Electrical and Computer Engineering Department

Machine Learning and Data Science - ENCS5341

Assignment #3

Rana Musa 1210007

Leyan Burait 1211439

Dr. Ismail Khater

SEC:#3

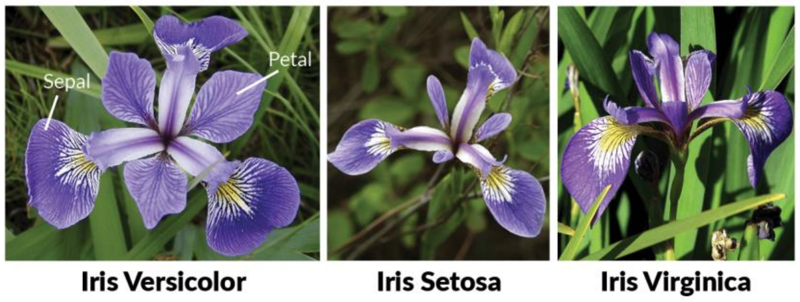
Contents

1. Introduction

1.1 Dataset

In our assignment, we utilized the Iris Dataset, a classic dataset in the realm of machine learning, originally introduced by Ronald A. Fisher in 1936. This dataset consists of 150 records of iris flowers, with each sample characterized by four features: Sepal Length, Sepal Width, Petal Length, and Petal Width (all measured in centimeters). The iris samples are categorized into three distinct species: Iris-setosa, Iris-versicolor, and Iris-virginica. Due to its clean and balanced composition, the Iris Dataset serves as an ideal choice for a variety of tasks, including classification, clustering, and demonstrations of machine learning algorithms. Its widespread use in educational settings makes it a valuable resource for exploring fundamental concepts in data analysis and model training.

https://www.kaggle.com/datasets/uciml/iris



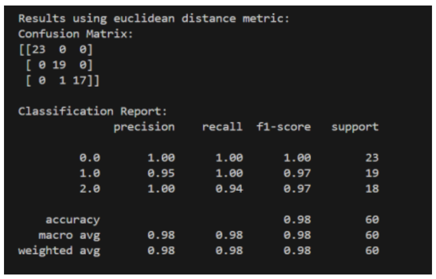
2.Discussion:

2.1. K-Nearest Neighbors (KNN)

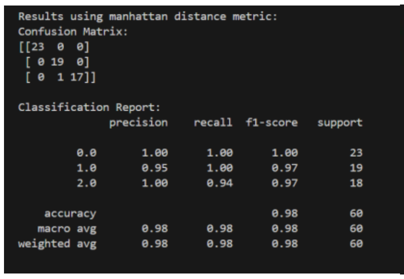
K-Nearest Neighbors (KNN) is a straightforward supervised machine learning algorithm used for classification and regression tasks. It operates on the principle that similar data points are close to each other in feature space. When predicting a label for a new data point, KNN identifies the (K) closest training examples and assigns the label based on the majority class among those neighbors.

The process begins with loading and preprocessing the Iris dataset, which includes handling missing values, encoding the Species column, and scaling the features using StandardScaler. The KNN algorithm is then trained with various distance metrics, including Euclidean, Manhattan, and Cosine, to assess their performance. The dataset is divided into training (60%) and testing (40%) sets, and model evaluation is conducted using a confusion matrix and classification report. To identify the optimal value of K, 5-fold cross-validation is performed for K values ranging from 1 to 20, with the best K selected based on the highest cross-validation accuracy. Finally, results are visualized to enhance interpretation.

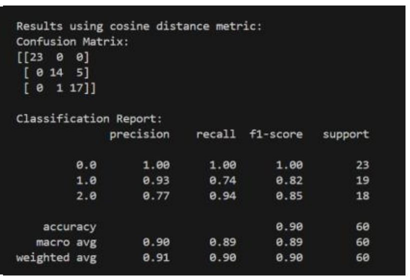
2.1.1. Results



```
Results using euclidean distance metric:
Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.95      1.00      0.97        19
         2.0       1.00      0.94      0.97        18

    accuracy                           0.98        60
   macro avg       0.98      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60
```



```
Results using manhattan distance metric:
Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.95      1.00      0.97        19
         2.0       1.00      0.94      0.97        18

    accuracy                           0.98        60
   macro avg       0.98      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60
```

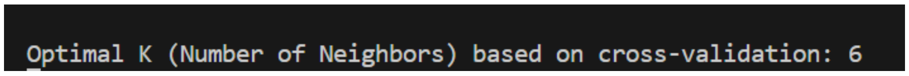


```
Results using cosine distance metric:
Confusion Matrix:
[[23  0  0]
 [ 0 14  5]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.93      0.74      0.82        19
         2.0       0.77      0.94      0.85        18

    accuracy                           0.90        60
   macro avg       0.90      0.89      0.89        60
weighted avg       0.91      0.90      0.90        60
```

The **Euclidean distance** metric performs very well, achieving an overall accuracy of **98%**. The confusion matrix shows that both Class 0 and Class 1 were predicted correctly without any mistakes, while only one instance of Class 2 was wrongly classified as Class 1. This means that the Euclidean distance does a great job of separating the classes, resulting in high precision and recall for all of them.

The **Manhattan distance** metric yields the same results as the Euclidean distance, also achieving **98%** accuracy with nearly perfect precision, recall, and F1-scores. Both metrics appear to be equally effective for the Iris dataset, as they recognize patterns in the feature space in a similar way.

The **Cosine distance** metric performs worse than both the Euclidean and Manhattan distances, achieving an accuracy of **90%**. The confusion matrix shows some misclassifications: 5 instances of Class 1 are incorrectly labeled as Class 2, and 1 instance of Class 2 is misclassified as Class 1. As a result, Class 1 has a lower recall of **74%**, and Class 2 has a precision of only **77%**. Since Cosine distance measures angular similarity, it may not effectively capture the relationships present in the Iris dataset, which depends more on distances than on angular alignment.



```
Optimal K (Number of Neighbors) based on cross-validation: 6
```

The optimal value of K (the number of neighbors) for the K-Nearest Neighbors (KNN) algorithm was identified through 5-fold cross-validation. After testing K values from 1 to 20, the model reached the highest cross-validation accuracy at **K = 6**. This suggests that using 6 neighbors provides the best balance between underfitting (with a low K) and overfitting (with a high K), leading to better model generalization. With K set to 6, the algorithm effectively utilizes the local structure of the data for accurate predictions.

In conclusion, the KNN algorithm showed outstanding performance with both Euclidean and Manhattan distance metrics, achieving **98%** accuracy with perfect classification for Classes 0 and 1. In contrast, the Cosine distance metric resulted in lower accuracy of **90%** and significant misclassifications, particularly between Classes 1 and 2. This indicates that distance-based metrics are more suitable for the Iris dataset, as they effectively capture the relationships among species. Choosing the right distance metric is crucial for optimal classification performance.

2.2. Logistic Regression

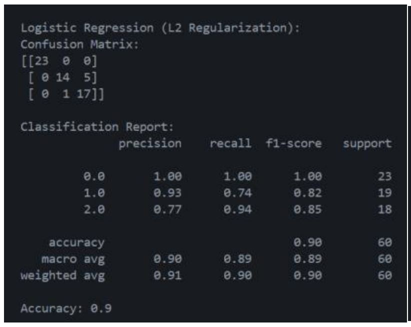
Logistic Regression: This is a supervised learning method used for classification tasks. It predicts the likelihood of an item belonging to a certain class by using a logistic (sigmoid) function, which helps it make binary predictions.

L1 Regularization (Lasso): This technique improves the model by adding the absolute values of the weights to the loss function. It encourages some weights to become zero, which simplifies the model by selecting only the most important features.

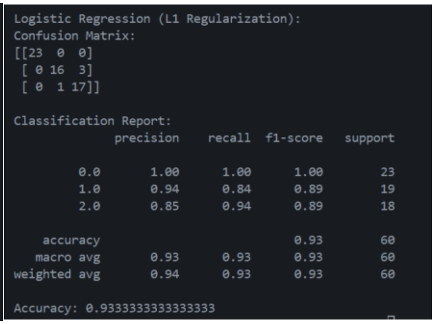
L2 Regularization (Ridge): Ridge Regularization adds the square of the weights to the loss function. This helps prevent the model from becoming too complex (overfitting) by discouraging very large weights, making the model more stable and better at predicting new data.

The approach to experimenting with the Logistic Regression algorithm begins with data preparation, where the dataset is cleaned by converting numeric columns, filling missing values with column means, and encoding the target variable "Species" into numeric form. Next, the dataset is split into training (60%) and testing (40%) sets to evaluate the model's performance on unseen data. To ensure that all features contribute equally, StandardScaler is used to standardize the features. The Logistic Regression model is then trained using both L1 (Lasso) and L2 (Ridge) regularization techniques with the liblinear solver. Finally, the model's performance is assessed using key metrics such as accuracy, confusion matrix, and the classification report to compare the effectiveness of the two regularization methods

2.2.1. Results



```
Logistic Regression (L2 Regularization):
Confusion Matrix:
[[23  0  0]
 [ 0 14  5]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.93      0.74      0.82        19
         2.0       0.77      0.94      0.85        18

    accuracy                           0.90        60
   macro avg       0.90      0.89      0.89        60
weighted avg       0.91      0.90      0.90        60

Accuracy: 0.9
```



```
Logistic Regression (L1 Regularization):
Confusion Matrix:
[[23  0  0]
 [ 0 16  3]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.94      0.84      0.89        19
         2.0       0.85      0.94      0.89        18

    accuracy                           0.93        60
   macro avg       0.93      0.93      0.93        60
weighted avg       0.94      0.93      0.93        60

Accuracy: 0.9333333333333333
```

As shown above figure **Logistic Regression with L1** regularization performs better **than L2 regularization**

2.2.2. Comparison of Logistic Regression and KNN

KNN outperforms Logistic Regression in terms of accuracy and overall classification performance on this dataset. KNN achieves a consistent **98% accuracy** using both **Euclidean** and **Manhattan distance metrics**, with high precision, recall, and F1-scores across all classes. In contrast, Logistic Regression achieves **93.33% accuracy** with L1 regularization and **90% accuracy** with L2 regularization, showing a slight drop in performance, particularly for **Class 1**. While Logistic

Regression performs well, especially with L1 regularization, KNN demonstrates superior robustness and precision, making it the better-performing model for this dataset.

2.3. Support Vector Machines (SVM)

Support Vector Machine (SVM): This is a supervised learning algorithm mainly used for classification tasks. It works by finding the best boundary that separates different classes in the data.

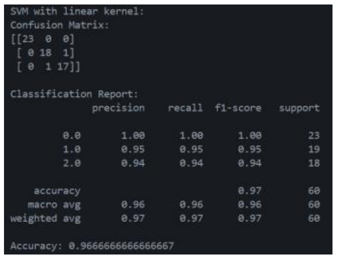
Linear Kernel: This kernel classifies data by identifying a straight line (or hyperplane) that effectively divides the classes. It's ideal for linearly separable data.

Polynomial Kernel: This kernel employs a polynomial function to manage non-linear relationships, allowing the model to capture more complex patterns in the data.

RBF Kernel (Radial Basis Function): The RBF kernel is designed to handle complex, non-linear relationships by transforming the data into higher dimensions. This enables the SVM to identify intricate patterns that would be difficult to separate in the original feature space.

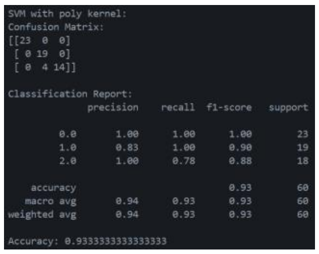
The dataset is divided into training (60%) and testing (40%) sets, with features standardized using StandardScaler to ensure they contribute equally. The SVM model is trained with three different kernels—linear, polynomial, and RBF—to assess their performance on the data. Model evaluation is conducted using essential metrics, including the confusion matrix, classification report, and accuracy score, for effective comparison.

2.3.1 Results



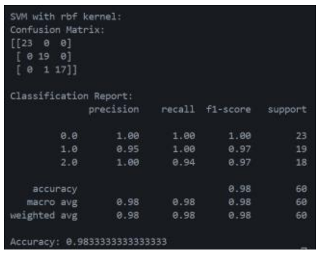
```
SVM with linear kernel:
Confusion Matrix:
[[23  0  0]
 [ 0 18  1]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.95      0.95      0.95        19
         2.0       0.94      0.94      0.94        18

    accuracy                           0.97        60
   macro avg       0.96      0.96      0.96        60
weighted avg       0.97      0.97      0.97        60

Accuracy: 0.9666666666666667
```

Figure 1: SVM with linear kernel



```
SVM with poly kernel:
Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  4 14]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.83      1.00      0.90        19
         2.0       1.00      0.78      0.88        18

    accuracy                           0.93        60
   macro avg       0.94      0.93      0.93        60
weighted avg       0.94      0.93      0.93        60

Accuracy: 0.9333333333333333
```

Figure 2:SVM WITH POLY kereal



```
SVM with rbf kernel:
Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.95      1.00      0.97        19
         2.0       1.00      0.94      0.97        18

    accuracy                           0.98        60
   macro avg       0.98      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60

Accuracy: 0.9833333333333333
```

Figure 3: SVM with rbf kernel

As shown above the performance of Support Vector Machines (SVM) was evaluated using three kernels: **linear**, **polynomial**, and **radial basis function (RBF)**, and their results were compared using classification metrics. The **linear kernel** achieved an accuracy of **96.67%**, performing well for linearly separable classes but slightly misclassifying instances from Class 1 and Class 2, indicating some non-linear patterns in the data. The **polynomial kernel** had a lower accuracy of **93.33%**, as it struggled to generalize for Class 2, leading to a noticeable drop in recall (78%) despite strong performance for the other classes. In contrast, the **RBF kernel** outperformed both, achieving the highest accuracy of **98.33%** with near-perfect precision, recall, and F1-scores for all classes, demonstrating its ability to handle complex, non-linear decision boundaries effectively.

These results highlight that while the linear kernel works well for simpler datasets, and the polynomial kernel can introduce some non-linearity, the **RBF kernel** is the most effective for this dataset due to its flexibility in capturing intricate patterns and achieving superior classification performance.

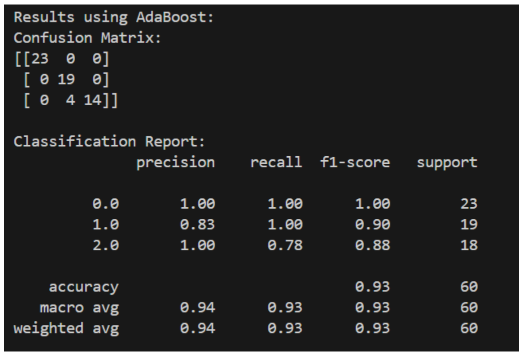
2.4. Ensemble Methods

AdaBoost (Adaptive Boosting): AdaBoost is an ensemble learning technique that combines several weak classifiers, such as decision trees, to form a strong classifier. It enhances performance by assigning greater weights to misclassified instances, thereby concentrating on the more challenging samples in each iteration.

DecisionTreeClassifier: This serves as the base estimator in AdaBoost. It creates a tree-like model of decisions based on the features, dividing the dataset into subsets according to feature values. While it can be prone to overfitting, it works effectively with boosting methods like AdaBoost to enhance overall generalization.

An **AdaBoost model** is trained with a **DecisionTreeClassifier** as the base estimator, using 50 estimators to create a strong classifier. The model's performance is evaluated using the **confusion matrix** and **classification report** for insights into the model's accuracy, precision, recall, and F1-score across different classes

2.4.1. Results



```
Results using AdaBoost:
Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  4 14]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.83      1.00      0.90        19
         2.0       1.00      0.78      0.88        18

    accuracy                           0.93        60
   macro avg       0.94      0.93      0.93        60
weighted avg       0.94      0.93      0.93        60
```

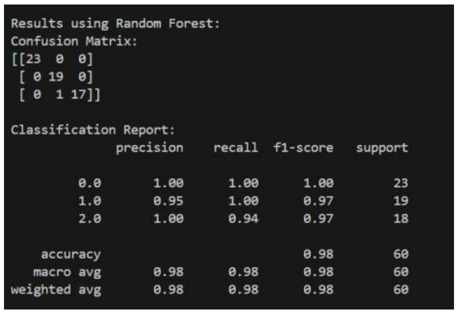
AdaBoost achieved an accuracy of **93%**, demonstrating outstanding performance in identifying Iris-setosa (class 0), where precision, recall, and F1-score were all 1.00. However, some misclassifications occurred between Iris-versicolor (class 1) and Iris-virginica (class 2), impacting the precision and recall for these classes. Class 1 exhibited **83% precision** and **100% recall**, while class 2 achieved perfect precision but a lower recall of 78%. Overall, the model's performance is robust, with macro and weighted averages of 0.94 and 0.93, respectively.

Bagging (Bootstrap Aggregating): Bagging is an ensemble technique that involves training multiple base models, typically decision trees, on different bootstrapped subsets of the training data. This approach enhances model performance by reducing variance and mitigating overfitting through the averaging of predictions. The BaggingClassifier in scikit-learn effectively implements this concept.

Random Forest: Random Forest builds upon the bagging method by creating an ensemble of decision trees, each trained on random subsets of the data and random features. This strategy not only boosts model accuracy but also helps to reduce overfitting by averaging the predictions from individual trees, leading to a more robust and reliable model.

The RandomForestClassifier is trained using 100 estimators to evaluate its performance. Model predictions are compared against the test data, and results are evaluated using confusion matrix and classification report.

2.4.2. Results



```
Results using Random Forest:
Confusion Matrix:
[[23  0  0]
 [ 0 19  0]
 [ 0  1 17]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        23
         1.0       0.95      1.00      0.97        19
         2.0       1.00      0.94      0.97        18

    accuracy                           0.98        60
   macro avg       0.98      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60
```

The Random Forest model achieves a high accuracy of **98%** with excellent performance across all classes. The confusion matrix shows correct classifications for most instances, with one misclassification in **Iris-virginica**. Precision, recall, and F1-scores are all high, particularly for **Iris-setosa** and **Iris-virginica**. The model's weighted and macro averages are consistently strong, indicating balanced performance. In conclusion, Random Forest is highly effective for this classification task, offering both precision and reliability.

Based on the results, Bagging (Random Forest) outperformed Boosting (AdaBoost). The Random Forest model achieved an impressive accuracy of **98%**, demonstrating superior performance across all classes, while AdaBoost attained a lower accuracy of **93%**. Random Forest's effectiveness in handling class imbalances and delivering higher precision and recall for each class contributed to its better performance. In contrast, AdaBoost faced challenges with Class 2, leading to its diminished overall accuracy.

When comparing ensemble methods to individual models like KNN, Logistic Regression, and SVM, ensemble techniques generally yield better results. Although individual models can be effective, they are often susceptible to overfitting or underfitting based on data complexity. Ensemble methods, like Random Forest (Bagging) and AdaBoost (Boosting), combine multiple models to minimize variance and bias, resulting in improved generalization and more robust predictions. Specifically, Bagging helps reduce overfitting, while Boosting focuses on correcting the errors of previous models. Consequently, ensemble methods typically provide enhanced performance and stability, particularly in complex or noisy datasets.

Group Work

Rana Musa 1210007	K-Nearest Neighbors (KNN) + Ensemble Methods and report for those parts
Leyan Burait 1211439	Logistic Regression + Support Vector Machines (SVM) and report for those parts