Iris Analysis

Madawi Bin Shawiah - Yasmin Al sulaiman - Rafaa

Introduction

The Iris Analysis Project aims to carry out two main steps including supervised and unsupervised machine learning tasks employing the well–known Iris dataset. The Iris Dataset is one of the most known datasets consisting of 150 samples of a flower each characterized by four feature domains namely sepal length, sepal width, petal length, and petal width with the target variable indicating either the iris species (setosa, versicolor, or virginica).

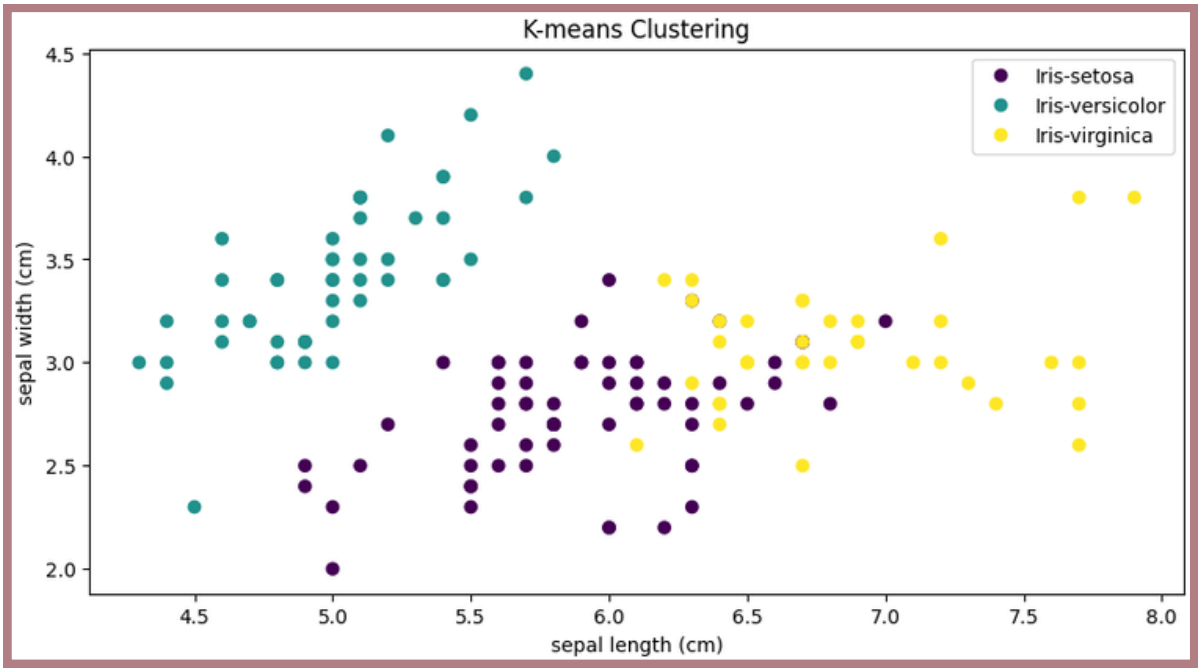
Throughout this project, I performed several tasks:

1. **Unsupervised Learning: we applied K-means clustering to partition samples that are alike to each other and integrate outlier detection techniques for detecting abnormal samples.**
2. **Supervised Learning: we did a performance evaluation, and after this, we split the dataset into a training data set and a testing data set.**
3. **Model Comparison: doing this, we applied and tested different machine learning algorithms, ranking their effectiveness using 10-fold cross-validation approaches.**
4. **Model Tuning and Ensemble: we fine-tuned the optimal algorithm leveraging hyperparameter tuning and we developed an ensemble of the models that performed best.**

Unsupervised Learning

This section explains the unsupervised learning methods through the Iris dataset, including clustering with K-means and outlier detection with DBSCAN.

1. K-means Clustering: The K-means algorithm has been used in this analysis to identify a set of closed natural clusters in the Iris dataset using feature values. It comprises the steps of allocating the data data points to the nearest centroid and updating them until convergence. Visualization techniques, for example, scatter plots are applied to depict the clusters.



2.Outlier Detection: DBSCAN, a density-based outlier detection technique, was used in the dataset to detect outliers in it. It selects areas of high concentration and views the data outside these regions as the outliers. Parameters such as a minimum number of points and epsilon were clarified to decide about cluster generation and the outlier detection.

3. Evaluation: The clustering was evaluated using metrics such as the Silhouette Score. DBSCAN-detected outliers being verified for authenticity, and possible misclassification or mistake were also considered.

```
[ ]  n_clusters=len(set(dbs.labels_))-(1 if -1 in dbs.labels_ else 0)
     outlier=list(dbs.labels_).count(-1)
     print("CLUSTERS NUMBER: %d" % n_clusters)
     print("OUTLIRE: %d" % outlier)

     CLUSTERS NUMBER: 3
     OUTLIRE: 96
```

Supervised learning

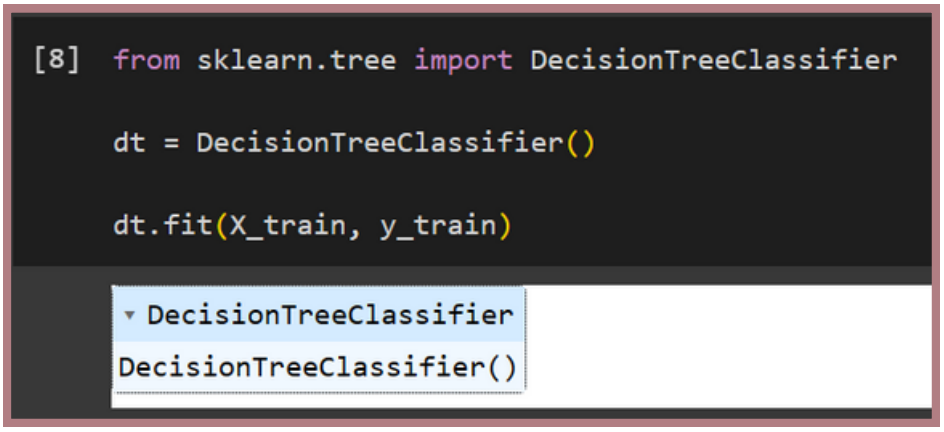
In the supervised learning section, a baseline model was developed and evaluated for the Iris dataset.

1. **Dataset Splitting: The dataset was divided into training and testing sets. This division ensures that the model is trained on a portion of the data and evaluated on unseen data. In this case, the training set was set to 80% of the data, while the testing set comprised the remaining 20%.**

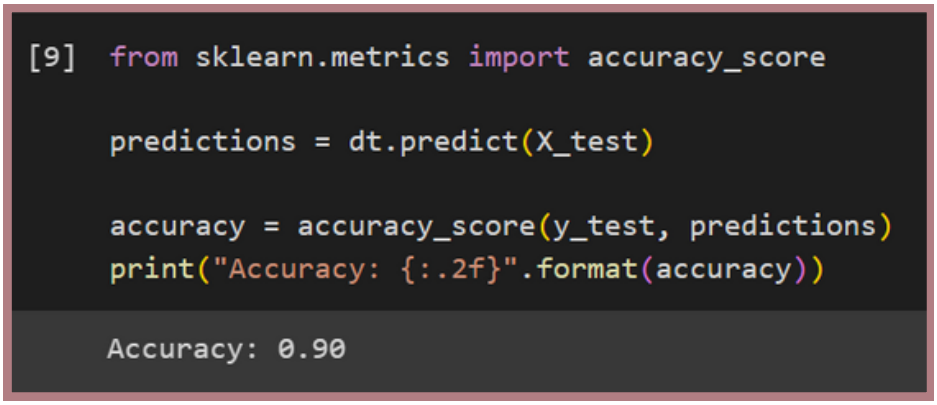
Split the dataset into training and testing sets.

```
[7]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2. Baseline Model Implementation: The baseline model was implemented using a decision tree classifier algorithm. Decision tree classifiers are a popular choice for classification tasks and are known for their interpretability.



3. Performance Evaluation: The performance of the baseline decision tree classifier model was evaluated using the accuracy. The model achieved an accuracy of 90% on the Iris dataset.



Model Comparison

In order to identify the best-performing algorithm for the Iris dataset classification problem, a model comparison was conducted:

1. **Selection of Machine Learning Algorithms: In the model comparison phase, three machine learning algorithms were selected for evaluation on the Iris dataset classification problem. The chosen algorithms were Support Vector Machines (SVM), RandomForestClassifier, and GradientBoostingClassifier.**

2. **Implementation and Cross-Validation: Each selected algorithm was implemented and applied to the Iris dataset. To ensure robust evaluation, cross-validation was employed.**

```
Support Vector Machines Accuracy for each fold:
Fold 1: 0.96
Fold 2: 1.00
Fold 3: 0.92
Fold 4: 1.00
Fold 5: 1.00
Average Accuracy: 0.97

Random Forest Accuracy for each fold:
Fold 1: 0.96
Fold 2: 1.00
Fold 3: 0.92
Fold 4: 0.96
Fold 5: 1.00
Average Accuracy: 0.97

Gradient Boosting Accuracy for each fold:
Fold 1: 0.96
Fold 2: 1.00
Fold 3: 0.92
Fold 4: 0.92
Fold 5: 1.00
Average Accuracy: 0.96
```

3. Performance Comparison: The performance of each algorithm was compared based on the chosen evaluation metrics. here we see all algorithms have same accuracy that's because of the balanced data and it's simple.

```
Evaluation Metrics for SVM Classifier
Accuracy: 0.93
Precision: 0.92
Recall: 0.94
F1-Score: 0.92

Evaluation Metrics for Random Forest Classifier
Accuracy: 0.93
Precision: 0.92
Recall: 0.94
F1-Score: 0.92

Evaluation Metrics for Gradient Boosting Classifier
Accuracy: 0.93
Precision: 0.92
Recall: 0.94
F1-Score: 0.92
```

4. Selection of the Best-Performing Algorithm:

```
accuracy_scores = [svm_accuracy, rf_accuracy, gb_accuracy]
best_accuracy = max(accuracy_scores)

if best_accuracy == svm_accuracy:
    best_algorithm = "Support Vector Machines"
elif best_accuracy == rf_accuracy:
    best_algorithm = "Random Forest"
else:
    best_algorithm = "Gradient Boosting"

print("Best performing algorithm based on accuracy:" , best_algorithm)
```

```
Best performing algorithm based on accuracy: Support Vector Machines
```

Model Tuning and Ensemble

1. **In the model tuning and ensemble phase, the best-performing algorithm was refined through hyperparameter tuning using Grid Search.**

```
from sklearn.model_selection import GridSearchCV

svm = SVC()
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': [0.1, 1, 10]
}
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_svm = grid_search.best_estimator_
```

2. Performance Evaluation: Once the hyperparameters were tuned using Grid Search, the performance of the tuned model was evaluated.

```
svm_predictions = best_svm.predict(X_test)
svm_f1_scoreaccuracy = accuracy_score(y_test, svm_predictions)

print("Accuracy for Tuned SVM Model:")
print("Accuracy: {:.2f}".format(svm_accuracy*100),"%")
```

```
Accuracy for Tuned SVM Model:
Accuracy: 93.33 %
```

3. Ensemble Implementation: In addition to tuning the best-performing algorithm, an ensemble of the top-performing algorithms was implemented.

```
from sklearn.ensemble import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier

model1 = RandomForestClassifier(n_estimators=100)
model2 = GradientBoostingClassifier(n_estimators=100)
model3 = KNeighborsClassifier(n_neighbors=3)


top_model = RandomForestClassifier(n_estimators=100)

stacking_model = StackingClassifier(
    estimators=[('rf', model1), ('gb', model2), ('knn', model3)],
    final_estimator=top_model
)

stacking_model.fit(X_train, y_train)
y_pred = stacking_model.predict(X_test)

st_accuracy = accuracy_score(y_test, y_pred)

print("Accuracy ensemble Model:")
print("Accuracy: {:.2f}".format(st_accuracy*100),"%")
```

```
Accuracy ensemble Model:
Accuracy: 93.33 %
```

4. Performance Comparison: the performance of the tuned model and the ensemble model was compared using the evaluation metric of accuracy. Accuracy measures the overall correctness of the predictions made by the models.

```
if svm_accuracy > st_accuracy:
    print("svm is higher than staking.")
elif svm_accuracy < st_accuracy:
    print("svm is lower than staking.")
else:
    print("svm is equal to staking.")
```

```
svm is higher than staking.
```

Conclusion

In this project, we analyzed the Iris dataset, a well-known dataset. The project involved several tasks, including data preprocessing, unsupervised learning, supervised learning, model comparison, model tuning, and ensemble modeling.
The project report concluded with a summary of the key findings and insights. It discussed the strengths and limitations of the implemented models and provided recommendations for further improvement. Overall, the project provided a comprehensive analysis of the Iris dataset and demonstrated the effectiveness of various machine learning techniques.