task 1 jn

May 21, 2023

[ ]:
```
# Import NumPy library with alias 'np'.
import numpy as np

# Import Matplotlib's pyplot module with alias 'plt'.
import matplotlib.pyplot as plt

# Import Seaborn library.
import seaborn as sns

# Import Pandas library with alias 'pd'.
import pandas as pd

# Display Matplotlib plots directly in Jupyter notebook output.
%matplotlib inline
```

```
<frozen importlib._bootstrap>:228: RuntimeWarning:
scipy._lib.messagestream.MessageStream size changed, may indicate binary
incompatibility. Expected 56 from C header, got 64 from PyObject
```

[ ]:
```
# Define the attribute names as a list.
attributes = ["sepal_length", "sepal_width", "petal_length", "petal_width",␣
 ↪"class"]

# Load the Iris dataset from the UCI Machine Learning Repository as a Pandas␣
 ↪dataframe.
iris_df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/
 ↪iris/iris.data', header=None)

# Assign the attribute names to the columns of the Iris dataframe.
iris_df.columns = attributes

# Print the first few rows of the Iris dataframe to ensure it has been loaded␣
 ↪correctly.
iris_df.head()
```

[ ]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa

1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

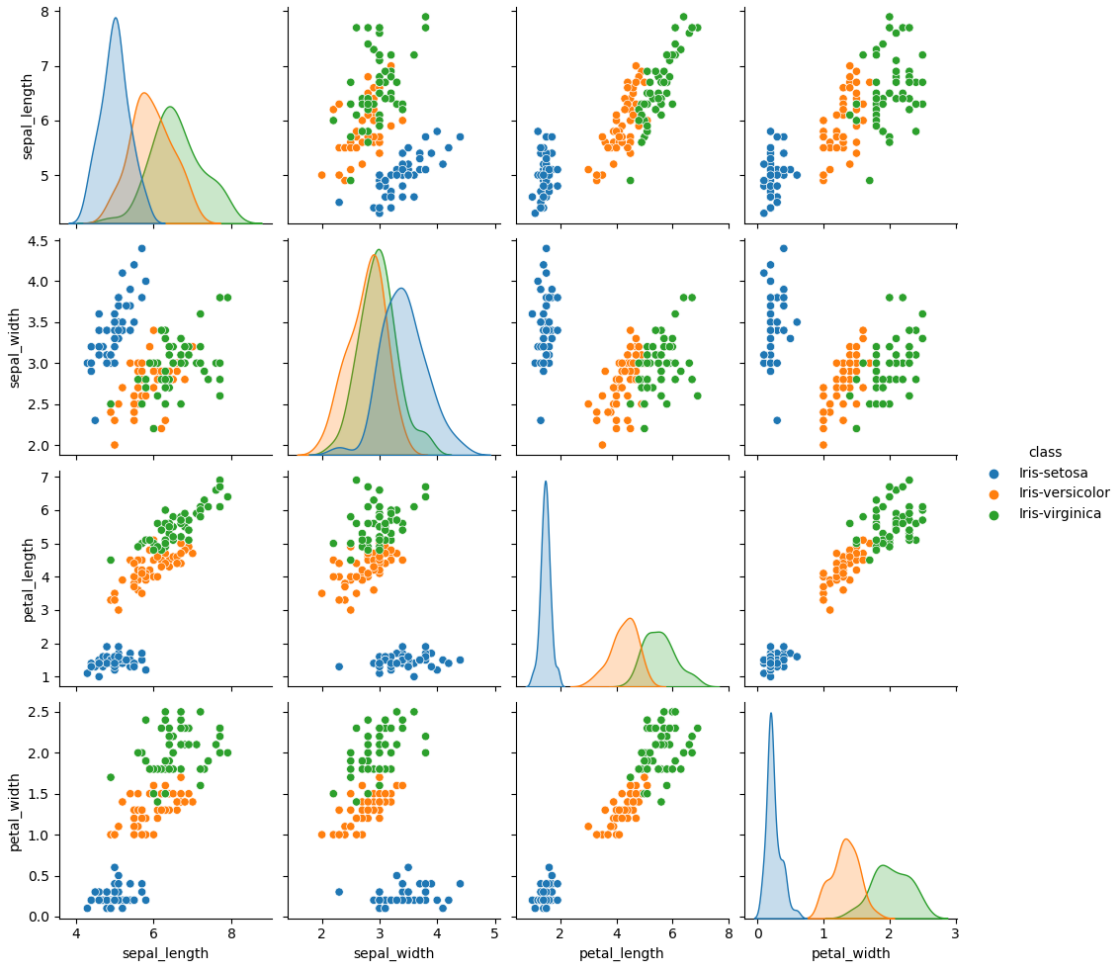
[ ]:
```
# Print a summary of statistical information about the Iris dataframe using␣
↪describe().
iris_df.describe()
```

[ ]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

[ ]:
```
# Create a pairplot using Seaborn to visualize relationships between pairs of␣
↪attributes in the Iris dataframe.
# The 'hue' parameter is set to 'class', which colors the data points according␣
↪to their class label.
sns.pairplot(iris_df, hue='class')
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f152022cf70>
```



[ ]:
```
# Convert the Iris dataframe to a NumPy array using the 'values' attribute.
data = iris_df.values

# Select the first four columns of the data as the input features (X).
X = data[:,0:4]

# Select the last column of the data as the target variable (Y).
Y = data[:,4]
```

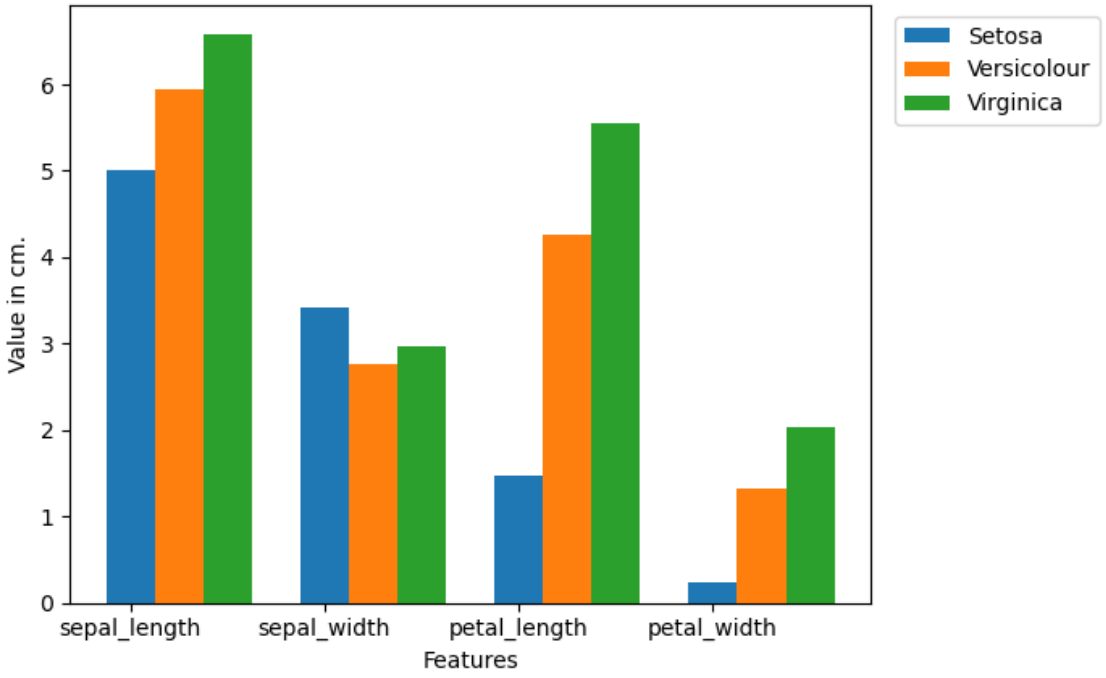
[ ]:
```
# Calculate the average of each feature for all classes using NumPy.
Y_Data = np.array([np.average(X[:, i][Y==j].astype('float32')) for i in range␣
 ↪(X.shape[1]) for j in (np.unique(Y))])

# Reshape the result into a 2D array of shape (3,4) and swap the rows and␣
 ↪columns.
Y_Data_reshaped = Y_Data.reshape(4, 3)
Y_Data_reshaped = np.swapaxes(Y_Data_reshaped, 0, 1)
```

```
# Set up the x-axis values for the bar chart using NumPy arange().
X_axis = np.arange(len(attributes)-1)

# Set the width of the bars in the bar chart to 0.25.
width = 0.25
```

[ ]:
```
# Plot the average for each feature using Matplotlib's bar chart function.
# Three sets of bars will be plotted side by side, one for each class of iris.
# The x-axis values are set using X_axis, the width of the bars is set to␣
 ↪'width'.
# The tick labels for the x-axis are set to the first four elements of␣
 ↪'attributes'.
# Labels for the x- and y-axes are added, as well as a legend in the upper␣
 ↪right corner of the plot.
plt.bar(X_axis, Y_Data_reshaped[0], width, label = 'Setosa')
plt.bar(X_axis+width, Y_Data_reshaped[1], width, label = 'Versicolour')
plt.bar(X_axis+width*2, Y_Data_reshaped[2], width, label = 'Virginica')
plt.xticks(X_axis, attributes[:4])
plt.xlabel("Features")
plt.ylabel("Value in cm.")
plt.legend(bbox_to_anchor=(1.3,1))
plt.show()
```



[ ]:
```
# Split the dataset into training and testing sets using Scikit-learn's train_test_split function.
# The input features are X, the target variable is Y, and the test size is set to 0.2 (20% of the data).
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
```

[ ]:
```
# Train a Support Vector Machine (SVM) classifier using Scikit-learn's SVC function.
# The SVM is initialized with default parameters, which use a radial basis function (RBF) kernel by default.
# The classifier is trained on the training data using the fit method.
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, y_train)
```

[ ]:
```
SVC()
```

[ ]:
```
# Use the trained SVM classifier to make predictions on the test data.
predictions = svm.predict(X_test)

# Calculate the accuracy of the SVM classifier using Scikit-learn's accuracy_score function.
# The accuracy is calculated by comparing the predicted class labels with the true class labels from the test data.
from sklearn.metrics import accuracy_score
accuracy_score(y_test, predictions)
```

[ ]:
```
1.0
```

[ ]:
```
# Calculate a comprehensive classification report for the SVM classifier using Scikit-learn's classification_report function.
# The classification report includes precision, recall, F1-score, and support metrics for each class label.
from sklearn.metrics import classification_report
print(classification_report(y_test, predictions))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      1.00      1.00        12
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

[ ]:
```
# Create a new NumPy array called X_new with 3 rows, each representing a new␣
 ↪data point to predict.
X_new = np.array([[3, 2, 1, 0.2], [4.9, 2.2, 3.8, 1.1], [5.3, 2.5, 4.6, 1.9]])

# Use the trained SVM classifier to make predictions on the new data points in␣
 ↪X_new.
prediction = svm.predict(X_new)

# Print the predicted species for the new data points.
print("Prediction of Species: {}".format(prediction))
```

```
Prediction of Species: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

[ ]:
```
# Save the trained SVM classifier to a file using Python's pickle module.
# The SVM classifier is saved to a file called 'SVM.pickle' using the 'wb' mode␣
 ↪for writing binary data.
import pickle
with open('SVM.pickle', 'wb') as f:
    pickle.dump(svm, f)

# Load the saved SVM model from the file using Python's pickle module.
# The contents of the file are loaded into a new object called 'model'.
with open('SVM.pickle', 'rb') as f:
    model = pickle.load(f)

# Use the loaded SVM model to make predictions on new data points.
# The predictions should be the same as those made earlier using the original␣
 ↪SVM classifier.
model.predict(X_new)
```

[ ]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```