# Detailed Report on Predicting Iris Species Using Supervised Learning

## 1. Problem Statement

The objective of this analysis is to predict the species of Iris flowers based on their feature. It contains measurements of Iris flowers classified into three species: Setosa, Versicolor, and Virginica.

The Iris dataset is a well known dataset. It contains 150 samples of iris flower with four features each sample; sepal length, sepal width, petal length, and petal with all measured in centimeters (cm).

## 2. Data Preprocessing

The dataset is loaded using the **load_iris()** function from **sklearn.datasets.** The dataset is split into training and testing sets. Splitting ensures each class is represented in both sets. Feature scaling is performed using **StandardScaler()** this is important for the 2 algorithms I used which are Decision Trees, and Random Forests.

## 3. Model Training

The 2 learning algorithms I used are Decision Tree Classifier & Random Forest Classifier. Training and Hyper-parameter tuning of the Decision Tree Classifier is initialized and trained on the scaled training data. Hyper-parameter tuning is performed using **GridSearchSV** to find values for **max_depth** & **min_samples_split**.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_scaled, y_train)  # Train the model
param_grid_dt = {'max_depth': [None, 2, 4, 6], 'min_samples_split': [2, 5, 10]}
grid_dt = GridSearchCV(dt, param_grid_dt, cv=5)  # Hyperparameter tuning
grid_dt.fit(X_train_scaled, y_train)
best_dt = grid_dt.best_estimator_
```

The Random Forest Classifier is trained similarly with hyper-parameter tuning for **n_estimators** and **max_depth**.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_scaled, y_train)  # Train the model
param_grid_rf = {'n_estimators': [50, 100, 150], 'max_depth': [None, 2, 4, 6]}
```

```
grid_rf = GridSearchCV(rf, param_grid_rf, cv=5)  # Hyperparameter tuning
grid_rf.fit(X_train_scaled, y_train)
best_rf = grid_rf.best_estimator_
```

## 4. Model Evaluation

The models are evaluated using accuracy and classification reports, which include precision, recall, and F1-score for each class here is a code snippet:

```
from sklearn.metrics import accuracy_score, classification_report

y_pred_dt_best = best_dt.predict(X_test_scaled)  # Predictions with best Decision Tree model
y_pred_rf_best = best_rf.predict(X_test_scaled)  # Predictions with best Random Forest model

print("Decision Tree Performance:")
print(classification_report(y_test, y_pred_dt_best))

print("\nRandom Forest Performance:")
print(classification_report(y_test, y_pred_rf_best))
```

### Data Preprocessing Code Snippet

```
# Load the Iris dataset
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# Data Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

### Model Training and Evaluation Code Snippet

```
# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_scaled, y_train)
y_pred_dt_best = best_dt.predict(X_test_scaled)

# Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_scaled, y_train)
```

```
y_pred_rf_best = best_rf.predict(X_test_scaled)

# Evaluation
print("Decision Tree Performance:")
print(classification_report(y_test, y_pred_dt_best))

print("\nRandom Forest Performance:")
print(classification_report(y_test, y_pred_rf_best))
```

## 5. Comparative Analysis

The performance of both models is compared based on the accuracy and classification reports. Decision Tree Accuracy: Generally shows a good understanding of the dataset but may be prone to overfitting, especially with deeper trees. Random Forest Accuracy: Typically achieves higher accuracy due to the ensemble nature of the model, which averages multiple decision trees, thereby reducing overfitting. Strengths and Weaknesses Decision Tree: Strengths: Easy to interpret, visual representation, requires no feature scaling. Weaknesses: Prone to overfitting, sensitive to data variability. Random Forest: Strengths: More robust to overfitting, handles a larger number of features well, provides feature importance. Weaknesses: Less interpretable than a single decision tree, longer training time.

## 6. Conclusion

Conclusions The Random Forest model outperformed the Decision Tree model in terms of accuracy and generalization. The analysis highlighted the effectiveness of ensemble methods in handling complex datasets. Future Work Experimenting with Other Models: Testing additional algorithms such as Support Vector Machines (SVM) or Gradient Boosting to compare performance. Further Hyperparameter Tuning: Investigating more hyperparameters or using different tuning techniques such as Random Search. Feature Engineering: Exploring new feature combinations or dimensionality reduction techniques like PCA. Real-World Applications: Applying the model to real-world datasets or live prediction scenarios to assess performance in practical settings.

## Link to Github Repository

https://github.com/JefferyDirden/Module-2-Real-world-Application-of-Supervised-Learning