ds-practical-6

October 28, 2024

0.1 DS PRACTICAL 6

0.2 Performing Classifiaction using sklearn for a given problem statement.

0.2.1 Problem Statement: Iris Flower Classification using KNN Algorithm

The aim of this experiment is to develop a machine learning model that can accurately classify Iris flowers into three distinct species: Setosa, Versicolor, and Virginica based on four features: sepal length, sepal width, petal length, and petal width. Using the Iris dataset from sklearn, the K-Nearest Neighbors (KNN) algorithm will be implemented to classify the flower species based on the given feature set.

0.2.2 Dataset Description:

- **Size:** The dataset contains a total of 150 samples (instances), with 50 samples from each of the three species of Iris flowers: Iris-setosa, Iris-versicolor, and Iris-virginica.
- **Features:** Sepal length, sepal width, petal length, and petal width (in cm).
- **Target variable:** Species of Iris flower (Setosa, Versicolor, Virginica).

0.2.3 Load the Iris dataset from sklearn and create a DataFrame with the data

[6]:
```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df
```

[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	…	…	…	…

```
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

[7]:
```
df['species'] = iris.target
df
```

[7]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                 …                  …                 …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

     species
0          0
1          0
2          0
3          0
4          0
..       …
145        2
146        2
147        2
148        2
149        2

[150 rows x 5 columns]
```

In the context of the Iris dataset, the values 0, 1, and 2 in iris.target represent the three different species of iris flowers:

0: Setosa 1: Versicolor 2: Virginica

0.2.4 Split the dataset into features (X) and target (y)

[9]:
```
X = df.drop('species', axis=1)  # Features
y = df['species']  # Target
```

[10]:
```
print(X)
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 …                 …                  …                 …
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

[150 rows x 4 columns]
```

[11]:
```
print(y)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: species, Length: 150, dtype: int64
```

0.2.5 Split the data into training and testing sets

[12]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[13]:
```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (105, 4)
X_test shape: (45, 4)
y_train shape: (105,)
y_test shape: (45,)
```

[14]:
```
print("X_train:\n", X_train)
```

```
X_train:
      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
81                  5.5               2.4                3.7               1.0
133                 6.3               2.8                5.1               1.5
137                 6.4               3.1                5.5               1.8
75                  6.6               3.0                4.4               1.4
109                 7.2               3.6                6.1               2.5
..                  …                 …                  …                 …
71                  6.1               2.8                4.0               1.3
106                 4.9               2.5                4.5               1.7
14                  5.8               4.0                1.2               0.2
92                  5.8               2.6                4.0               1.2
102                 7.1               3.0                5.9               2.1

[105 rows x 4 columns]
```

[38]:
```
print("X_test:\n", X_test.head())
```

```
X_test:
      sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
73                  6.1               2.8                4.7               1.2
18                  5.7               3.8                1.7               0.3
118                 7.7               2.6                6.9               2.3
78                  6.0               2.9                4.5               1.5
76                  6.8               2.8                4.8               1.4
```

[16]:
```
print("y_train:\n", y_train)
```

```
y_train:
 81     1
133    2
137    2
75     1
109    2
      ..
71     1
106    2
14     0
```

```
92     1
102    2
Name: species, Length: 105, dtype: int64
```

[37]:
```
print("y_test:\n", y_test.head())
```

```
y_test:
 73     1
18     0
118    2
78     1
76     1
Name: species, dtype: int64
```

0.2.6 Perform Feature Scaling (Z-Score Normalization)

This step ensures that each feature contributes equally to the model

[21]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler() # mean 0 and std of 1
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[45]:
```
print(X_test_scaled[:5])
```

```
[[ 0.3100623  -0.50256349  0.484213   -0.05282593]
 [-0.17225683  1.89603497 -1.26695916 -1.27039917]
 [ 2.23933883 -0.98228318  1.76840592  1.43531914]
 [ 0.18948252 -0.26270364  0.36746819  0.35303182]
 [ 1.15412078 -0.50256349  0.54258541  0.2177459 ]]
```

This line transforms the test data using the mean and standard deviation computed from the training data. It is important to note that the scaler should only be fit on the training data and then applied to both the training and test data to avoid any bias or data leakage. This ensures that the test data is scaled in the same way as the training data, maintaining consistency.

---

[46]:
```
print(X_train_scaled[:5])
```

```
[[-0.4134164  -1.46200287 -0.09951105 -0.32339776]
 [ 0.55122187 -0.50256349  0.71770262  0.35303182]
 [ 0.67180165  0.21701605  0.95119225  0.75888956]
 [ 0.91296121 -0.02284379  0.30909579  0.2177459 ]
 [ 1.63643991  1.41631528  1.30142668  1.70589097]]
```

It fits the scaler to the X_train data (calculates the mean and standard deviation for each feature) and then transforms the training data by applying the standardization. After this transformation, each feature will have a mean of 0 and a standard deviation of 1 in the training data.

0.2.7 Initialize the KNN classifier

[26]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5) # You can experiment with different k␣
↪values
knn.fit(X_train, y_train) # Train the model
```

[26]:
```
KNeighborsClassifier()
```

0.2.8 Make predictions on the test set

[27]:
```
y_pred = knn.predict(X_test)
y_pred
```

[27]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0])
```

0.2.9 Evaluate performance of the model

[51]:
```
from sklearn.metrics import confusion_matrix

print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)

class_names = iris.target_names
print("\nConfusion Matrix (labeled):")
print(pd.DataFrame(cm, index=class_names, columns=class_names))
```

```
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Confusion Matrix (labeled):
```

```
            setosa  versicolor  virginica
setosa          19           0          0
versicolor       0          13          0
virginica        0           0         13
```

The confusion_matrix provides a summary of prediction results, comparing the predicted values (y_pred) with the actual values (y_test). It's especially useful for classification tasks as it shows how many instances were correctly or incorrectly classified in each category.

---

[30]:
```
from sklearn.metrics import classification_report

print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

The classification_report provides detailed metrics for each class, giving a better understanding of model performance across categories.

---

[52]:
```
from sklearn.metrics import accuracy_score

print("Accuracy Score:")
print(accuracy_score(y_test, y_pred))
```

```
Accuracy Score:
1.0
```

The accuracy_score metric calculates the proportion of correct predictions over the total number of predictions, giving an overall measure of model performance.

0.2.10 Example

[55]:
```
import numpy as np

new_data = np.array([[5.1, 3.5, 1.4, 0.2]])  # Example data (Sepal Length,␣
↪Sepal Width, Petal Length, Petal Width)
new_data_scaled = scaler.transform(new_data)  # Scale the new data
prediction = knn.predict(new_data_scaled)
print("\nPrediction for new data:", iris.target_names[prediction][0])
```

```
Prediction for new data: setosa
```