Iris Flower Classification

Data Science Intern Project (ML_FA_DA)

Name : Sushanth Babu Tammali

Project Overview

This project focuses on applying supervised machine learning to classify species of the Iris flower dataset. Through structured data preparation, exploratory analysis, and predictive modeling using Logistic Regression, the workflow demonstrates a complete lifecycle of a classification problem. The solution is framed with scalability and interpretability in mind—key attributes in public-sector analytics.

Objectives

- Understand the structure of the Iris dataset
- Apply preprocessing techniques for clean input data
- Visualize relationships among features and classes
- Train and evaluate a Logistic Regression model
- Present classification performance using metrics and visual tools

Step-by-Step Code Explanation

1. Library Imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns ...

- **Why?** Libraries are modular tools for analysis:
- pandas and numpy: efficient data manipulation.

- matplotlib and seaborn: insightful visualizations.
- sklearn: robust ML algorithms and preprocessing functions.

2. Dataset Loading

```
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

- The load_iris() function brings in a pre-cleaned dataset with four key features.
- Assigning species helps link feature data to labeled classes: *Setosa*, *Versicolor*, and *Virginica*.

3. Data Preprocessing

```
X = df.drop('species', axis=1)
y = df['species']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- **Feature Matrix (X)** and **Target Vector (y)** are separated.
- **Standardization** ensures uniform scaling, aiding model performance—especially for algorithms sensitive to feature magnitude like Logistic Regression.

4. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
```

⍰⍰

- Splits data for training (80%) and validation (20%), maintaining generalizability and reducing overfitting.
- The random_state ensures reproducibility.

5. Exploratory Data Analysis (EDA)

```
sns.pairplot(df, hue="species")
sns.heatmap(df.drop('species', axis=1).corr(), annot=True, cmap='coolwarm')
```

- **Pairplot**: Visualizes pairwise feature relationships, showing strong class separation particularly via petal dimensions.
- **Correlation Heatmap**: Quantifies feature relationships to inform model understanding (e.g., petal length and width are highly correlated).

6. Model Training: Logistic Regression

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

- Logistic Regression is efficient for multiclass classification via the One-vs-Rest (OvR) approach.
- The model learns to separate classes by identifying optimal decision boundaries in scaled feature space.

7. Model Evaluation

```
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

- **Confusion Matrix**: Evaluates true vs. predicted labels across three classes.
- **Classification Report**: Breaks down precision, recall, F1-score, and support per class—offering granular performance insights.

Findings & Insights

- The model shows strong performance, often achieving near-perfect precision and recall.
- Most errors arise between *Versicolor* and *Virginica*—understandable given their feature overlap.
- Logistic Regression proves effective for baseline classification and interpretability.

Potential Extensions

- Cross-validation for generalization.
- Model comparison with SVM or Decision Trees.
- Feature importance visualization using model coefficients.
- Confusion matrix heatmap for intuitive presentation.
- Interactive dashboards using plotly or dash.