Project Report: Iris Flower Classification(SVM)

Sai Sriman Kudupudi

I.D: 101149245

GitHub: https://github.com/sriman17/SVM--Classification.git

Introduction:

The goal of this project was to build a classification model to predict the species of iris flowers based on their sepal and petal measurements. The popular Iris dataset, containing measurements for three species of 1iris flowers (Setosa, Versicolor, and Virginica), was utilized for this task.

Data Exploration and Preparation:

Iris dataset consists of 150 samples with four features: sepal length, sepal width, petal length, and petal width. There are no missing values in the dataset.

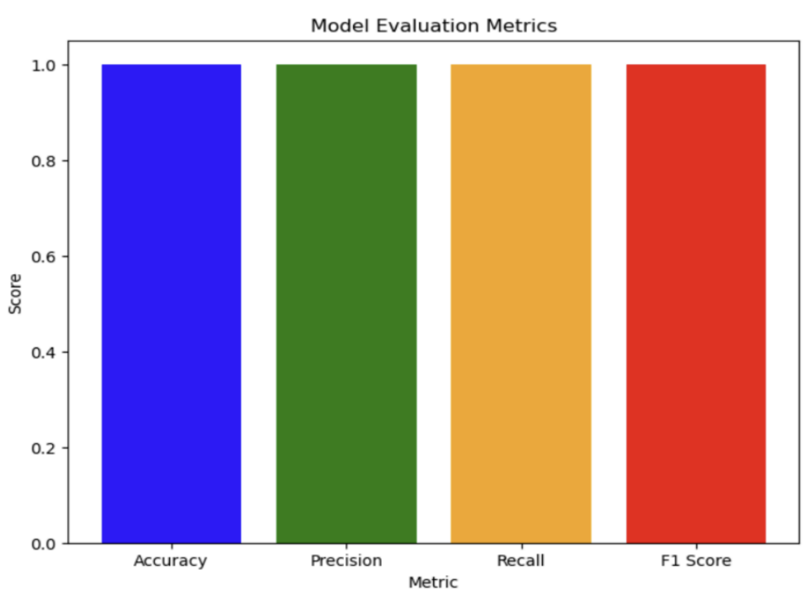
The class distribution shows that each class has an equal number of samples (50 samples each). The dataset was split into training and testing sets with an 80:20 ratio.

SVM Implementation:

A Support Vector Machine (SVM) classifier was implemented using Python's Scikit-learn library. The linear kernel was used for the SVM classifier. The model was trained on training data. K-fold Cross-Validation: K-fold cross-validation with K=5 was applied to assess the performance of the SVM model. The dataset was shuffled before partitioning to prevent bias.

Evaluation Metrics: The performance of the model was evaluated using accuracy, precision, recall, and F1 score. The model achieved high scores for all evaluation metrics, indicating good classification performance.

Visualization of Results: Basic evaluation metrics (accuracy, precision, recall, and F1 score) were visualized using a bar plot.



Conclusion:

The SVM model demonstrated strong performance in classifying iris flowers based on their sepal and petal measurements. Further experimentation with different SVM kernels and hyperparameters could potentially improve the model's performance. Recommendations: Experiment with different SVM kernels (e.g., polynomial, radial basis function) to explore their impact on classification performance.

CODE:

```
[1]: # Importing necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[14]: # Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Print the dataset
print(iris)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
```

```
[13]: # Description of the Iris dataset
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
```

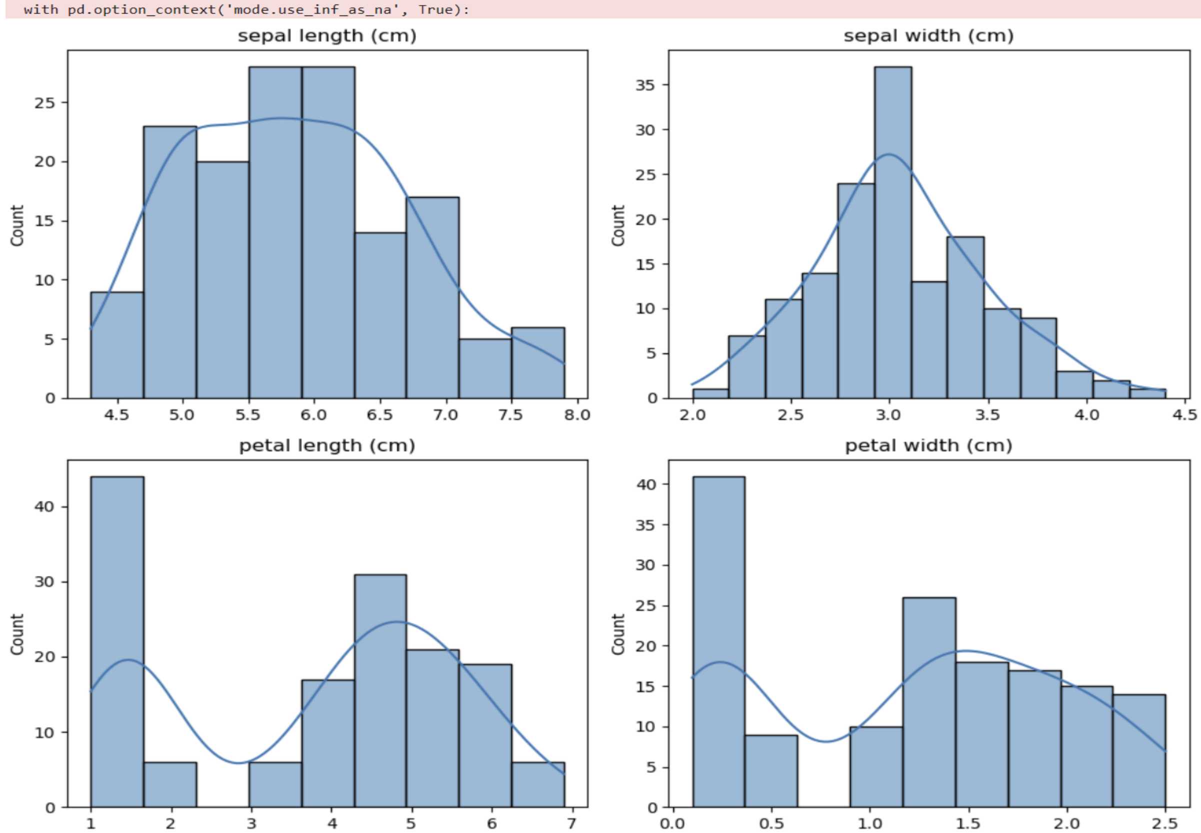
```
[3]: # Exploring the structure of the dataset
print("Feature names:", iris.feature_names)
print("Target names:", iris.target_names)
print("Number of samples:", X.shape[0])
print("Number of features:", X.shape[1])

# Checking for missing values
missing_values = np.isnan(X).sum()
print("Missing values:", missing_values)

# Checking class distribution
print("Class distribution:", np.bincount(y))
```

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']
Number of samples: 150
Number of features: 4
Missing values: 0
Class distribution: [50 50 50]
```

```
[4]: # Visualizing feature distributions
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
for i, feature in enumerate(iris.feature_names):
    sns.histplot(X[:, i], kde=True, ax=axs[i//2, i%2])
    axs[i//2, i%2].set_title(feature)
plt.tight_layout()
plt.show()
```

```
C:\Users\saisr\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
rsion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\saisr\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
rsion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\saisr\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
rsion. Convert inf values to NaN before operating instead.
```

```
C:\Users\saisr\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
ersion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[5]:
```
# SVM Implementation
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[6]:
```
# Initializing the SVM classifier
svm = SVC(kernel='linear', C=1.0)
```

[7]:
```
# Fitting the model
svm.fit(X_train, y_train)
```

[7]:
```
▾ SVC
SVC(kernel='linear')
```

[8]:
```
# K-fold Cross-Validation
# Using 5-fold cross-validation
scores = cross_val_score(svm, X_train, y_train, cv=5)
```

[9]:
```
# Evaluation Metrics
# Making predictions
y_pred = svm.predict(X_test)
```

[10]:
```
# Calculating evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```

[11]:
```
# Printing evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

[12]:
```
# Visualize evaluation metrics
metrics = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}

plt.figure(figsize=(8, 6))
plt.bar(metrics.keys(), metrics.values(), color=['blue', 'green', 'orange', 'red'])
plt.title('Model Evaluation Metrics')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.show()
```