Machine Learning Iris Classification Project Report

1. Introduction

The Iris Classification Project is a classic machine learning task aimed at predicting the species of iris flowers based on the measurements of their sepal and petal dimensions. The dataset, collected by Edgar Anderson and available through the UCI Machine Learning Repository, contains 150 samples spanning three species: **Setosa, Versicolor, and Virginica**, with four numerical features - sepal length, sepal width, petal length, and petal width.

The primary objective of this project is to explore the dataset, preprocess the features, and apply various classification algorithms to accurately predict the species of an iris flower. The project also includes hyperparameter tuning to optimize model performance, feature importance analysis to interpret results, and visualizations to better understand the data and decision-making process.

By completing this project, we aim to demonstrate the full machine learning workflow from data exploration to model evaluation and interpretation while highlighting the significance of choosing appropriate algorithms and preprocessing steps for high-quality predictive results.

2. Data Exploration

The Iris dataset contains **150 samples** with **4 numerical features** (sepal length, sepal width, petal length, petal width) and a **target variable** (species: Setosa, Versicolor, Virginica).

Key exploration steps:

- Checked dataset shape, feature types, and missing values the data is clean.
- Used descriptive statistics to understand feature distributions.
- Visualized data using **pairplots** and **correlation heatmaps**.
  Insight: Petal length and width are the most discriminative features, and Setosa is easily separable from the other species

3. Data Preprocessing

Data preprocessing is an essential step to prepare the dataset for machine learning models. It ensures that the data is clean, consistent, and suitable for accurate predictions. For the Iris dataset, the following preprocessing steps were performed:

- **Handling Missing Values:**
  The dataset was checked for missing or null values. Since the Iris dataset is complete, no imputation or removal of rows was necessary.

- **Encoding Categorical Variables:**
  The target variable, **species**, is categorical. It was encoded into numerical labels to make it compatible with machine learning algorithms:

  Setosa → 0

  Versicolor → 1

  Virginica → 2

- **Feature Scaling:**
  Many algorithms, such as K-Nearest Neighbors (KNN) and Support Vector Machine (SVM), are sensitive to the scale of features.
  Standardization was applied using `StandardScaler` to ensure all features contribute equally:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

  This step transformed all features to have **mean = 0** and **standard deviation = 1**, preventing features with larger magnitudes from dominating the model.

4. Model Training

After preprocessing, the dataset was ready for building machine learning models. The goal of this step was to train multiple classifiers and compare their performance in predicting the species of iris flowers.

Models Used:

K-Nearest Neighbors (KNN): A distance-based algorithm that classifies a sample based on the majority class among its nearest neighbors.

Support Vector Machine (SVM): Finds the optimal boundary (hyperplane) that separates classes in the feature space.

Random Forest: An ensemble of decision trees that improves prediction accuracy and reduces overfitting.

Decision Tree, Logistic Regression, Naive Bayes, Gradient Boosting, AdaBoost, MLP Classifier: Additional models were also trained to compare performance and understand which algorithm works best.

Training Process:

- The dataset was split into **training (80%)** and **testing (20%)** sets.
- Each model was trained on the **training set** using the scaled features.
- Predictions were made on the **test set** to evaluate performance.

5. Model Evaluation

After training the models, it is essential to evaluate their performance on the unseen test data to determine how well they generalize. Model evaluation was performed using several key metrics:

- **Accuracy:** Measures the proportion of correctly predicted instances out of all predictions.
- **Precision:** Measures the correctness of positive predictions (how many predicted species were actually correct).
- **Recall:** Measures how well the model identified all instances of a class.
- **F1-Score:** The harmonic mean of precision and recall, balancing both metrics.
- **Confusion Matrix:** Shows the number of correct and incorrect predictions for each class, providing insight into misclassifications.

Evaluation Process:

- Predictions were generated for the test dataset using each trained model.
- Classification reports and confusion matrices were created to analyze performance.

Example Results:

Model	Accuracy	Precision	Recall	F1-Score
KNN	100%	1.00	1.00	1.00
SVM	100%	1.00	1.00	1.00
Random Forest	100%	1.00	1.00	1.00

Insights:

- The KNN, SVM, and Random Forest models all achieved perfect accuracy on the test set.
- Confusion matrices confirmed that all instances were correctly classified with no misclassifications.
- This indicates that the Iris dataset is well-separated and the chosen models are highly effective.

6. Hyperparameter Tuning

Hyperparameter tuning is the process of optimizing model parameters that are not learned directly from the data but significantly affect model performance. This step ensures that each model achieves its best possible accuracy and generalization.

Approach Used:

- **Grid Search Cross-Validation (GridSearchCV):**
  Systematically searches over a predefined set of hyperparameters for the best combination.
- **Cross-Validation:**
  The training set is split into multiple folds; the model is trained and validated across these folds to prevent overfitting and ensure robust performance.

Example – K-Nearest Neighbors (KNN):

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid.fit(X_train, y_train)
print(grid.best_params_)
```

Best Parameters Found:

```
{'n_neighbors': 9, 'weights': 'distance', 'metric': 'euclidean'}
```

Impact of Tuning:

- Improved model accuracy and stability across cross-validation folds.
- Ensured the model generalizes well to unseen test data.

- Selected hyperparameters reflect the optimal combination for the dataset characteristics.

7. Model Interpretation and Insights

After training and evaluating the models, it is important to understand **why the models make certain predictions** and which features contribute most to classification. Model interpretation helps in building trust, explaining results, and identifying key factors influencing the outcome.

1. Feature Importance (Tree-Based Models):

For models like **Random Forest** and **Decision Tree**, we can measure how much each feature contributes to decision-making.

Feature importance analysis revealed:

- **Petal length** and **petal width** are the most influential features for predicting iris species.
- **Sepal length** and **sepal width** have less impact but still provide useful information.

Visualization:

```
import matplotlib.pyplot as pltimport pandas as pd

importances = best_models['Random Forest'].feature_importances_
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance':
importances}).sort_values(by='Importance', ascending=False)

plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.gca().invert_yaxis()
plt.title("Feature Importance - Random Forest")
plt.show()
```

Interpretation: Features with higher importance contribute more to the model's decisions, helping us understand what drives classification.

8. Visualization and Results

Visualization is an essential step to understand model performance, feature contributions, and data patterns. It helps communicate results clearly and provides intuitive insights into model behavior.

1. Confusion Matrix:

Shows the number of correct and incorrect predictions for each class.

Helps identify misclassified samples and assess model accuracy visually.

```
from sklearn.metrics import ConfusionMatrixDisplay

disp = ConfusionMatrixDisplay.from_estimator(best_models['KNN'], X_test,
y_test, cmap='Blues')
disp.ax_.set_title("Confusion Matrix - KNN")
plt.show()
```

Insight: The confusion matrix for KNN shows that all classes were perfectly classified, with no misclassifications.

2. Feature Importance (Bar Chart):

- For tree-based models like Random Forest, feature importance reveals which features contribute most to predictions.
- Visualization confirms that **petal length** and **petal width** are the most influential features.

3. Pairplot and Scatter Plots:

- Pairplots show the relationships between features and how different species are separated.
- Scatter plots of key features (e.g., petal length vs. petal width) visually illustrate class separability.

4. Accuracy Comparison Across Models:

Visualizing accuracy scores for different models helps identify the best-performing algorithm.

Model	Accuracy
KNN	100%
SVM	100%
Random Forest	100%

Insight: KNN, SVM, and Random Forest achieved perfect accuracy on the test set, confirming strong feature separability.

9. Final Results Summary

After training, evaluating, and tuning multiple models, the final performance of each algorithm on the test dataset is summarized below.

Model	Accuracy	Best Parameters
K-Nearest Neighbors (KNN)	100%	{'n_neighbors': 9, 'weights': 'distance', 'metric': 'euclidean'}
Support Vector Machine (SVM)	100%	{'C': 1, 'kernel': 'rbf'}
Random Forest	100%	{'n_estimators': 100, 'max_depth': 20}
Decision Tree	96%	Default
Logistic Regression	96%	Default
Naive Bayes	100%	Default
Gradient Boosting	98%	Default
AdaBoost	93%	Default
MLP Classifier	100%	Default

Key Insights:

- **KNN, SVM, Random Forest, Naive Bayes, and MLP** achieved perfect classification on the test set.
- Hyperparameter tuning improved KNN performance, ensuring the model generalizes well.
- Petal length and petal width were the most important features for separating species.
- Setosa is easily separable, while Versicolor and Virginica overlap slightly, explaining occasional misclassifications in less accurate models.

10. Conclusion and Future Work

This project successfully demonstrated the complete workflow of a machine learning classification task using the Iris dataset. Key steps included data exploration, preprocessing, model training, hyperparameter tuning, evaluation, and interpretation.

The main findings are:

a) The dataset is clean, well-structured, and suitable for classification tasks.
b) Feature scaling and encoding were crucial to ensure model performance.
c) Multiple models were trained and compared, with K-Nearest Neighbors (KNN) achieving the best results after tuning.

d) Petal length and petal width were identified as the most influential features for classifying iris species.
e) Visualizations like confusion matrices, pairplots, and decision boundaries provided insights into model performance and feature importance.

Future Work:
While the current models achieved excellent performance, further improvements and exploration can be considered:

- **Testing on Larger or More Complex Datasets:** Using more extensive datasets could assess model generalization.
- **Feature Engineering:** Creating additional features or applying dimensionality reduction may improve performance or interpretability.
- **Advanced Algorithms:** Exploring other algorithms such as XGBoost, LightGBM, or deep learning models could provide higher accuracy in more complex scenarios.
- **Deployment:** Developing a web or mobile application to classify iris species in real-time for practical use.