Iris Flower Classification Project

Introduction

The Iris Flower Classification project aims to classify iris flowers into three species based on their physical characteristics. The dataset used for this project is the famous Iris dataset, which consists of 150 samples from three species of iris flowers: Setosa, Versicolor, and Virginica. Each sample contains four features: sepal length, sepal width, petal length, and petal width. This project utilizes machine learning techniques, particularly Random Forest Classifier, to perform classification.

1. Data Loading

Code Explanation

```
python
Copy code
import warnings
warnings.filterwarnings('ignore')

import pandas as pd

data_path = r"D:\# DATA SCIENCE\# PROJECTS\- PROJECTS INTERNSHIPS\CODEALPHA
- DATA SCIENCE\Iris Flower Classification Project\Iris.csv"

iris_df = pd.read_csv(data_path)

print("First few rows of the dataset:")
print(iris_df.head())
```

- **Library Import**: The code begins by importing necessary libraries. Here, `warnings` is imported to suppress any potential warnings during execution. `pandas` is imported for data manipulation and analysis.
- **Data Path**: The path to the dataset is defined. Make sure this path is correct based on your local directory structure.
- **Loading the Dataset**: The dataset is loaded into a pandas DataFrame using `pd.read_csv()`. This function reads the CSV file and returns a DataFrame, which is a two-dimensional size-mutable data structure with labeled axes (rows and columns).
- **Display Data**: The first few rows of the dataset are displayed using `head()`, providing an overview of the data structure and the values it contains.

Dataset Overview

The dataset consists of the following columns:

- **SepalLengthCm**: Length of the sepal in centimeters.
- **SepalWidthCm**: Width of the sepal in centimeters.
- **PetalLengthCm**: Length of the petal in centimeters.
- **PetalWidthCm**: Width of the petal in centimeters.
- **Species**: The species of the iris flower (Setosa, Versicolor, Virginica).

2. Data Exploration

Code Explanation

```
python
Copy code
import seaborn as sns
import matplotlib.pyplot as plt

print("\nDataset Info:")
print(iris_df.info())

print("\nSummary Statistics:")
print(iris_df.describe())

print("\nMissing Values in the Dataset:")
print(iris_df.isnull().sum())

sns.pairplot(iris_df, hue='Species')
plt.show()
```

- **Visualization Libraries**: The code imports `seaborn` for statistical data visualization and `matplotlib.pyplot` for plotting graphs.
- **Dataset Information**: The `info()` method is used to display concise information about the DataFrame, including the number of non-null entries, data types, and memory usage. This step helps in understanding the structure and size of the dataset.
- **Summary Statistics**: The `describe()` method generates descriptive statistics of the DataFrame. This includes count, mean, standard deviation, min, and max values, which provide insight into the distribution of numerical features.
- **Missing Values**: The `isnull().sum()` method checks for missing values in the dataset. This is crucial for ensuring data quality, as missing values can affect model training and evaluation.
- **Pairplot Visualization**: A pairplot is created to visualize relationships between the features, colored by species. This allows us to see how different species overlap in feature space and helps in understanding the separability of classes.

Insights from Data Exploration

From the pairplot and summary statistics, we can derive the following insights:

- The three species can be visually distinguished based on petal length and petal width. Setosa is easily separable from the other two species.
- The sepal dimensions have some overlap between species, suggesting they may not be as useful for classification.

3. Data Preprocessing

Code Explanation

```
python
Copy code
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler

if 'Id' in iris_df.columns:
    iris_df = iris_df.drop(columns=['Id'])

X = iris_df.drop(columns=['Species'])
y = iris_df['Species']

y = y.astype('category').cat.codes

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

plt.figure(figsize=(15, 10))
for i, column in enumerate(X.columns, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x='Species', y=column, data=iris_df)
    plt.title(f'Box Plot of {column} by Species')
plt.tight_layout()
plt.show()
```

- **Importing Preprocessing Libraries**: The code imports `train_test_split` from `sklearn.model_selection` for splitting the dataset into training and testing sets and `StandardScaler` for standardizing features.
- **Dropping Unused Columns**: If an 'Id' column is present, it is dropped. This step is optional and depends on whether the dataset includes an identifier.
- **Feature and Target Separation**: Features are separated into `X`, which contains all columns except 'Species', and `y`, which contains the target variable (species). This step is crucial for supervised learning.
- **Encoding Target Variable**: The target variable `y` is converted to categorical codes using `astype('category').cat.codes`. This is necessary for models that require numerical input.
- **Train-Test Split**: The dataset is split into training and testing sets using `train_test_split()`. 70% of the data is used for training, and 30% is reserved for testing. The `stratify` parameter ensures that the class distribution is maintained in both sets.
- **Standardization**: The `StandardScaler` is used to standardize the features by removing the mean and scaling to unit variance. This step is particularly important for algorithms sensitive to feature scaling, such as SVM and Logistic Regression.
- **Box Plot Visualization**: Box plots are generated for each feature categorized by species. This helps visualize the distribution and potential outliers in the dataset.

Insights from Data Preprocessing

Standardizing the features can help improve the performance of machine learning models, especially those that rely on distance calculations. The box plots can reveal important characteristics about the distribution of features, including central tendency and spread.

4. Model Building

Code Explanation

```
python
Copy code
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)
```

- **Random Forest Import**: The Random Forest Classifier from `sklearn.ensemble` is imported for building the classification model.
- **Model Initialization**: The model is initialized with `n_estimators=100`, which specifies the number of trees in the forest. The `random_state` parameter ensures reproducibility.
- **Model Training**: The model is trained using the `fit()` method on the training data (`X_train` and `y_train`). This step involves constructing decision trees based on the training data to learn the patterns associated with each species.

Insights from Model Building

Random Forest is a robust ensemble learning method that reduces overfitting by averaging the predictions from multiple decision trees. This method is particularly effective for classification tasks with a moderate number of features.

5. Model Training with Hyperparameter Tuning

Code Explanation

```
python
Copy code
from sklearn.ensemble import RandomForestClassifier
from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import HalvingGridSearchCV
import numpy as np

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3]
}

halving_grid_search = HalvingGridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    factor=2,
    n_jobs=-1,
    cv=5,
    random_state=42
)

halving_grid_search.fit(X_train, y_train)

print("\nBest Parameters from Halving Grid Search:")
```

```
print(halving_grid_search.best_params_)
print(f"Best Cross-Validation Score: {halving_grid_search.best_score_ *
100:.2f}%")
```

- **Hyperparameter Tuning**: The code employs `HalvingGridSearchCV`, an efficient way to perform hyperparameter tuning. This method reduces the number of candidates as it proceeds through the cross-validation process.
- **Parameter Grid Definition**: A grid of hyperparameters is defined, including:
  - `n_estimators`: Number of trees in the forest.
  - `max_depth`: Maximum depth of the trees.
  - `min_samples_split`: Minimum number of samples required to split an internal node.
  - `min_samples_leaf`: Minimum number of samples required to be at a leaf node.
- **HalvingGridSearchCV Initialization**: The `HalvingGridSearchCV` object is initialized with the Random Forest model and the defined parameter grid. The `factor` determines how many candidates are retained at each iteration, while `cv` specifies the number of cross-validation folds.
- **Model Fitting**: The halving grid search is fitted to the training data to find the best combination of hyperparameters.
- **Best Parameters and Score**: Finally, the best parameters and the best cross-validation score are printed, providing insight into the optimal configuration for the Random Forest model.

Insights from Hyperparameter Tuning

Hyperparameter tuning is crucial for optimizing model performance. Using techniques like `HalvingGridSearchCV` can significantly reduce computation time while efficiently finding the best parameters.

6. Model Evaluation

Code Explanation

```
python
Copy code
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

y_pred = halving_grid_search.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy: {accuracy * 100:.2f}%")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Setosa', 'Versicolor', 'Virginica'], yticklabels=['Setosa',
'Versicolor', 'Virginica'])
```

```
plt.title('Confusion Matrix')
plt.xlabel('Predicted Species')
plt.ylabel('True Species')
plt.show()
```

- **Prediction**: The trained model predicts the species for the test data using the `predict()` method.
- **Accuracy Calculation**: The accuracy of the model is calculated using `accuracy_score()` and displayed as a percentage.
- **Classification Report**: A detailed classification report is generated using `classification_report()`, providing precision, recall, and F1-score for each class. This report offers deeper insights into the model's performance beyond mere accuracy.
- **Confusion Matrix**: The confusion matrix is computed using `confusion_matrix()` and visualized with a heatmap. This matrix shows the counts of true positive, false positive, true negative, and false negative predictions, helping to understand the model's strengths and weaknesses.

Insights from Model Evaluation

Evaluation metrics such as accuracy, precision, recall, and F1-score are critical for understanding model performance. The confusion matrix provides a clear visual representation of the classification results, indicating where the model is performing well and where it may need improvement.

Conclusion

The Iris Flower Classification project demonstrates a systematic approach to data science, from data loading and exploration to preprocessing, model building, tuning, and evaluation. Utilizing the Random Forest Classifier, we achieved a high level of accuracy in classifying iris species based on their features. This project highlights the importance of proper data preparation, model selection, and evaluation techniques in machine learning.

Future Work

Future enhancements may include:

- Exploring other classification algorithms (e.g., Support Vector Machine, K-Nearest Neighbors) to compare performance.
- Implementing additional feature engineering techniques to improve model accuracy.
- Conducting a more thorough exploratory data analysis (EDA) to gain deeper insights into feature relationships.