In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
import sys
sys.path.append('drive/MyDrive/USC Spring 2024/Machine Learning/Project/')
```

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score, confusion_matrix
import matplotlib.pyplot as plt
```

1. Loading the data

In [4]:
```
def load_data(filename):
    #Loading the file into data
    data = np.array(pd.read_csv(filename))
    return data
```

In [5]:
```
drybeans_dataset_train_filename = 'drive/MyDrive/USC Spring 2024/Machine Learning/Project/dry_bean_classification_train.csv'
drybeans_dataset_test_filename = 'drive/MyDrive/USC Spring 2024/Machine Learning/Project/dry_bean_classification_test.csv'
```

Splitting the data

In [6]:
```
drybeans_dataset_train = load_data(drybeans_dataset_train_filename)
#Splitting into X and y dataset
drybeans_dataset_train_X = drybeans_dataset_train[:, :-1]
drybeans_dataset_train_y = drybeans_dataset_train[:, -1]

drybeans_dataset_test = load_data(drybeans_dataset_test_filename)
drybeans_dataset_test_X = drybeans_dataset_test[:, :-1]
drybeans_dataset_test_y = drybeans_dataset_test[:, -1]
```

In [8]:
```
print(drybeans_dataset_train_X.shape)
```

```
(10889, 16)
```

In [ ]:
```
class_labels_dictionary = {'SEKER': 0, 'BARBUNYA': 1, 'DERMASON': 2, 'CALI': 3, 'HOROZ': 4, 'SIRA': 5, 'BOMBAY': 6}
class_labels = [0, 1, 2, 3, 4, 5, 6]
train_class_labels = [class_labels_dictionary[drybeans_dataset_train_y[i]] for i in range(len(drybeans_dataset_train_y))]
test_class_labels = [class_labels_dictionary[drybeans_dataset_test_y[i]] for i in range(len(drybeans_dataset_test_y))]
```

2. Important Functions for the Nearest Means Classifier

2.1. General Functions

In [ ]:
```
def calculate_class_means(dataset, labels):
    classes = np.unique(labels)
    num_classes = len(classes)
    num_features = dataset.shape[1]

    num_class_datapoints = np.zeros_like(classes)
    class_means = np.zeros((num_classes, num_features))

    for i in range(len(dataset)):
        for j in range(len(classes)):
            if labels[i] == classes[j]:
                class_means[j, :] = class_means[j, :] + dataset[i]
                num_class_datapoints[j] += 1

    for i in range(len(class_means)):
        class_means[i, :] /= num_class_datapoints[i]

    return class_means
```

In [ ]:
```
def predict_labels(dataset, means):
    labels = np.empty((dataset.shape[0],))

    for i in range(len(dataset)):
        distances = []
        for j in range(len(means)):
            distance = -np.linalg.norm(dataset[i]-means[j])
            distances.append(distance)

        label = np.argmax(distances)
        labels[i] = label

    return labels
```

In [ ]:
```
def calculate_accuracy(actual, predicted):
    return 100*np.sum(actual == predicted)/len(actual)
```

In [ ]:
```
def compute_F1_macro(actual, predicted):
    return f1_score(y_true=actual, y_pred=predicted, average='macr
o')

def compute_F1_weighted(actual, predicted):
    return f1_score(y_true=actual, y_pred=predicted, average='weigh
ted')
```

In [ ]:
```
def create_confusion_matrix(actual, predicted, labels):
    return confusion_matrix(y_true=actual, y_pred=predicted, labels
=labels)
```

2.2. Feature Engineering

In [ ]:
```
def standardize_data_train(dataset):
    standardized_data = dataset.copy()
    means = []
    standard_devs = []

    for i in range(dataset.shape[1]):
        mean = np.mean(dataset[:, i])
        std_dev = np.std(dataset[:, i])

        standardized_data[:, i] = (dataset[:, i]-mean)/std_dev
        means.append(mean)
        standard_devs.append(std_dev)

    return standardized_data, means, standard_devs
```

In [ ]:
```
def standardize_data_test(dataset, means, standard_devs):
    standardized_data = dataset.copy()

    for i in range(dataset.shape[1]):
        standardized_data[:, i] = (dataset[:, i]-means[i])/standard
_devs[i]

    return standardized_data
```

3. Evaluation

3.1. Evaluation Using Nearest Means Classifier on Unstandardized Data without Removal of Features

In [ ]:
```
class_means_unstandardized = calculate_class_means(drybeans_dataset
_train_X, train_class_labels)
predicted_labels_test_unstandardized = predict_labels(drybeans_data
set_test_X, class_means_unstandardized)

test_accuracy_unstandardized = calculate_accuracy(test_class_label
s, predicted_labels_test_unstandardized)
print('Testing Accuracy for unstandardized data without feature sel
ection:', test_accuracy_unstandardized, '%')
```

```
Testing Accuracy for unstandardized data without feature selection:
61.86627479794269 %
```

In [ ]:
```
F1_macro_NMC_unstandardized = compute_F1_macro(test_class_labels, p
redicted_labels_test_unstandardized)
F1_weighted_NMC_unstandardized = compute_F1_weighted(test_class_lab
els, predicted_labels_test_unstandardized)

print('Macro-Averaged F1 Score =', F1_macro_NMC_unstandardized)
print('Weighted-Averaged F1 Score =', F1_weighted_NMC_unstandardize
d)
```

```
Macro-Averaged F1 Score = 0.6348322573531793
Weighted-Averaged F1 Score = 0.6262446392735608
```

In [ ]:
```
confusion_matrix_unstandardized = create_confusion_matrix(test_clas
s_labels, predicted_labels_test_unstandardized, class_labels)
print(confusion_matrix_unstandardized)
```

```
[[210   0  82   0  15  89   0]
 [  1 106   0  98  40   9   0]
 [170   0 561   0   0   0   0]
 [  0 113   0 189  12   0   0]
 [ 27  51   1   4 254  69   0]
 [152   0  10   0  95 256   0]
 [  0   0   0   0   0   0 108]]
```

3.2. Evaluation Using Nearest Means Classifier on Standardized Data without Removal of Features

In [ ]:
```
standardized_drybeans_dataset_train_X, feature_means, feature_stand
ard_devs = standardize_data_train(drybeans_dataset_train_X)
class_means_standardized = calculate_class_means(standardized_drybe
ans_dataset_train_X, train_class_labels)

standardized_drybeans_dataset_test_X = standardize_data_test(drybea
ns_dataset_test_X, feature_means, feature_standard_devs)
predicted_labels_test_standardized = predict_labels(standardized_dr
ybeans_dataset_test_X, class_means_standardized)

test_accuracy_standardized = calculate_accuracy(test_class_labels,
predicted_labels_test_standardized)
print('Testing Accuracy for standardized data without feature selec
tion:', test_accuracy_standardized, '%')
```

```
Testing Accuracy for standardized data without feature selection: 8
8.79500367376929 %
```

In [ ]:
```
F1_macro_NMC_standardized = compute_F1_macro(test_class_labels, pre
dicted_labels_test_standardized)
F1_weighted_NMC_standardized = compute_F1_weighted(test_class_label
s, predicted_labels_test_standardized)

print('Macro-Averaged F1 Score =', F1_macro_NMC_standardized)
print('Weighted-Averaged F1 Score =', F1_weighted_NMC_standardized)
```

```
Macro-Averaged F1 Score = 0.904689289004804
Weighted-Averaged F1 Score = 0.8886337990050704
```

In [ ]:
```
confusion_matrix_standardized = create_confusion_matrix(test_class_
labels, predicted_labels_test_standardized, class_labels)

print(confusion_matrix_standardized)
```

```
[[367   2  11   0   0  16   0]
 [  2 207   0  22   3  20   0]
 [ 22   0 611   0   2  96   0]
 [  0  11   0 295   5   3   0]
 [  0   0   0  17 383   6   0]
 [  4   2  45   1  15 446   0]
 [  0   0   0   0   0   0 108]]
```