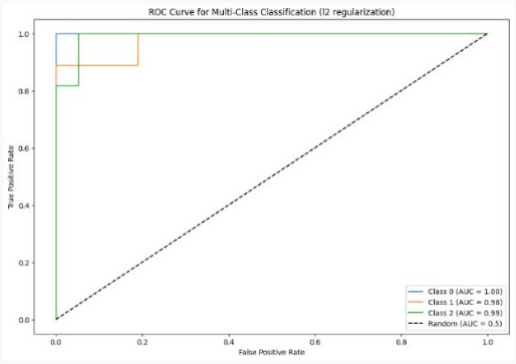


The curve shows near-perfect performance, with AUCs of 1.0 for Class 0 and Class 2, and 0.99 for Class 1.

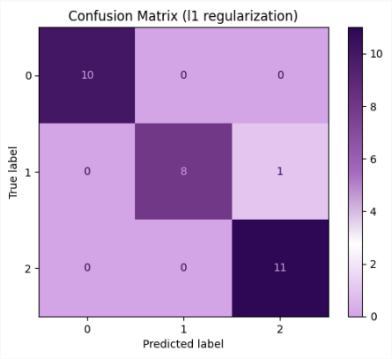
Results for l2 regularization (Before Hyperparameter Tuning):

```
Results for l2 regularization (Before Hyperparameter Tuning):
Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

As shown in the results here, both L1 and L2 regularization models gave the same accuracy and similar metrics.



The curve for L2 shows an AUC of 1.0 for Class 0 and 0.98 for Classes 1 and 2. Although the accuracy is the same as L1, the curves **differ** because of how regularization works. L1 makes models simpler by reducing some coefficients to zero, which can improve class separation. L2 applies regularization uniformly across all coefficients., creating smoother boundaries but less clear separation between some classes.



The confusion matrices for both models are also similar shows that both regularization techniques effectively prevent overfitting and maintain model accuracy.

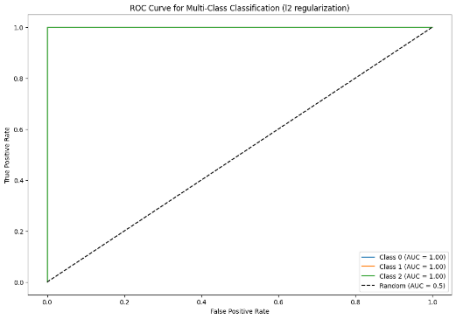
Results for l1 & l2 regularization (After Hyperparameter Tuning):

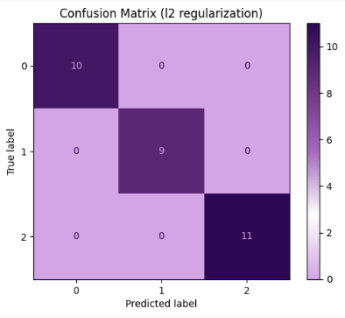
```
Results for l1 regularization (After Hyperparameter Tuning):
Best C value: 100
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
Results for l2 regularization (After Hyperparameter Tuning):
Best C value: 100
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Both L1 and L2 regularization models after hyperparameter tuning shows perfect performance across all classes, with an accuracy, precision, recall and F1-scores for all classes were 1.0, The macro and weighted averages were perfect too, showing that the models worked well.

The optimal C value of 100 allowed the models to fit the data closely without overfitting. Since both models produced identical results, L1 and L2 regularization are equally effective for this dataset, highlighting the flexibility of logistic regression.



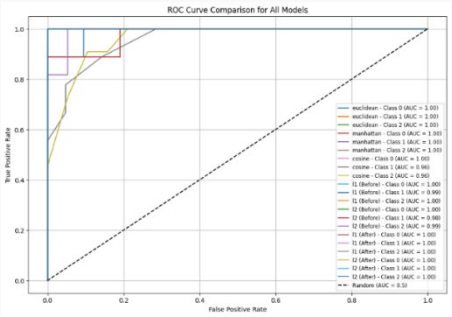


For l1 & l2 after hyperparameter tuning, the confusion matrix shows no misclassifications, and the ROC curve also shows perfect separability with AUC values of 1.0 for all classes.

Comparison KNN Distance Metrics with Logistic:

Comparison of all models:

	Accuracy	Precision	Recall	F1-Score	Average ROC-AUC
euclidean	1.000000	1.000000	1.000000	1.000000	1.000000
manhattan	1.000000	1.000000	1.000000	1.000000	1.000000
cosine	0.900000	0.901389	0.900000	0.899233	0.973131
Logistic (l1)	0.966667	0.969444	0.966667	0.966411	0.996473
Logistic (l2)	0.966667	0.969444	0.966667	0.966411	0.989756
Logistic (l1 tuned)	1.000000	1.000000	1.000000	1.000000	1.000000
Logistic (l2 tuned)	1.000000	1.000000	1.000000	1.000000	1.000000

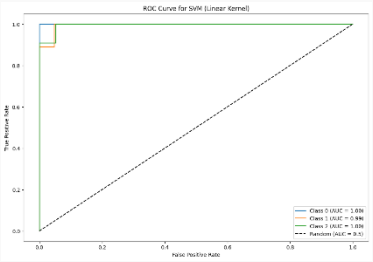


As seen above, **KNN using Euclidean and Manhattan distances** achieves perfect results for all metrics without needing any tuning, while **Cosine distance** performs worse with 90% accuracy and an Average ROC-AUC of 0.973 which emphasizing the importance of choosing the right metric. **Logistic Regression** initially lower performance than KNN with 96.67% accuracy and Average ROC-AUC of 0.996 (L1) and 0.990 (L2). But after tuning, both L1 and L2 reach perfect performance, matching KNN in all metrics. This shows that Logistic Regression is flexible and can achieve great results with tuning, while KNN is easier to use and performs perfectly without tuning, so Logistic Regression is a stronger option for tasks requiring tuning for different datasets.

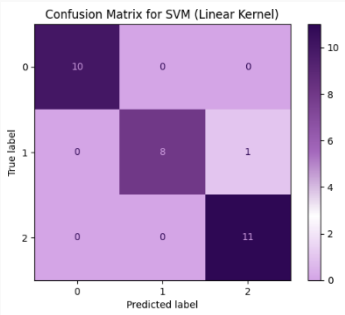
2.3. Part 3: Support Vector Machines (SVM):

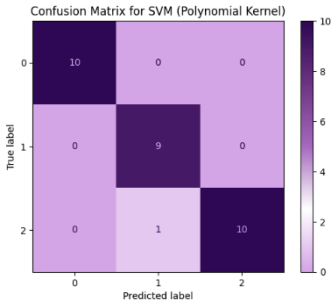
```
SVM (Linear Kernel) Classification Report:
Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30



SVM (Polynomial Kernel) Classification Report:
Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.90      1.00      0.95         9
           2       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

The performance of the SVM models varies depending on the kernel used. As seen in the results, both Linear and Polynomial kernels show high accuracy of 96.67% and similar precision, recall, and F1-scores across all classes. Their overall averages for these metrics are very close, so both kernels are work well but could improve. For example, the Linear kernel has a recall of 0.89 for Class 1, while the Polynomial kernel has a recall of 0.91. Also, the ROC-AUC for both kernels is 0.9966, meaning they separate the classes well.



The curve for both Linear and Polynomial shows high performance with AUC values of 1.0 for Class 0 and Class 2, and 0.99 for Class 1.



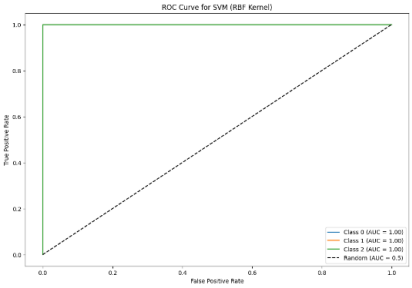


The Linear kernel misclassifies one Class 1 sample as Class 2, while the Polynomial kernel misclassifies one Class 2 sample as Class 1.

```
SVM (RBF Kernel) Classification Report:
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30



                  Accuracy  Precision (Macro)  Recall (Macro)  \
Linear Kernel     0.966667           0.972222        0.962963
Polynomial Kernel 0.966667           0.966667        0.969697
RBF Kernel        1.000000           1.000000        1.000000

                  F1-Score (Macro)   ROC-AUC
Linear Kernel             0.965899  0.996641
Polynomial Kernel         0.966583  0.996641
RBF Kernel                1.000000  1.000000
```

On the other hand, the RBF kernel performs perfectly on all metrics. The macro averages and ROC-AUC are also 1.0. The RBF ability to handle complex, non-linear relationships in the data. Compared to the Linear and Polynomial kernels it performs much better making it the best choice for this dataset.

The choice of kernel affects SVM performance, that the Linear and Polynomial kernels have high accuracy of 96.67% and strong ROC-AUC values of 0.9966, so they work well for simpler patterns but have lower recall for Class 1, which lowers their F1-scores. The RBF kernel performs perfectly on all metrics including accuracy, recall, F1-scores and ROC-AUC of 1.0. It handles complex patterns better. So, while the Linear and Polynomial kernels are faster and good for simpler data, the RBF kernel is the best for more complex datasets.



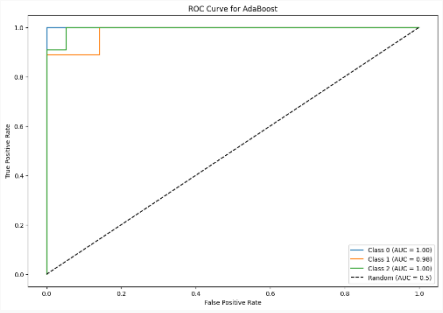
The ROC for the RBF kernel shows perfect results with AUC values of 1.0 for all classes which means the classes are completely separated.

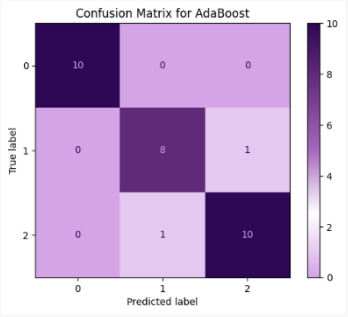
2.4. Part 4: Ensemble Methods:

AdaBoost Accuracy: 0.9333333333333333
AdaBoost Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.89	0.89	0.89	9
2	0.91	0.91	0.91	11
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

AdaBoost is a boosting method that performs well with an accuracy of 0.933 and high precision, recall, and F1-scores, but it struggles with Class 1, where recall is lower at 0.89 which reduces its overall accuracy. This shows that AdaBoost may not handle complex datasets as well as Bagging methods and might need more tuning or more advanced datasets to perform at its best.





The curve shows high AUC values, especially 1.0 for Classes 0 and 2. However, the two misclassifications in the confusion matrix show some difficulty in separating Classes 1 and 2, which could be improved

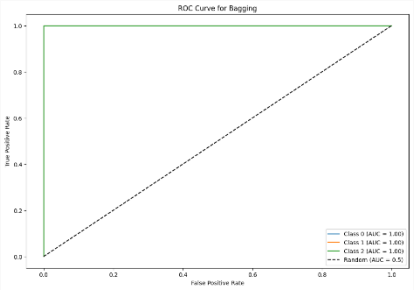
SVM (Polynomial Kernel) Classification Report:
Accuracy: 0.9666666666666667

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.90	1.00	0.95	9
2	1.00	0.91	0.95	11
accuracy			0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

SVM (RBF Kernel) Classification Report:
Accuracy: 1.0

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

Bagging methods and **Random Forest** achieve perfect accuracy, precision, recall, and F1-scores of 1.0 across all classes. These methods are highly effective for this dataset, as they combine multiple decision trees to reduce overfitting and improve consistency. Random Forest, as a specific type of Bagging, adds strength by using random feature selection, which makes it perform even better.



The ROC curve for Bagging and Random Forest shows perfect results with AUC values of 1.0 for all classes. So, they are effective and reliable more than AdaBoost**.**

Model Comparison Table:

	Evaluation Metric	AdaBoost	Bagging	Random Forest
0	Accuracy	0.933333	1.0	1.0
1	Precision	0.933333	1.0	1.0
2	Recall	0.933333	1.0	1.0
3	F1-score	0.933333	1.0	1.0
4	ROC-AUC	0.993114	1.0	1.0

As a final result, Bagging methods, particularly Random Forest, worked better due to their perfect results and ability to handle all classes effectively, while AdaBoost faced limitations with Class 1.

Ensemble methods are generally better than individual models because they combine the strengths of multiple base learners to improve accuracy and handle complex datasets. Bagging and Random Forest achieved perfect performance across all metrics, including accuracy, precision, recall, and F1-scores. They work well by using multiple decision trees and adding randomness (in Random Forest), which reduces overfitting and makes them very reliable. While AdaBoost performed well with an accuracy of 0.933, it struggled with Class 1 and did not match the consistent results of Bagging and Random Forest. For individual models like KNN, Logistic Regression, and SVM also performed strongly but had some weaknesses. KNN with Euclidean and Manhattan distances performed perfectly, but its results depended on choosing the right distance metric and number of neighbors. Logistic Regression needed tuning to achieve the best results, showing its flexibility but requiring extra effort. SVM performed perfectly with the RBF kernel, but the Linear and Polynomial kernels had lower recall for Class 1. So, while these individual models work well in certain situations, ensemble methods like Bagging and Random Forest are more reliable and handle complex data better.

We divided the work like this: Layan focused on implementing and experimenting with the K-Nearest Neighbors (KNN) algorithm, while I concentrated on training and evaluating the Logistic Regression model. We worked together on the Support Vector Machines (SVM) section and split the ensemble methods, with Layan handling Boosting and me handling Bagging. We both contributed to writing the report to ensure each section was clear.

3. References

[1] Python Machine Learning - K-nearest neighbors (KNN)

[2] Logistic Regression for Machine Learning - MachineLearningMastery.com

[3] 1.4. Support Vector Machines — scikit-learn 1.6.0 documentation

[4] Ensemble Modeling Tutorial | Explore Ensemble Learning Techniques | DataCamp