นาย สัณหณัฐ พรมจรรย์ 63070501069

1) Source Code

Link: https://colab.research.google.com/drive/1tDYw-kmhs1Axc6g1Ey4bQKBzDqM68Vu1?usp=sharing

```
[ ]  import cv2 as cv
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.neighbors import KNeighborsClassifier
     from scipy.spatial.distance import cdist
```

```
[ ]  # Define means and standard deviations for red, blue, and yellow classes
     number_points = 50

     red_mean = (20, 30)
     red_std = [[3, 0], [0, 10]]
     red = np.random.multivariate_normal(red_mean, red_std, number_points)

     blue_mean = (40, 40)
     blue_std = [[10, 0], [0, 10]]
     blue = np.random.multivariate_normal(blue_mean, blue_std, number_points)

     yellow_mean = (50, 40)
     yellow_std = [[15, 0], [0, 15]]
     yellow = np.random.multivariate_normal(yellow_mean, yellow_std, number_points)
```

```
[ ]  # Plot the 3 classes using the training data
     # matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, ...)
     fig = plt.figure(figsize=(8, 8)) #8 inch by 8 inch plot size
     plt.xlim(10, 60) #x min and max
     plt.ylim(10, 60) #y min and max
     plt.scatter(red[:, 0],red[:, 1], color='red', label='Red')
     plt.scatter(blue[:,0],blue[:,1],color='blue',label='Blue')
     plt.scatter(yellow[:,0],yellow[:,1],color='yellow',label='Yellow')
     plt.show()
```

```
[ ] # Combine the points into a single dataset
    X = np.concatenate((red, blue, yellow), axis=0)

    # Assign labels to each point
    Y = np.concatenate((np.zeros(50), np.ones(50), 2 * np.ones(50)))

    # Split the dataset into training data (70%) and testing data (30%)
    training_features, testing_features, training_labels, testing_labels = train_test_split(X, Y, test_size=0.3, random_state=0)

    X_train = training_features.astype(np.float32)
    Y_train = training_labels.astype(np.int32)
    X_test = testing_features.astype(np.float32)
    Y_test = testing_labels.astype(np.float32)
```

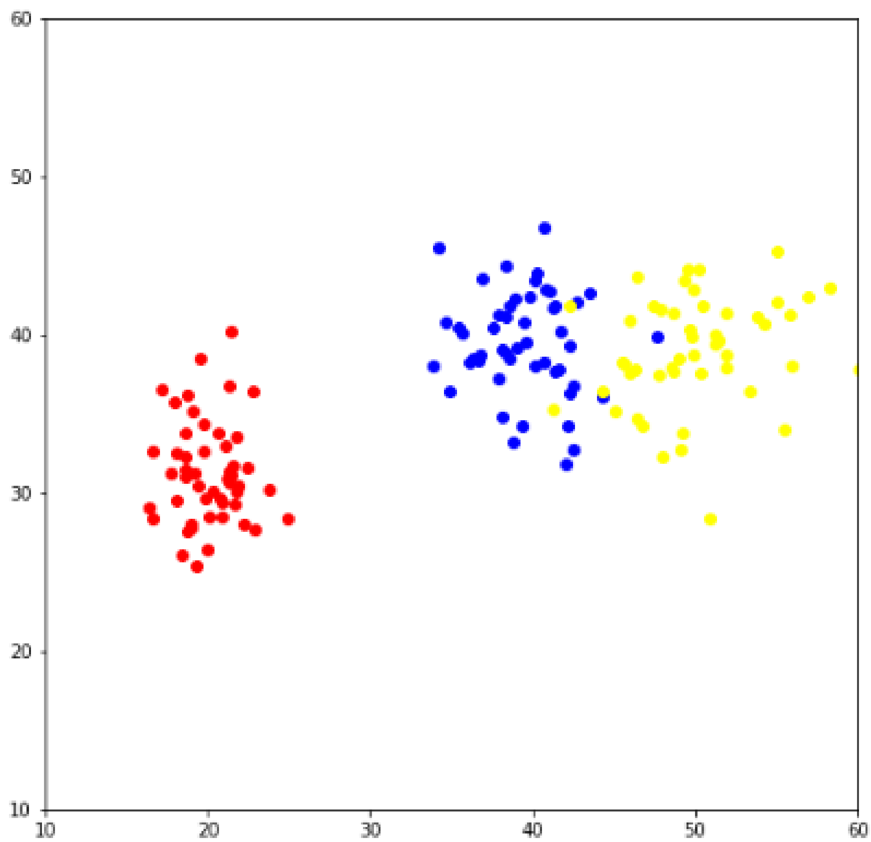
```
[ ] # Train the KNN classifier
    knn = cv.ml.KNearest_create()
    knn.train(X_train, cv.ml.ROW_SAMPLE, Y_train)

    # Using KNN with K = 5, Report the total accuracy of the testing data
    ret, results, neighbours, dist = knn.findNearest(X_test, 5)
    correct = np.count_nonzero(results == Y_test)
    accuracy = correct / float(X_test.shape[0])
    print("Total Accuracy Using KNN: %5.2f %%" % accuracy)
```

```
[ ] # Calculate the mean of each class in the training data
    mean_red_train = np.mean(X_train[Y_train == 0], axis=0)
    mean_blue_train = np.mean(X_train[Y_train == 1], axis=0)
    mean_yellow_train = np.mean(X_train[Y_train == 2], axis=0)

    # Report the training data cluster mean for each class of red, blue, and yellow.
    print("Mean of Red class in training data:", mean_red_train)
    print("Mean of Blue class in training data:", mean_blue_train)
    print("Mean of Yellow class in training data:", mean_yellow_train)

    # Calculate the distance to each class mean
    dist_red = cdist(X_test, [mean_red_train])
    dist_blue = cdist(X_test, [mean_blue_train])
    dist_yellow = cdist(X_test, [mean_yellow_train])

    predicted = np.argmin(np.hstack([dist_red, dist_blue, dist_yellow]), axis=1)

    # Calculate the accuracy of the classifier
    # Report the total accuracy of the testing data using this classifier
    accuracy = 100*np.mean(predicted == Y_test)
    print("Total accuracy of the testing data using the minimum distance classifier: %5.2f %% " % accuracy)
```

Output

1.1)

```
[3]  # Plot the 3 classes using the training data
     # matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, ...)
     fig = plt.figure(figsize=(8, 8)) #8 inch by 8 inch plot size
     plt.xlim(10, 60) #x min and max
     plt.ylim(10, 60) #y min and max
     plt.scatter(red[:, 0],red[:, 1], color='red', label='Red')
     plt.scatter(blue[:,0],blue[:,1],color='blue',label='Blue')
     plt.scatter(yellow[:,0],yellow[:,1],color='yellow',label='Yellow')
     plt.show()
```



1.2)

```
[6] # Train the KNN classifier
    knn = cv.ml.KNearest_create()
    knn.train(X_train, cv.ml.ROW_SAMPLE, Y_train)

    # Using KNN with K = 5, Report the total accuracy of the testing data
    ret, results, neighbours, dist = knn.findNearest(X_test, 5)
    correct = np.count_nonzero(results == Y_test)
    accuracy = correct / float(X_test.shape[0])
    print("Total Accuracy Using KNN: %5.2f %%" % accuracy)
```

```
Total Accuracy Using KNN: 15.27 %
```

1.3)

```
[7] # Calculate the mean of each class in the training data
    mean_red_train = np.mean(X_train[Y_train == 0], axis=0)
    mean_blue_train = np.mean(X_train[Y_train == 1], axis=0)
    mean_yellow_train = np.mean(X_train[Y_train == 2], axis=0)

    # Report the training data cluster mean for each class of red, blue, and yellow.
    print("Mean of Red class in training data:", mean_red_train)
    print("Mean of Blue class in training data:", mean_blue_train)
    print("Mean of Yellow class in training data:", mean_yellow_train)

    # Calculate the distance to each class mean
    dist_red = cdist(X_test, [mean_red_train])
    dist_blue = cdist(X_test, [mean_blue_train])
    dist_yellow = cdist(X_test, [mean_yellow_train])

    predicted = np.argmin(np.hstack([dist_red, dist_blue, dist_yellow]), axis=1)

    # Calculate the accuracy of the classifier
    # Report the total accuracy of the testing data using this classifier
    accuracy = 100*np.mean(predicted == Y_test)
    print("Total accuracy of the testing data using the minimum distance classifier: %5.2f %% " % accuracy)
```

```
Mean of Red class in training data: [20.06521  31.035688]
Mean of Blue class in training data: [39.119057 40.22529 ]
Mean of Yellow class in training data: [49.790318 38.7916  ]
Total accuracy of the testing data using the minimum distance classifier: 97.78 %
```

2) Source Code

Link: https://colab.research.google.com/drive/1bcBKtD-kecEFge4nSWKS4hAC_QcQ_Y9c?usp=sharing

```
[1] import numpy as np
    import pandas as pd
    from sklearn.datasets import load_iris
    from sklearn.model_selection import train_test_split
    from sklearn.naive_bayes import GaussianNB
```

```
[2] # Get the iris dataset and split it into data features and target values
    iris = load_iris()
    X = iris.data
    Y = iris.target

    # Spliting our dataset into 2 parts for training and testing
    training_features, testing_features, training_labels, testing_labels  = train_test_split(X, Y, test_size=0.3, random_state=0)
```

```
[3] # Find the mean and standard deviation for each of the 4 features of each of the 3 classes from the training data.
    class_means = np.zeros((3, 4))
    class_stds = np.zeros((3, 4))
    for i in range(3):
        features_class = training_features[training_labels == i]
        class_means[i] = np.mean(features_class, axis=0)
        class_stds[i] = np.std(features_class, axis=0)

    # Show mean and standard deviation for each feature of each class
    for i in range(3):
        print("Class:", i+1)
        for j in range(4):
            print("Feature %d = Mean:%5.2f , Standard Deviation:%5.2f"%(j+1, class_means[i][j], class_stds[i][j]))
```

[38]
```python
# Find P(c_k) the percent frequency of each class in the training data
# Print P(c_k) the percent frequency of each class in the training data
p_c = np.zeros(3)
for i in range(3):
    p_c[i] = np.mean(training_labels == i)
    print("Percent frequency of class %d in training data: %5.2f" % (i+1, p_c[i]))

# Calculate the pdfs P(x_i | c_k) for each class using the Gaussian distribution equation
p_xc = []
for i in range(3):
    mu = class_means[i]
    sigma = class_stds[i]
    pdf = lambda x: (1/(np.sqrt(2*np.pi)*sigma))*(np.exp(-(x-mu)**2/(2*sigma**2)))
    p_xc.append(pdf)

# Calculate the posterior probabilities P(c_k | x_1, x_2, x_3, x_4) using Bayes' rule
def predict(x):
    p_cx = []
    for i in range(3):
        prior = p_c[i]
        possibility = np.prod([p_xc[i](x[j]) for j in range(4)])
        posterior = possibility * prior
        p_cx.append(posterior)
    return np.argmax(p_cx)

# Test the model on the testing data and display the accuracy
y_pred = [predict(x) for x in testing_features]
accuracy = 100*np.mean(y_pred == testing_labels)
print(f"Accuracy: %5.2f%%" % accuracy)
```

[19]
```python
# Find the class k for which P(c_k | x_1, x_2, x_3, x_4) is maximum
my_predicted_labels = np.zeros(len(testing_features), dtype=int)
for i, x in enumerate(testing_features):

# Calculate the possibility for each class
    possibility = np.zeros(3)
    for j in range(3):
        possibility[j] = np.prod(1 / (np.sqrt(2 * np.pi) * class_stds[j]) * np.exp(-(x - class_means[j])**2 / (2 * class_stds[j]**2)))
    posterior_probs = possibility * p_c

# Print class k for which P(c_k) is maximum
    my_predicted_labels[i] = np.argmax(posterior_probs)
print("Class k for which P(c_k) is maximum: %.f" % my_predicted_labels[i])
```

[27]
```python
# Calculate and Show the accuracy score from your implementation of Naive Bayes from scratch
accuracy = 100*np.mean(my_predicted_labels == testing_labels)
print("Accuracy from scratch: %5.2f%%" % accuracy)
```

[28]
```python
# Use sklearn's GaussianNB classifier to report the accuracy score. Compare result to sklearn's.
gnb = GaussianNB()
gnb.fit(training_features, training_labels)
gnb_pred = gnb.predict(testing_features)
gnb_acc = 100*gnb.score(testing_features, testing_labels)
print("Accuracy from sklearn: %5.2f%%" % gnb_acc)
```

Output

a)

```
[30] # Find the mean and standard deviation for each of the 4 features of each of the 3 classes from the training data.
     class_means = np.zeros((3, 4))
     class_stds = np.zeros((3, 4))
     for i in range(3):
         features_class = training_features[training_labels == i]
         class_means[i] = np.mean(features_class, axis=0)
         class_stds[i] = np.std(features_class, axis=0)

     # Show mean and standard deviation for each feature of each class
     for i in range(3):
         print("Class:", i+1)
         for j in range(4):
             print("Feature %d = Mean:%5.2f , Standard Deviation:%5.2f"%(j+1, class_means[i][j], class_stds[i][j]))
```

```
Class: 1
Feature 1 = Mean: 4.99 , Standard Deviation: 0.35
Feature 2 = Mean: 3.38 , Standard Deviation: 0.39
Feature 3 = Mean: 1.45 , Standard Deviation: 0.14
Feature 4 = Mean: 0.23 , Standard Deviation: 0.09
Class: 2
Feature 1 = Mean: 5.92 , Standard Deviation: 0.53
Feature 2 = Mean: 2.76 , Standard Deviation: 0.34
Feature 3 = Mean: 4.20 , Standard Deviation: 0.48
Feature 4 = Mean: 1.31 , Standard Deviation: 0.20
Class: 3
Feature 1 = Mean: 6.65 , Standard Deviation: 0.65
Feature 2 = Mean: 2.99 , Standard Deviation: 0.35
Feature 3 = Mean: 5.60 , Standard Deviation: 0.56
Feature 4 = Mean: 2.03 , Standard Deviation: 0.26
```

b)

```
[39] # Find P(c_k) the percent frequency of each class in the training data
     # Print P(c_k) the percent frequency of each class in the training data
     p_c = np.zeros(3)
     for i in range(3):
         p_c[i] = np.mean(training_labels == i)
         print("Percent frequency of class %d in training data: %5.2f" % (i+1, p_c[i]))

     # Calculate the pdfs P(x_i | c_k) for each class using the Gaussian distribution equation
     p_xc = []
     for i in range(3):
         mu = class_means[i]
         sigma = class_stds[i]
         pdf = lambda x: (1/(np.sqrt(2*np.pi)*sigma))*(np.exp(-(x-mu)**2/(2*sigma**2)))
         p_xc.append(pdf)

     # Calculate the posterior probabilities P(c_k | x_1, x_2, x_3, x_4) using Bayes' rule
     def predict(x):
         p_cx = []
         for i in range(3):
             prior = p_c[i]
             possibility = np.prod([p_xc[i](x[j]) for j in range(4)])
             posterior = possibility * prior
             p_cx.append(posterior)
         return np.argmax(p_cx)

     # Test the model on the testing data and display the accuracy
     y_pred = [predict(x) for x in testing_features]
     accuracy = 100*np.mean(y_pred == testing_labels)
     print(f"Accuracy: %5.2f%%" % accuracy)
```

```
Percent frequency of class 1 in training data:  0.32
Percent frequency of class 2 in training data:  0.30
Percent frequency of class 3 in training data:  0.37
Accuracy: 24.44%
```

c)

```
[40] # Find the class k for which P(c_k | x_1, x_2, x_3, x_4) is maximum
     my_predicted_labels = np.zeros(len(testing_features), dtype=int)
     for i, x in enumerate(testing_features):

     # Calculate the possibility for each class
       possibility = np.zeros(3)
       for j in range(3):
         possibility[j] = np.prod(1 / (np.sqrt(2 * np.pi) * class_stds[j]) * np.exp(-(x - class_means[j])**2 / (2 * class_stds[j]**2)))
       posterior_probs = possibility * p_c

     # Print class k for which P(c_k) is maximum
       my_predicted_labels[i] = np.argmax(posterior_probs)
     print("Class k for which P(c_k) is maximum: %.f" % my_predicted_labels[i])
```

```
Class k for which P(c_k) is maximum: 0
```

d)

```
[41] # Calculate and Show the accuracy score from your implementation of Naive Bayes from scratch
     accuracy = 100*np.mean(my_predicted_labels == testing_labels)
     print("Accuracy from scratch: %5.2f%%" % accuracy)
```

```
Accuracy from scratch: 100.00%
```

e)

```
[42] # Use sklearn's GaussianNB classifier to report the accuracy score. Compare result to sklearn's.
     gnb = GaussianNB()
     gnb.fit(training_features, training_labels)
     gnb_pred = gnb.predict(testing_features)
     gnb_acc = 100*gnb.score(testing_features, testing_labels)
     print("Accuracy from sklearn: %5.2f%%" % gnb_acc)
```

```
Accuracy from sklearn: 100.00%
```

3) Source Code

Link:

https://colab.research.google.com/drive/1afEbVpRM169k8QOknI6yLXEom_NPqnIi?usp=sharing

```
[ ] from sklearn.datasets import load_iris, load_breast_cancer, load_digits
    from sklearn.model_selection import train_test_split
    from sklearn.naive_bayes import GaussianNB
    from sklearn.metrics import accuracy_score
    from sklearn.neighbors import KNeighborsClassifier
```

```
[ ] # Get user input for dataset selection
    dataset = 'none'
    while dataset not in {'iris', 'cancer','digits'}:
      dataset = input("Enter the name of the dataset (iris, cancer, digits): ")
      if dataset not in {'iris', 'cancer','digits'}:
        print("Invalid response: '%s'. Please try again." % dataset)
```

```
[ ] if dataset == 'iris':
      data = load_iris()
    elif dataset == "cancer":
      data = load_breast_cancer()
    elif dataset == "digits":
      data = load_digits()
    else:
        exit()

    print("\nThe Entire Dataset \n", data)
```

```
[ ] # Spliting our dataset into 2 parts for training and testing
    training_features, testing_features, training_labels, testing_labels  = train_test_split(data.data, data.target, test_size=0.3, random_state=0)
    print("\nTraining features: \n", training_features)
    print("\nTraining Labels: \n", training_labels)
    print("\nTesting Labels: \n", testing_labels)
```

```
[ ] # Naive Bayes classifier
    gnb = GaussianNB()
    gnb.fit(training_features, training_labels)
    gnb_pred = gnb.predict(testing_features)
    gnb_acc = 100*accuracy_score(testing_labels, gnb_pred)

    print("Naive Bayes Accuracy: %5.2f%%" %gnb_acc)
```

```
[ ] # KNN classifier
    best_k = 0
    best_acc = 0
    errors_list = []
    for k in range(1, 21):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(training_features, training_labels)
        knn_pred = knn.predict(testing_features)
        knn_acc = 100*accuracy_score(testing_labels, knn_pred)
        if knn_acc > best_acc:
            best_k = k
            best_acc = knn_acc
        errors_list.append(100-knn_acc)
        print("KNN Accuracy for k =%2d: %5.2f%%" % (k, knn_acc))
    print("Best K for KNN: %2d" % best_k)
    print("Errors = ", errors_list)
```

Output

```
[3] # Get user input for dataset selection
    dataset = 'none'
    while dataset not in {'iris', 'cancer','digits'}:
     dataset = input("Enter the name of the dataset (iris, cancer, digits): ")
     if dataset not in {'iris', 'cancer','digits'}:
       print("Invalid response: '%s'. Please try again." % dataset)
```

```
Enter the name of the dataset (iris, cancer, digits): digits
```

```
[7] # Naive Bayes classifier
    gnb = GaussianNB()
    gnb.fit(training_features, training_labels)
    gnb_pred = gnb.predict(testing_features)
    gnb_acc = 100*accuracy_score(testing_labels, gnb_pred)

    print("Naive Bayes Accuracy: %5.2f%%" %gnb_acc)
```

```
Naive Bayes Accuracy: 82.41%
```

```
[8] # KNN classifier
    best_k = 0
    best_acc = 0
    errors_list = []
    for k in range(1, 21):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(training_features, training_labels)
        knn_pred = knn.predict(testing_features)
        knn_acc = 100*accuracy_score(testing_labels, knn_pred)
        if knn_acc > best_acc:
            best_k = k
            best_acc = knn_acc
        errors_list.append(100-knn_acc)
        print("KNN Accuracy for k =%2d: %5.2f%%" % (k, knn_acc))
    print("Best K for KNN: %2d" % best_k)
    print("Errors = ", errors_list)
```

```
KNN Accuracy for k = 1: 98.89%
KNN Accuracy for k = 2: 98.15%
KNN Accuracy for k = 3: 98.70%
KNN Accuracy for k = 4: 97.96%
KNN Accuracy for k = 5: 98.15%
KNN Accuracy for k = 6: 97.59%
KNN Accuracy for k = 7: 97.96%
KNN Accuracy for k = 8: 97.59%
KNN Accuracy for k = 9: 97.59%
KNN Accuracy for k =10: 97.41%
KNN Accuracy for k =11: 97.04%
KNN Accuracy for k =12: 97.41%
KNN Accuracy for k =13: 97.04%
KNN Accuracy for k =14: 97.04%
KNN Accuracy for k =15: 97.22%
KNN Accuracy for k =16: 96.85%
KNN Accuracy for k =17: 96.30%
KNN Accuracy for k =18: 96.30%
KNN Accuracy for k =19: 96.30%
KNN Accuracy for k =20: 96.11%
Best K for KNN:  1
```

4) Source Code

Link:

https://colab.research.google.com/drive/1ZIw3LhCrwHlnGi1j_MMCXSxJ0kzPN02H?usp=sharing

```
[1]  import numpy as np
     from sklearn.datasets import load_breast_cancer
     from sklearn.model_selection import train_test_split
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import accuracy_score
     from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
[2]  # Get the cancer dataset and split it into data features and target values
     cancer = load_breast_cancer()
     X = cancer.data
     Y = cancer.target
```

```
[3]  # Spliting our dataset into 2 parts for training and testing
     training_features, testing_features, training_labels, testing_labels = train_test_split(X, Y, test_size=0.3, random_state=0)
```

```
[4]  normalize = "none"
     while normalize not in {"y", "n"}:
       normalize = input("Do you want to normalize the data? (y/n): ")
       if normalize not in {"y", "n"}:
         print("Invalid response: '%s'. Please try again (y/n):" % normalize)

     if normalize == "y":
       scaler = StandardScaler()
       model = scaler.fit(training_features) #use training_features to come up with the scaling model (mean, sd, min, max)
       train_std = model.transform(training_features) #use that model to transform training_features
       test_std = model.transform(testing_features) #use that model to transform testing_features
       print("Mean and Std Dev. of Transformed Data: ", train_std.mean(), test_std.std())
     else:
       print("I don't want to normalize the data.")

     Do you want to normalize the data? (y/n): y
     Mean and Std Dev. of Transformed Data:  7.617208557562213e-17 1.0080470501714467
```

```
[7]  if normalize == "y":
         knn = KNeighborsClassifier(n_neighbors=5)
         knn.fit(training_features, training_labels)
         knn_pred = knn.predict(testing_features)
         knn_acc = 100*accuracy_score(testing_labels, knn_pred)
         print("Accuracy using normalized data: %5.2f%%" % knn_acc)
     else:
         knn = KNeighborsClassifier(n_neighbors=5)
         knn.fit(training_features, training_labels)
         knn_pred = knn.predict(testing_features)
         knn_acc = 100*accuracy_score(testing_labels, knn_pred)
         print("Accuracy using unnormalized data: %5.2f%%" % knn_acc)
```

Output

[4]
```
normalize = "none"
while normalize not in {"y", "n"}:
  normalize = input("Do you want to normalize the data? (y/n): ")
  if normalize not in {"y", "n"}:
     print("Invalid response: '%s'. Please try again (y/n):" % normalize)

if normalize == "y":
  scaler = StandardScaler()
  model = scaler.fit(training_features) #use training_features to come up with the scaling model (mean, sd, min, max)
  train_std = model.transform(training_features) #use that model to transform training_features
  test_std = model.transform(testing_features) #use that model to transform testing_features
  print("Mean and Std Dev. of Transformed Data: ", train_std.mean(), test_std.std())
else:
  print("I don't want to normalize the data.")
```

```
Do you want to normalize the data? (y/n): y
Mean and Std Dev. of Transformed Data:  7.617208557562213e-17 1.0080470501714467
```

[7]
```
if normalize == "y":
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(training_features, training_labels)
    knn_pred = knn.predict(testing_features)
    knn_acc = 100*accuracy_score(testing_labels, knn_pred)
    print("Accuracy using normalized data: %5.2f%%" % knn_acc)
else:
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(training_features, training_labels)
    knn_pred = knn.predict(testing_features)
    knn_acc = 100*accuracy_score(testing_labels, knn_pred)
    print("Accuracy using unnormalized data: %5.2f%%" % knn_acc)
```

```
Accuracy using normalized data: 94.74%
```