UNIFIED MENTOR

YOUR SKILL, SUCCESS & JOURNEY

A Project Report

By

Alladi Nikhila Sarea

Project Title	Iris Classification
Technologies	Machine learning
Domain	Data Analytics
Contribution	Individual

Dataset: Dataset

To access the code, here is the link for Google Colab: Colab

GitHub Link: Github

Introduction

The Iris Flower – A Symbol of Beauty and Grace

The iris flower is one of nature's most elegant and colorful blooms, admired for its intricate petals and vibrant hues. Belonging to the genus *Iris*, which includes over 300 species, this flower derives its name from the Greek word for "rainbow," reflecting the wide range of colors it exhibits—purple, blue, yellow, white, and more.

Native to Europe, the Middle East, and northern Africa, iris flowers are known for their distinct petal arrangement: three upright petals (standards) and three hanging petals (falls). They thrive in temperate climates and can often be found in gardens, wetlands, and meadows. Irises are not only prized for their beauty but also for their symbolism. In various cultures, they represent faith, wisdom, courage, and hope.

Beyond their aesthetic value, some iris species are used in perfumery and herbal remedies. In scientific research, the iris flower gained fame as the subject of the famous *Iris dataset*, which is widely used in machine learning and data science.

In essence, the iris flower is more than just a beautiful bloom—it is a botanical marvel with cultural, historical, and scientific significance.

Project Description

The Iris Classification project focuses on developing a machine learning model that can identify the species of iris flowers—Setosa, Versicolour, or Virginica—by analyzing the measurements of their petal and sepal lengths and widths. This project showcases the application of machine learning techniques to a well-known classification problem, offering insights into the distinct features of various iris species and evaluating the performance of different algorithms in solving this task.

1. **Iris Dataset**: It contains iris flowers from three specific species—Setosa, Versicolor, and Virginica.
2. **Features**: The primary attributes used for classification are the lengths and widths of the sepals and petals.
3. **ML model**: The project focuses on building and training a machine learning model to effectively classify the iris species based on these features.
4. **Output**: The developed and trained model will be able to distinguish between the three types based on their measurements.

Project Objective

The Iris Flower Classification Project is a supervised machine learning model where it focuses on the classification of the three different types of species of flowers - *Setosa*, *Versicolor*, and *Virginica*. These three are differentiated based on their sepal and petal lengths which have been classified in the given data set. A total of 150 samples have been given in the dataset which are used to train and test the model.

Project steps:

i. The required libraries are imported along with the dataset

ii. Data Visualization is done to understand the relationships between the species

iii. Data is then split into *training data (60%)* and *testing data (40%)*.

iv. Decision trees and SVM and a few more are implemented for the model

v. Model evaluation is performed on the testing data and performance is evaluated from metrics such as *accuracy*, *precision*, *recall*, and *F1-score*.

vi. *Confusion Matrix* is visualized using heat map to understand the classification results in detail.

vii. Grid search technique is used to find the optimal hyperparameters for the chosen model.

Task 1

Import the necessary libraries

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

The libraries imported are explained below:

i. **NumPy (np)**: Fundamental package for numerical computing in Python. Used for arrays, mathematical functions, linear algebra, etc.
ii. **Pandas (pd)**: Essential for data manipulation and analysis. Provides Data frames and Series for structured data.
iii. **PCA (Principal Component Analysis)**: A technique to reduce the dimensionality of data while retaining as much variance as possible. It helps in speeding up machine learning algorithms and visualizing high-dimensional data.
iv. **Matplotlib (plt)**: The most commonly used library for 2D plotting in Python.
v. **Seaborn (sns)**: Built on top of matplotlib, used for making statistical graphics with better aesthetics.
vi. **%matplotlib inline**: A Jupyter Notebook magic command to display plots directly in the notebook.
vii. **train_test_split**: Splits dataset into training and testing subsets.
viii. **GridSearchCV**: Used for hyperparameter tuning by searching over specified parameter values with cross-validation.
ix. **Decision Tree Classifier**: A tree-structured classifier that splits data based on feature values.
x. **Random Forest Classifier**: An ensemble method that builds multiple decision trees and merges their results.
xi. **SVC (Support Vector Classifier)**: A powerful classifier for linear and non-linear problems.
xii. **K Neighbors Classifier**: A simple algorithm that classifies data based on the closest data points.
xiii. **Logistic Regression**: A linear model used for binary classification tasks.

xiv. **Classification report**: Displays precision, recall, F1-score, and support.
xv. **Accuracy score**: Measures the ratio of correctly predicted samples.
xvi. **Confusion matrix**: Summarizes true positives, false positives, true negatives, and false negatives.
xvii. **Standard Scaler**: Standardizes features by removing the mean and scaling to unit variance.
xviii. **Label Encoder**: Converts categorical labels into numerical form.

Load the dataset

```
df = pd.read_csv(r"Iris.csv")
```

The dataset has been imported for the model. From this dataset the training and testing sets are going to be prepared for the development of the model.

Task 2

Data Visualization

The primary goal of data visualization in this project is to:

- Understand feature distributions.
- Identify patterns and relationships.
- Spot class separation between the three Iris species (*Setosa*, *Versicolor*, *Virginica*).
- Support feature selection and modeling decisions.

The dataset contains a list of different species of Iris Flowers differentiated from the length and width of the petals and sepals. The initial shape of the dataset is (150, 5). There are 3 duplicates found in the dataset. The final dataset after removing duplicates consists of 147 rows and 5 columns. The final duplicate values and null values are 0.

From the **count plot** of the species, it can be seen that the species versicolor has the maximum number of samples in the dataset after cleaning.

Looking at the **pair plots**, we can see that throughout all the 4 plots, the Setosa has been clearly separated from the other species. Versicolor and Virginica show some overlap, especially in sepal measurements. Petal length vs. petal width is the most effective pair for distinguishing all three species. This plot gives an early indication that petal features are more informative than sepal features for classification.

From the **Distribution Curves**, Setosa has distinctly smaller petal lengths, with no overlap with other classes. Virginica has the longest petals. Distributions are visually separable, indicating that

petal length alone can contribute significantly to classification. All the three species overlap together in the sepal width distribution indicating it does not contribute much in the classification.

Summary on visualizations

Petal Length vs Width	High	Very High
Sepal Length vs Width	Low	Limited
Petal Length Distribution	High	Useful for single-feature models

Gini Impurity

Gini impurity is a measure of how often a randomly chosen element would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the dataset.

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

Where, p_i is the probability of class i.

It is a splitting criterion used in Decision Trees. A lower Gini value indicates a purer node (better class separation). Gini impurity helps understand how mixed the classes are in the target variable. It guides the decision tree in choosing the best features to split on.

In this case, Gini impurity of found to be 0.6665. It means the dataset is equally distributed among the three classes, which is expected for the Iris dataset (approx. 50 samples per class).

Task 3

Data Preprocessing and Classification Model

Data is split into training and testing data

```
x_train,x_test,y_train,y_test = train_test_split(x_std, y_new, test_size=0.2, random_state=42)
```

Four different datasets have been prepared. 2 of them being the training sets and the other 2 the testing sets with ‘x’ being the Feature matrices (input) and ‘y’ being the Target vectors (output).

Function of the Classification Model

The "**classification_model()**" function is a unified evaluation utility designed to:

- Train a given classification model.
- Evaluate its performance on both training and test sets.
- Generate and display performance metrics, classification reports, and confusion matrices.
- Return a list of essential performance metrics for further comparison or logging.

The inputs of the function would be

- **name**: a string used for naming the model.
- **model**: Any classification model object (e.g., SVC(), DecisionTreeClassifier(), etc.)
- **x_train, x_test**: Feature matrices for training and testing.
- **y_train, y_test**: Target vectors for training and testing.

```
def classification_model(name,model,x_train,x_test,y_train,y_test):
    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
```

- ✓ Trains the model on the training dataset.
- ✓ Predicts labels for both training and test data.
- ✓ This allows for comparing model performance on seen vs. unseen data (overfitting check).

```
accuracy_train = accuracy_score(y_train,y_pred_train)
accuracy_test = accuracy_score(y_test,y_pred_test)

print(f"accuracy score for {name} training data: {accuracy_train:.2f}")
print(f"accuracy score for {name} testing data: {accuracy_test:.2f}")

print(f"classification report for {name} training set: ")
cr_train = classification_report(y_train, y_pred_train, output_dict=True)
print(classification_report(y_train, y_pred_train))
print(f"classification report for {name} testing set: ")
cr_test = classification_report(y_test, y_pred_test, output_dict=True)
print(classification_report(y_test, y_pred_test))
```

- ✓ Computes accuracy (correct predictions / total predictions) on both datasets.
- ✓ High training but low testing accuracy → overfitting.
- ✓ Similar accuracy → good generalization.
- ✓ Displays accuracy with two decimal precision.

- ✓ Labels the results with the model's name.
- ✓ Computes metrics like **precision**, **recall**, and **F1-score** for each class.
- ✓ output_dict=True enables structured access to these metrics for logging.
- ✓ Helps evaluate model performance class-wise and on overall distribution.

```
con_mat_train = confusion_matrix(y_train, y_pred_train)
con_mat_test = confusion_matrix(y_test, y_pred_test)
```

A **confusion matrix** shows counts of:

- ➢ True Positives
- ➢ True Negatives
- ➢ False Positives
- ➢ False Negatives

Useful for diagnosing which classes are being confused.

```
class_names = y
le = LabelEncoder()
le.fit(y)
class_names_encoded = le.transform(y)
class_names = le.classes_
```

- ✓ Retrieves class labels from original target y.
- ✓ Uses LabelEncoder to get a sorted list of class names for display on the confusion matrix axes.

```
figure, f = plt.subplots(1,2,figsize=(12,4))
print("Confusion matrix for ",name)

sns.heatmap(con_mat_train, annot = True, fmt="d", cmap="YlGnBu", xticklabels = class_names, yticklabe
f[0].set_xlabel("Predicted Class")
f[0].set_ylabel("True Class")
f[0].set_title("TRAINING SET")

sns.heatmap(con_mat_test, annot = True, fmt="d", cmap="YlOrRd", xticklabels = class_names, yticklabel
f[1].set_xlabel("Predicted Class")
f[1].set_ylabel("True Class")
f[1].set_title("TESTING SET")

plt.tight_layout()
plt.show()
```

- ✓ Creates a figure with 2 subplots side-by-side: one for training and one for testing.
- ✓ Visualizes confusion matrices with color shading using Seaborn heatmaps.

- ✓ Includes axis labels (Predicted vs. True) and class names.
- ✓ Ensures plots don’t overlap and are properly spaced.

```
precision_train = cr_train['weighted avg']['precision']
recall_train = cr_train['weighted avg']['recall']
f1_score_train = cr_train['weighted avg']['f1-score']
precision_test = cr_test['weighted avg']['precision']
recall_test = cr_test['weighted avg']['recall']
f1_score_test = cr_test['weighted avg']['f1-score']

Metrics = [precision_train, recall_train, f1_score_train, accuracy_train, precision_test, recall_test,
return Metrics
```

- ✓ Extracts weighted average performance metrics from the classification reports.
- ✓ Weighted avg accounts for class imbalance by weighting each class’s contribution.
- ✓ All the metrics are returned.

Metric	Description
Precision train/test	True positives / (True + False Positives)
Recall train/test	True positives / (True + False Negatives)
f1_score train/test	Harmonic mean of precision & recall
Accuracy train/test	Overall prediction correctness

```
Metrics = pd.DataFrame(index = ['precision_train', 'recall_train', 'f1_score_train', 'accuracy_train', 'p
```

- ✓ A data frame is created for returning the function and comparing metrics for different classification models.

Classification models

Models considered

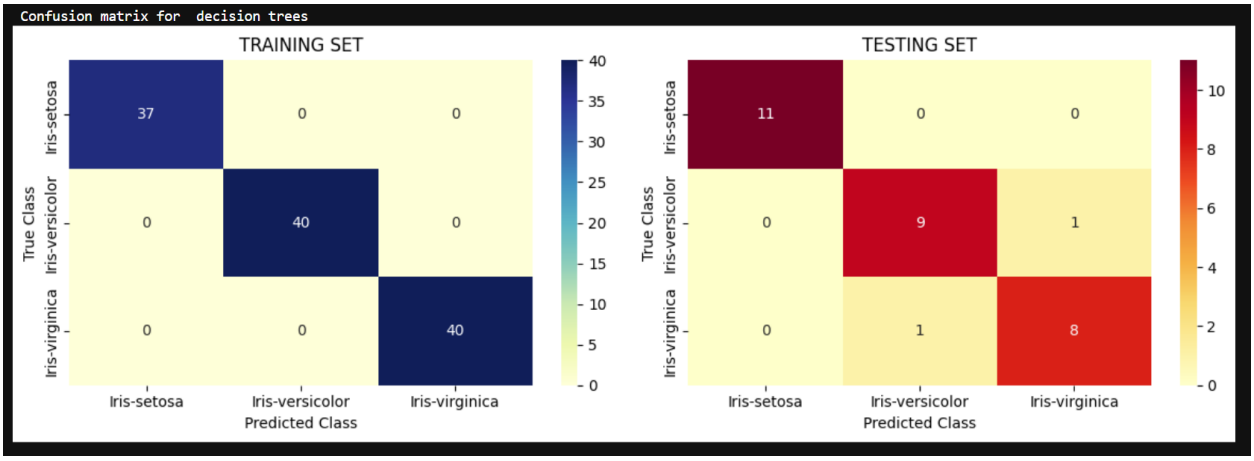
1. Decision Trees
2. SVM Classifier
3. Random Forest
4. KNN
5. Logistic Regression

Hyper parameter tuning - Grid Search technique

It provides a systematic and exhaustive approach to hyperparameter tuning. It helps evaluate all possible combinations of predefined parameter values using cross-validation, ensuring that the selected model performs well not just on training data but also on unseen data. It is easy to implement and interpret, Reliable for identifying optimal parameters in a manageable search space. While it can be computationally intensive, the accuracy and consistency it offers for small to medium-sized grids made it the ideal choice for my project.

After using the function for all the models considered, the following parameters have been found.

MODEL 1 - DECISION TREES



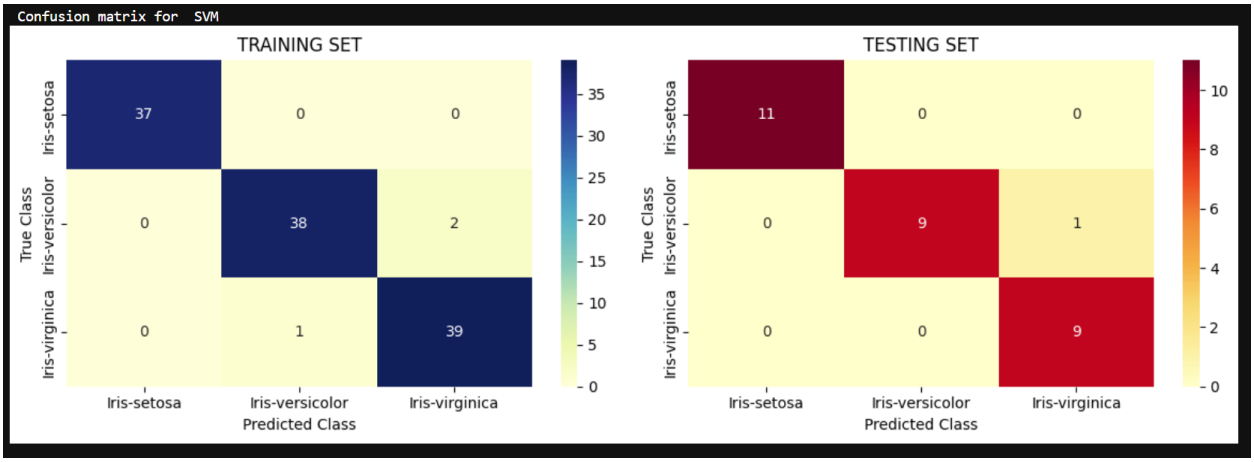
classification report for decision trees training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	1.00	1.00	1.00	40
Iris-virginica	1.00	1.00	1.00	40
accuracy			1.00	117
macro avg	1.00	1.00	1.00	117
weighted avg	1.00	1.00	1.00	117

classification report for decision trees testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	0.90	0.90	0.90	10
Iris-virginica	0.89	0.89	0.89	9
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

MODEL 2 - SVM CLASSIFIER



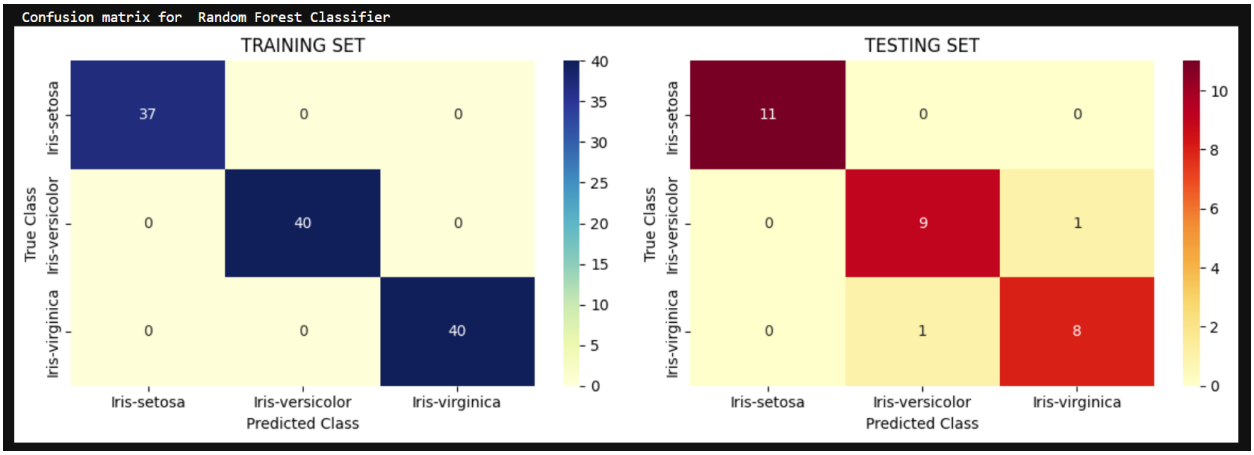
classification report for SVM training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	0.97	0.95	0.96	40
Iris-virginica	0.95	0.97	0.96	40
accuracy			0.97	117
macro avg	0.98	0.97	0.97	117
weighted avg	0.97	0.97	0.97	117

classification report for SVM testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.90	0.95	10
Iris-virginica	0.90	1.00	0.95	9
accuracy			0.97	30
macro avg	0.97	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

MODEL 3 - RANDOM FOREST CLASSIFIER



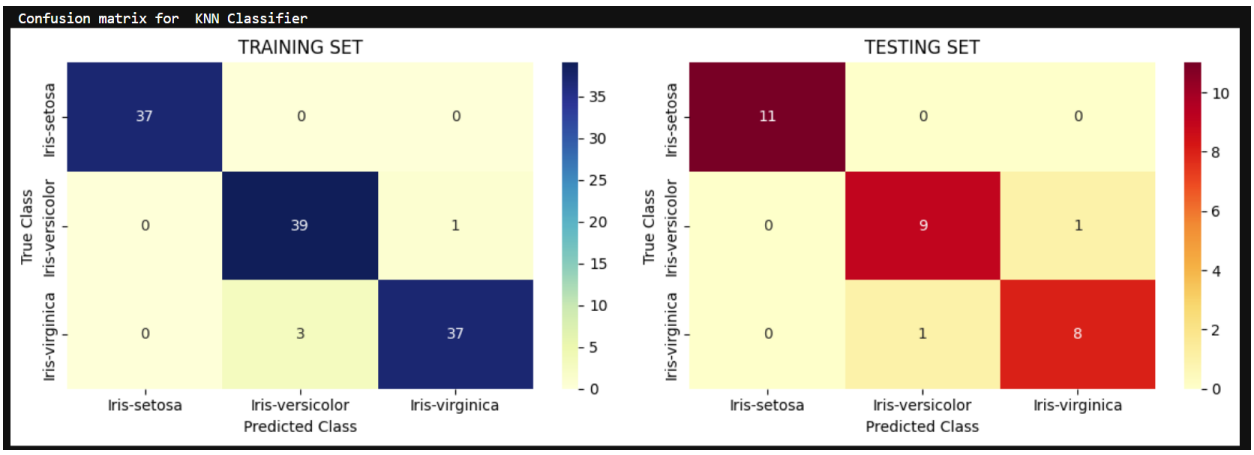
classification report for Random Forest Classifier training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	1.00	1.00	1.00	40
Iris-virginica	1.00	1.00	1.00	40
accuracy			1.00	117
macro avg	1.00	1.00	1.00	117
weighted avg	1.00	1.00	1.00	117

classification report for Random Forest Classifier testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	0.90	0.90	0.90	10
Iris-virginica	0.89	0.89	0.89	9
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

MODEL 4 - KNN CLASSIFIER



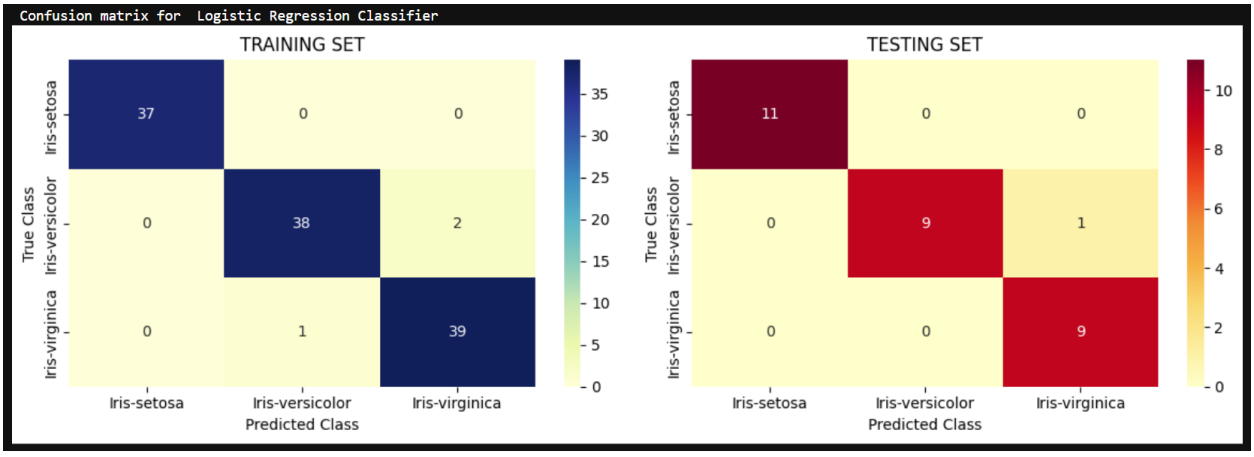
classification report for KNN Classifier training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	0.93	0.97	0.95	40
Iris-virginica	0.97	0.93	0.95	40
accuracy			0.97	117
macro avg	0.97	0.97	0.97	117
weighted avg	0.97	0.97	0.97	117

classification report for KNN Classifier testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	0.90	0.90	0.90	10
Iris-virginica	0.89	0.89	0.89	9
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

MODEL 5 - LOGISTIC REGRESSION CLASSIFIER



classification report for Logistic Regression Classifier training set:

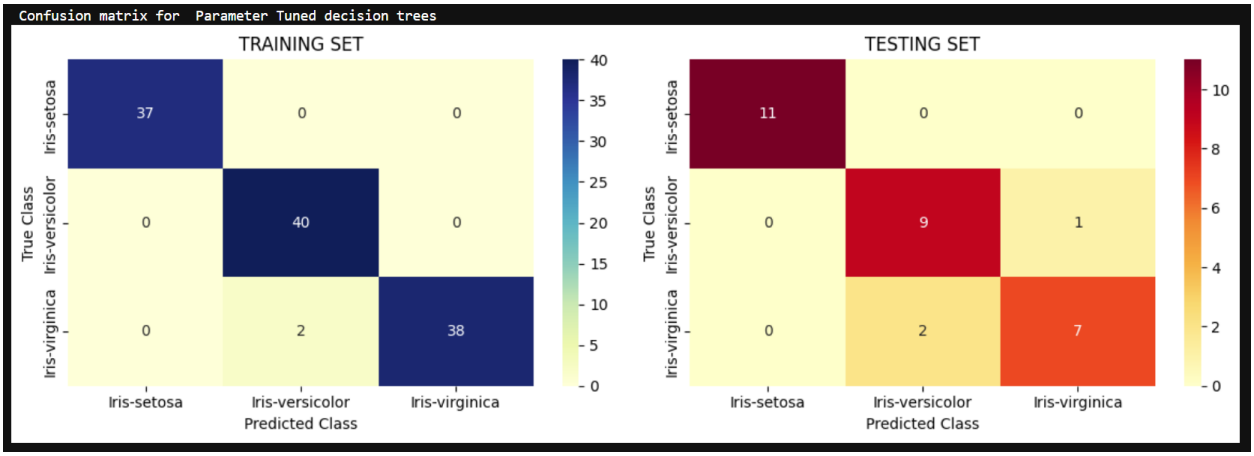
	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	0.97	0.95	0.96	40
Iris-virginica	0.95	0.97	0.96	40
accuracy			0.97	117
macro avg	0.98	0.97	0.97	117
weighted avg	0.97	0.97	0.97	117

classification report for Logistic Regression Classifier testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.90	0.95	10
Iris-virginica	0.90	1.00	0.95	9
accuracy			0.97	30
macro avg	0.97	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

After Hyperparameter tuning of the classification models, the outputs generated are below:

MODEL 1 - DECISION TREES



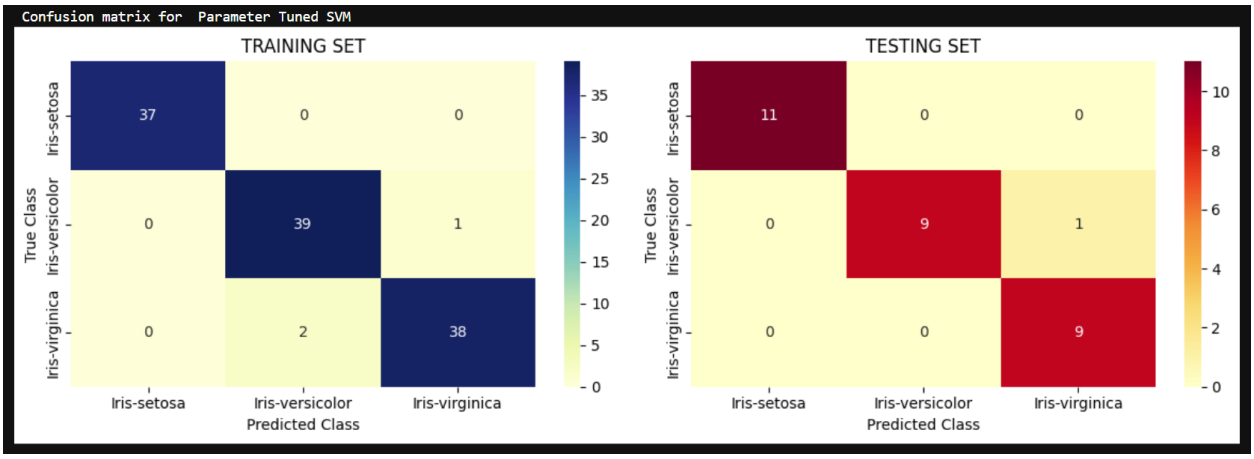
classification report for Parameter Tuned decision trees training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	0.95	1.00	0.98	40
Iris-virginica	1.00	0.95	0.97	40
accuracy			0.98	117
macro avg	0.98	0.98	0.98	117
weighted avg	0.98	0.98	0.98	117

classification report for Parameter Tuned decision trees testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	0.82	0.90	0.86	10
Iris-virginica	0.88	0.78	0.82	9
accuracy			0.90	30
macro avg	0.90	0.89	0.89	30
weighted avg	0.90	0.90	0.90	30

MODEL 2 - SVM CLASSIFIER



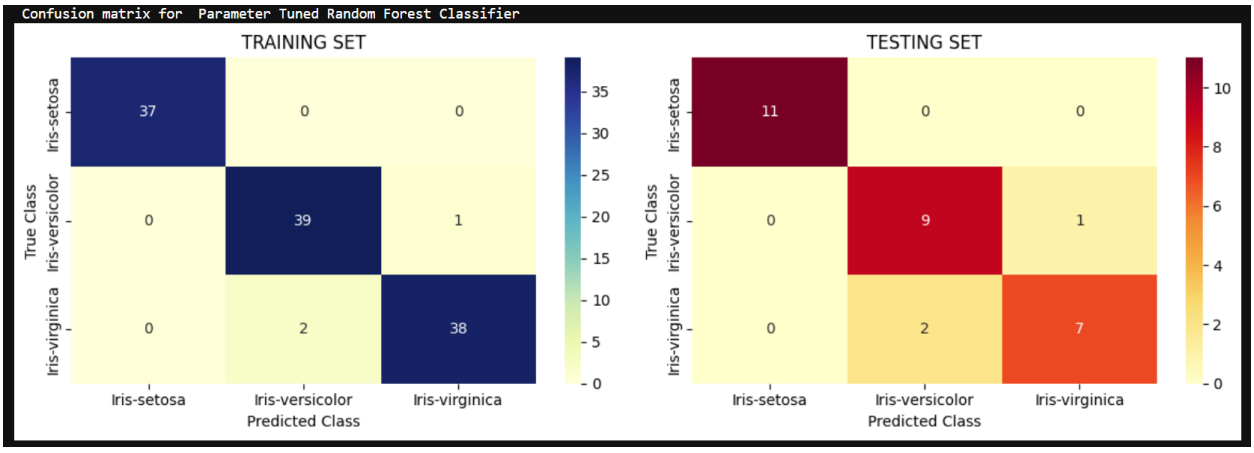
classification report for Parameter Tuned SVM training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	0.95	0.97	0.96	40
Iris-virginica	0.97	0.95	0.96	40
accuracy			0.97	117
macro avg	0.98	0.97	0.97	117
weighted avg	0.97	0.97	0.97	117

classification report for Parameter Tuned SVM testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.90	0.95	10
Iris-virginica	0.90	1.00	0.95	9
accuracy			0.97	30
macro avg	0.97	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

MODEL 3 - RANDOM FOREST CLASSIFIER



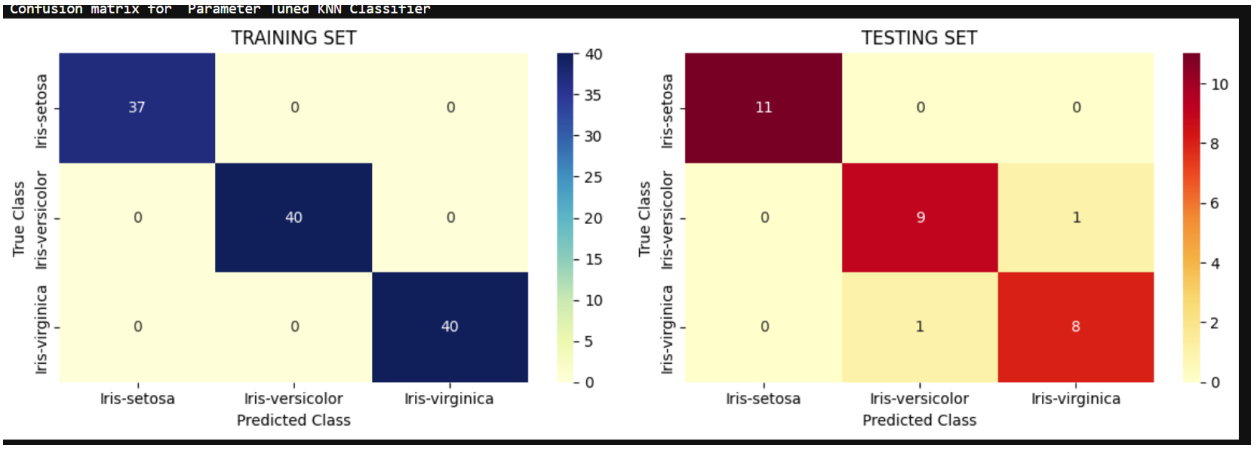
classification report for Parameter Tuned Random Forest Classifier training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	0.95	0.97	0.96	40
Iris-virginica	0.97	0.95	0.96	40
accuracy			0.97	117
macro avg	0.98	0.97	0.97	117
weighted avg	0.97	0.97	0.97	117

classification report for Parameter Tuned Random Forest Classifier testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	0.82	0.90	0.86	10
Iris-virginica	0.88	0.78	0.82	9
accuracy			0.90	30
macro avg	0.90	0.89	0.89	30
weighted avg	0.90	0.90	0.90	30

MODEL 4 - KNN CLASSIFIER



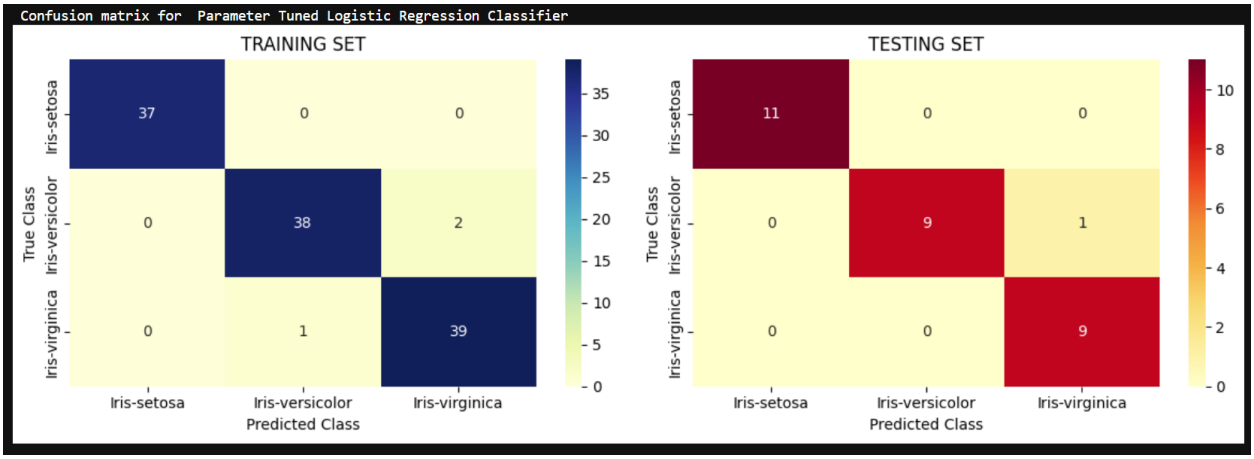
classification report for Parameter Tuned KNN Classifier training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	1.00	1.00	1.00	40
Iris-virginica	1.00	1.00	1.00	40
accuracy			1.00	117
macro avg	1.00	1.00	1.00	117
weighted avg	1.00	1.00	1.00	117

classification report for Parameter Tuned KNN Classifier testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	0.90	0.90	0.90	10
Iris-virginica	0.89	0.89	0.89	9
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

MODEL 5 - LOGISTIC REGRESSION CLASSIFIER



classification report for Parameter Tuned Logistic Regression Classifier training set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	37
Iris-versicolor	0.97	0.95	0.96	40
Iris-virginica	0.95	0.97	0.96	40
accuracy			0.97	117
macro avg	0.98	0.97	0.97	117
weighted avg	0.97	0.97	0.97	117

classification report for Parameter Tuned Logistic Regression Classifier testing set:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.90	0.95	10
Iris-virginica	0.90	1.00	0.95	9
accuracy			0.97	30
macro avg	0.97	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

The Metric Table of all the classification models:

	decision trees	Parameter Tuned decision trees	SVM	Parameter Tuned SVM	Random Forest	Parameter Tuned Random Forest	KNN	Parameter Tuned KNN	Logistic Regression	Parameter Tuned Logistic Regression
precision_train	1.000000	0.983720	0.974557	0.974557	1.000000	0.974557	0.966583	1.000000	0.974557	0.974557
recall_train	1.000000	0.982906	0.974359	0.974359	1.000000	0.974359	0.965812	1.000000	0.974359	0.974359
f1_score_train	1.000000	0.982895	0.974355	0.974355	1.000000	0.974355	0.965791	1.000000	0.974355	0.974355
accuracy_train	1.000000	0.982906	0.974359	0.974359	1.000000	0.974359	0.965812	1.000000	0.974359	0.974359
precision_test	0.933333	0.901894	0.970000	0.970000	0.933333	0.901894	0.933333	0.933333	0.970000	0.970000
recall_test	0.933333	0.900000	0.966667	0.966667	0.933333	0.900000	0.933333	0.933333	0.966667	0.966667
f1_score_test	0.933333	0.899440	0.966667	0.966667	0.933333	0.899440	0.933333	0.933333	0.966667	0.966667
accuracy_test	0.933333	0.900000	0.966667	0.966667	0.933333	0.900000	0.933333	0.933333	0.966667	0.966667

Model Interpretation and Insights

Is there any improvement in the models due to hyperparameter tuning?

- ✓ **DECISION TREES**: For the decision trees, it seems that the tuning did not improve the performance of the model. The parameters found after tuning appears to be less than the values appeared before tuning.
- ✓ **SVM**: For the svm classification, it seems that the tuning did not improve the performance of the model either. The parameter values after and before tuning appears to be the same.
- ✓ **RANDOM FOREST**: For the decision trees, it seems that the tuning did not improve the performance of the model. The parameters found after tuning appears to be less than the values appeared before tuning.
- ✓ **KNN**: for the knn classification, it appears that the performance of the model has been moderately improved. For the training set data the performance improved but for the testing set data the performance did not improve. The parameter values for the training set appear to have increased after tuning but for testing set, the values remaining the same.
- ✓ **LINEAR REGRESSION**: For the linear regression classification, it seems that the tuning did not improve the performance of the model either. The parameter values after and before tuning appears to be the same.

Best Model

```
The best model for precision: SVM - 0.9700
The best model for recall: SVM - 0.9667
The best model for f1_score: SVM - 0.9667
The best model for accuracy: SVM - 0.9667
```

Conclusion

The project aimed to classify Iris flowers into three distinct species: Iris-Setosa, Iris-Versicolor, and Iris-Virginica. This project focused on building and evaluating multiple classification models to identify the most effective algorithm for the given dataset thereby giving the best results in order to satisfy the aim. The workflow involved essential steps such as data preprocessing, exploratory data analysis, model training, hyperparameter tuning, and performance evaluation.

During the initial phase, the dataset was cleaned by identifying and removing duplicate values to prevent redundancy and improve model generalization. Visualizations such as scatter plots, histograms, and distribution plots provided insights into the feature relationships and class separability.

Several machine learning models were trained, including Decision Trees, K-Nearest Neighbors (KNN), Logistic Regression, Random Forest, and Support Vector Machines (SVM). To ensure optimal performance, GridSearchCV was employed for hyperparameter tuning. This method exhaustively searched through specified parameter combinations using cross-validation, helping prevent overfitting and ensuring that model performance generalized well to unseen data.

Model performance was assessed using four primary metrics:

1. Precision (how many predicted positives are actually positive)
2. Recall (how many actual positives are correctly identified)
3. F1-score (harmonic mean of precision and recall)
4. Accuracy (overall correctness of the model)

The results indicate that the SVM model is not only precise in its predictions but also maintains a strong balance between identifying true positives and minimizing false negatives, making it the most reliable and effective classifier for the given dataset.

SVM was successfully employed in training and testing the dataset and creating a model which classifies the iris flowers with accuracy giving reliable results.