d55kchfdz

December 15, 2023

Template Notebook:

- Please change this notebook name as "Lastname_Firstname_FinalExam.ipynb"
- You can add/delete/modify the cells accordingly
- Please make sure your print your outputs in each question
- Try to attempt all the questions for partial credits

##Importing the libraries

[18]:
```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
```

[19]:
```
pip install scikit-optimize
```

```
Requirement already satisfied: scikit-optimize in
/usr/local/lib/python3.10/dist-packages (0.9.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.3.2)
Requirement already satisfied: pyaml>=16.9 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (23.9.7)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.23.5)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.11.4)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from pyaml>=16.9->scikit-optimize) (6.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->scikit-
optimize) (3.2.0)
```

[20]:
```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
```

Question 1: Increasing Training Set Size Experiment

[21]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score
import matplotlib.pyplot as plt
```

##Loading the data

[22]:
```python
iris = load_iris()
```

##Data Exploration

[23]:
```python
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target
print(iris_df.head())
```

```
   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

   target
0       0
1       0
2       0
3       0
4       0
```

Head() funtion is used to retrieve first 5 rows of the data.

[24]:
```python
#Showing Last 5 rows
iris_df.tail()
```

[24]:
```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
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

     target
145       2
146       2
147       2
148       2
149       2
```

tail() is used to retrieve last 5 rows of the data.

[25]:
```
#Finding the size of the dataset
iris_df.shape
```

[25]:
```
(150, 5)
```

Dataset has 150 rows and 5 columns.

[26]:
```
#Finding Duplicate values
iris_df.duplicated()
```

[26]:
```
0      False
1      False
2      False
3      False
4      False
       …
145    False
146    False
147    False
148    False
149    False
Length: 150, dtype: bool
```

Observation - No duplicates found

[27]:
```
iris_df.describe(include='all')
```

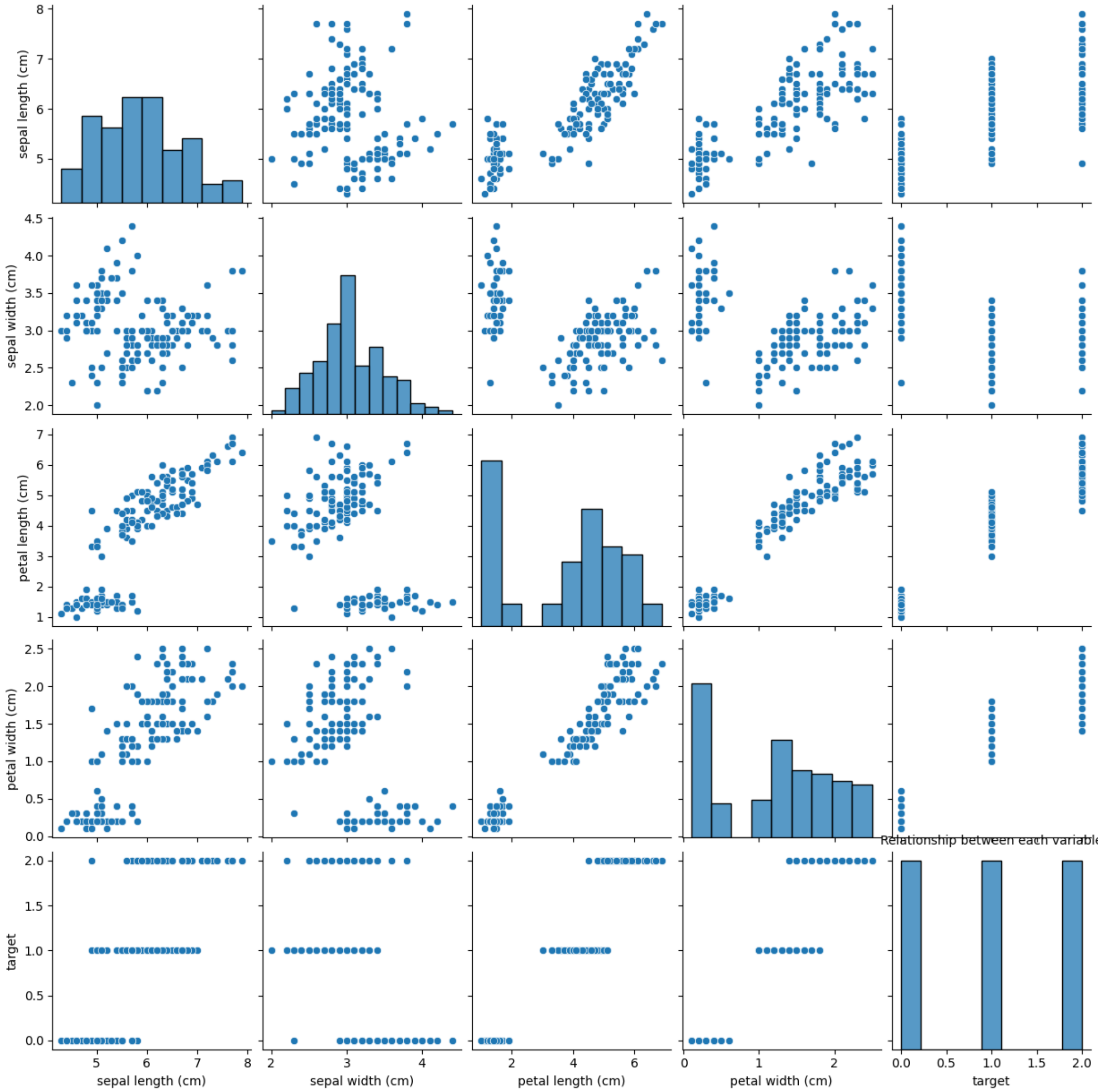
[27]:
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
```

```
max              7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

From this description, we can see all the descriptions about the data, like average length and width, minimum value, maximum value, the 25%, 50%, and 75% distribution value, etc.

##Relationship between each variable

[28]:
```python
import seaborn as sns
sns.pairplot(data=iris_df)
plt.title('Relationship between each variable', fontsize=10)

# Show the plot
plt.show()
```



##Let's split train and test data

[29]:
```python
# dividing the dataset in training and testing set

from sklearn.model_selection import train_test_split
iris = load_iris()
X, y = iris.data, iris.target

# Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# determining the shapes of training and testing sets
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(120, 4)
(120,)
(30, 4)
(30,)
```

##Model Building

[30]:
```
#Defining models
models = {
    'lr': LogisticRegression(),
    'dt': DecisionTreeClassifier(),
    'rf': RandomForestClassifier(),
    'nb': GaussianNB()
}
```

[31]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score

# Create dictionaries to store results
accuracy_results = {name: [] for name in models}
f1_results = {name: [] for name in models}

# Create a list to store training sizes
training_sizes = []

# Loop through different training sizes
for i in range(1, 21):
    # Calculate the training size percentage
    training_size_percentage = i * 5
    training_size = int(len(X_train) * (training_size_percentage / 100))

    # Train the models with the current training size
    for name, clf in models.items():
        clf.fit(X_train[:training_size], y_train[:training_size])
        y_pred = clf.predict(X_test)

        # Calculate accuracy and F1-score
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')

        # Store results
        accuracy_results[name].append(accuracy)
        f1_results[name].append(f1)
```

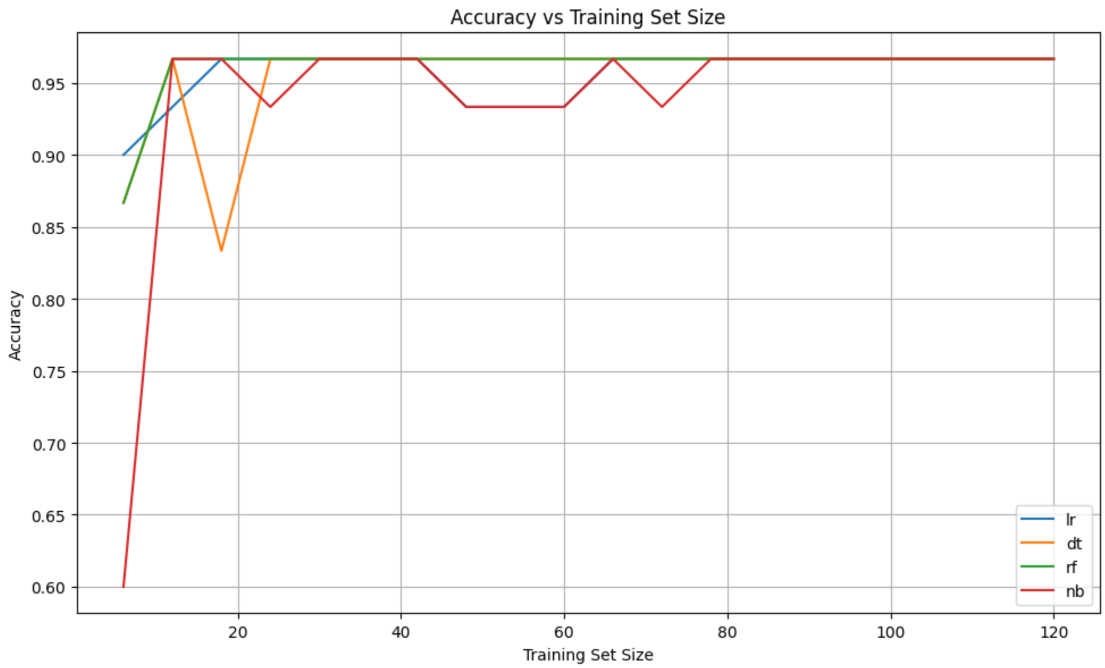
```
    # Store the current training size
    training_sizes.append(training_size)
```

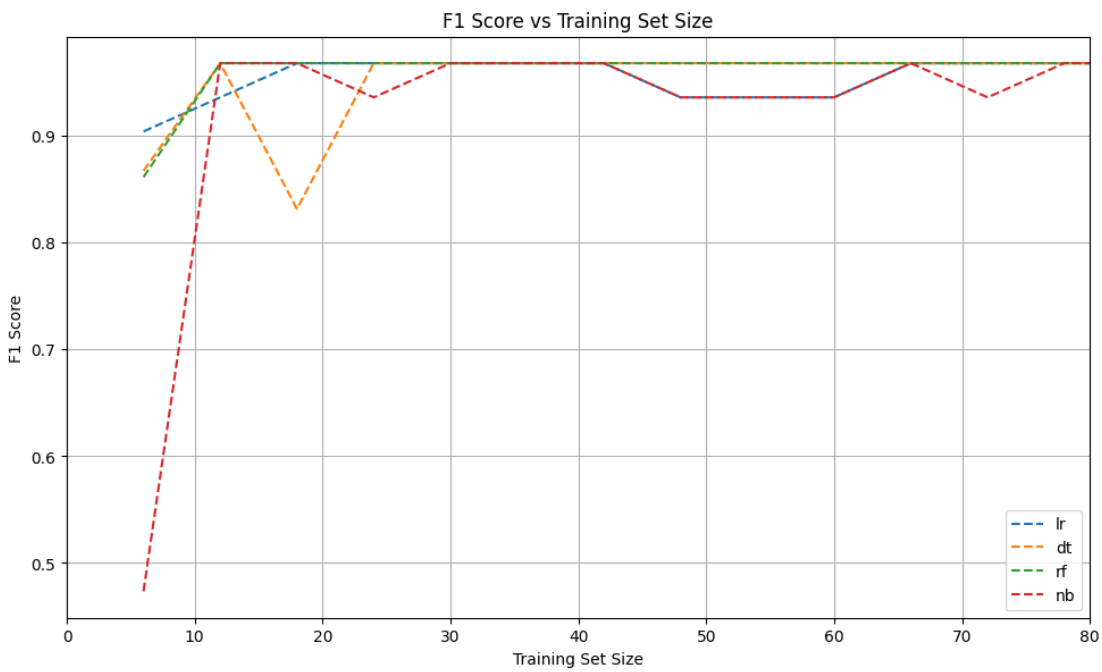
```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[32]:

```
# Plot the results
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 12))
```

```
for name, accuracies in accuracy_results.items():
    axes[0].plot(training_sizes, accuracies, label=name)

for name, f1_scores in f1_results.items():
    axes[1].plot(training_sizes, f1_scores, label=name, linestyle='dashed')

# Set titles and labels
axes[0].set_title('Accuracy vs Training Set Size')
axes[0].set_xlabel('Training Set Size')
axes[0].set_ylabel('Accuracy')
axes[0].legend()
axes[0].grid(True)

axes[1].set_title('F1 Score vs Training Set Size')
axes[1].set_xlabel('Training Set Size')
axes[1].set_ylabel('F1 Score')
axes[1].legend()
axes[1].grid(True)

plt.xlim(0,80)
plt.tight_layout()
plt.show()
```





Observation - This code generates two plots, one for accuracy and another for the F1 score with shared x-axis and different y-axes.

Question 2: Binary Classification with Discriminant

[33]:
```
X = [5, 1, 9, 6, 5, 6, 1, 9, 10, 11, 8, 7, 13, 8, 19]
Y = [14, 16, 17, 10, 9, 17, 15, 3, 3, 1, 4, 5, 1, 3, 15]
```

```
C = ['c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c2', 'c2', 'c2', 'c2', 'c2',␣
 ↪'c2', 'c2', 'c2']
len(X), len(Y),len(C)
```

[33]: (15, 15, 15)

[34]:
```
discriminant_function = lambda x, y: -x + 2 * y + x * y
```

[42]:
```
# Apply the discriminant function to each data point
discriminant_values = [discriminant_function(x, y) for x, y in zip(X, Y)]

# Predict class labels based on the discriminant values and threshold (35)
predicted_labels = ['c1' if value >= 35 else 'c2' for value in␣
 ↪discriminant_values]

# Count misclassifications for c1 and c2
misclassified_c1 = sum((true_label == 'c1' and pred_label == 'c2') for␣
 ↪true_label, pred_label in zip(C, predicted_labels))
misclassified_c2 = sum((true_label == 'c2' and pred_label == 'c1') for␣
 ↪true_label, pred_label in zip(C, predicted_labels))

# Print results
print("Predicted Labels:", predicted_labels)
print("Actual Labels   :", C)
print("Misclassified in c1:", misclassified_c1)
print("Misclassified in c2:", misclassified_c2)
```

```
Predicted Labels: ['c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c2', 'c2', 'c2',
'c1', 'c2', 'c2', 'c2', 'c1']
Actual Labels   : ('c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c2', 'c2', 'c2', 'c2',
'c1', 'c2', 'c2', 'c2', 'c2')
Misclassified in c1: 0
Misclassified in c2: 2
```

[39]:
```
print("Predicted Labels:", predicted_labels)
print("Actual Labels   :", C)
print("Accuracy        : {:.2%}".format(accuracy))
print("Misclassified in c1:", misclassified_c1)
print("Misclassified in c2:", misclassified_c2)
```

```
Predicted Labels: ['c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c2', 'c2', 'c2',
'c1', 'c2', 'c2', 'c2', 'c1']
Actual Labels   : ('c1', 'c1', 'c1', 'c1', 'c1', 'c1', 'c2', 'c2', 'c2', 'c2',
'c1', 'c2', 'c2', 'c2', 'c2')
Accuracy        : 96.67%
Misclassified in c1: 0
Misclassified in c2: 2
```

Observation - Number of misclassified in c1 = 1 Number of misclassified in c2 = 7

Question 3: K-Means Clustering

[43]:
```
from sklearn.cluster import KMeans
```

[44]:
```
data_dict = {(2.0, 3.43, 4.37):2, (2.49, 4.28, 4.83):2, (2.58, 4.36, 4.48):2,␣
↪(2.66, 4.45, 5.95):2,
(2.82, 3.66, 4.51): 2, (3.03, 4.37, 5.07): 2, (3.27, 4.54, 4.57): 2, (3.41, 3.
↪94, 5.35): 2,
(3.53, 4.32, 5.41): 2, (3.53, 4.6, 6.8): 1, (3.61, 4.25, 5.21): 1, (3.61, 4.78,␣
↪5.47): 1,
(3.72, 5.44, 5.88): 1, (3.87, 4.96, 4.52): 2, (4.13, 5.29, 6.6): 1, (4.25, 5.
↪97, 5.48): 1,
(4.61, 4.9, 5.11): 1, (4.73, 4.4, 6.78): 1, (4.97, 4.25, 5.0): 1, (4.98, 5.27,␣
↪6.79): 1,
(5.08, 3.51, 4.69): 3, (5.15, 3.58, 4.2): 3, (5.67, 2.27, 4.65): 3, (5.67, 3.
↪81, 5.75): 3,
(5.94, 2.34, 4.12): 3, (6.06, 3.16, 4.36): 3, (6.09, 3.19, 4.02): 3, (6.43, 3.
↪42, 4.18): 3,
(6.56, 2.7, 4.03): 3, (6.79, 3.46, 4.81): 3}
```

[45]:
```
for key, value in data_dict.items():
    print(key,value)
```

```
(2.0, 3.43, 4.37) 2
(2.49, 4.28, 4.83) 2
(2.58, 4.36, 4.48) 2
(2.66, 4.45, 5.95) 2
(2.82, 3.66, 4.51) 2
(3.03, 4.37, 5.07) 2
(3.27, 4.54, 4.57) 2
(3.41, 3.94, 5.35) 2
(3.53, 4.32, 5.41) 2
(3.53, 4.6, 6.8) 1
(3.61, 4.25, 5.21) 1
(3.61, 4.78, 5.47) 1
(3.72, 5.44, 5.88) 1
(3.87, 4.96, 4.52) 2
(4.13, 5.29, 6.6) 1
(4.25, 5.97, 5.48) 1
(4.61, 4.9, 5.11) 1
(4.73, 4.4, 6.78) 1
(4.97, 4.25, 5.0) 1
(4.98, 5.27, 6.79) 1
(5.08, 3.51, 4.69) 3
(5.15, 3.58, 4.2) 3
(5.67, 2.27, 4.65) 3
```

```
(5.67, 3.81, 5.75) 3
(5.94, 2.34, 4.12) 3
(6.06, 3.16, 4.36) 3
(6.09, 3.19, 4.02) 3
(6.43, 3.42, 4.18) 3
(6.56, 2.7, 4.03) 3
(6.79, 3.46, 4.81) 3
```

[46]:
```
true_labels = list(data_dict.values())

# Convert data_dict keys to a list of tuples
data_points = list(data_dict.keys())

# Centers for K-Means clustering
centers_dict = {
    (3, 4, 5): 1,
    (4, 5, 6): 2,
    (6, 3, 5): 3
}

# Convert centers_dict keys to a list of tuples
centers = list(centers_dict.keys())

# Applying K-Means clustering
kmeans = KMeans(n_clusters=len(centers), init=centers, n_init=1,
 ↪random_state=42)
kmeans.fit(data_points)

#predicted labels
predicted_labels = [label + 1 for label in kmeans.labels_]  # Adding 1 to match
 ↪the class labels (1, 2, 3)

# Print the number of correctly classified instances
correctly_classified = sum(label == true_label for label, true_label in
 ↪zip(predicted_labels, true_labels))
print("Number of correctly classified instances:", correctly_classified)
```

```
Number of correctly classified instances: 11
```

Observation - Based on the above discriminant functions, the number of correctly classified instances are 11.