Kunnithi Treecharoensomboon ID 63070503461

CPE383 Machine Learning: Quiz2

1. 1.5 hr. Min Distance classifier on 3 Gaussian Classes. Modify your KNN program from quiz 1 with 3 classes to create 50 random points for each class: red, blue, and yellow from a 2D Gaussian distribution (see Gaussian Data.ipynb) with means: (20, 30), (40, 40), (50, 40) and (s_x, s_y) of (3, 10), (10, 10), (15, 15), for red, blue, and yellow, respectively. Use 70% of the dataset as training data and 30% as testing data.

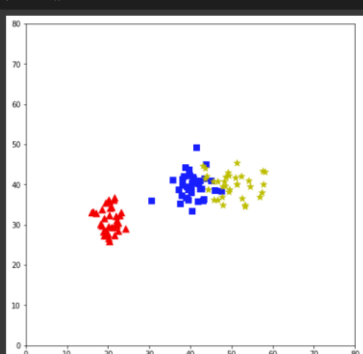
   A. 5 pts. Plot the 3 classes using the training data.

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import math
```

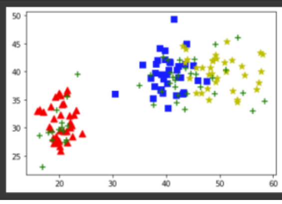
Create Training

```
number_points = 50

meanR = (20, 30) #mean_x, mean_y
covR = [[3, 0], [0, 10]] #s_x = 3, s_y = 10
dataR = np.random.multivariate_normal(meanR, covR, number_points)

meanB = (40, 40)
covB = [[10, 0], [0, 10]]
dataB = np.random.multivariate_normal(meanB, covB, number_points)

meanY = (50, 40)
covY = [[15, 0], [0, 15]]
dataY = np.random.multivariate_normal(meanY, covY, number_points)

testsize = 0.3
train_R, test_R, train_B, test_B, train_Y, test_Y = train_test_split(dataR, dataB, dataY, test_size=testsize, random_state=0)

train_data = np.concatenate((train_R, train_B, train_Y), axis=0)
train_data = train_data.astype(np.float32)
test_data = np.concatenate((test_R, test_B, test_Y), axis=0)
test_data = test_data.astype(np.float32)
```

```
#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(0, 80) #x min and max
plt.ylim(0, 80) #y min and max
plt.scatter(train_R[:, 0],train_R[:, 1], 80,'r','^')
plt.scatter(train_B[:,0],train_B[:,1],80,'b','s')
plt.scatter(train_Y[:,0],train_Y[:,1],80,'y','*')
plt.show()
```



```python
plt.scatter(train_R[:, 0],train_R[:, 1], 80,'r','^')
plt.scatter(train_B[:,0],train_B[:,1],80,'b','s')
plt.scatter(train_Y[:,0],train_Y[:,1],80,'y','*')
plt.scatter(test_data[:,0],test_data[:,1],80,'g','+') #green, circle
plt.show()
```



B. 5 pts. Using KNN with K = 5, report the total accuracy of the testing data.

```python
responses = np.concatenate((np.full(int(number_points*(1-testsize)), 0.0), np.full(int(number_points*(1-testsize)), 1.0), np.full(int(number_points*(1-testsize)), 2.0)))
responses = np.array(responses.reshape(-1, 1), dtype=np.float32)
expect_result = np.concatenate((np.full(int(number_points*testsize), 0.0), np.full(int(number_points*testsize), 1.0), np.full(int(number_points*testsize), 2.0)))
expect_result = np.array(expect_result.reshape(-1, 1), dtype=np.float32)
colorName = np.array(['Red', 'Blue', 'Yellow'])

# print("train_data shape:", train_data.shape)
# print("responses shape:", responses.shape)
knn = cv.ml.KNearest_create()
knn.train(train_data, cv.ml.ROW_SAMPLE, responses)
ret, results, neighbors, dist = knn.findNearest(test_data, 5)

correct = 0
for i in range(len(test_data)):
  resultColor = colorName[results[i].astype(int)]
  neighborColors = colorName[neighbors[i].astype(int)]

  if (results[i].astype(int) == expect_result[i].astype(int)):
    correct += 1

accuracy_percent = correct / len(expect_result)
print("KNN K = 5 accuray percent is",accuracy_percent)
```

```
KNN K = 5 accuray percent is 0.9555555555555556
```

C. Using the minimum distance classifier: 5 pts. Report the training data cluster mean for each class of red, blue, and yellow.

```python
[48] mean_R = np.mean(train_R, axis=0)
     mean_B = np.mean(train_B, axis=0)
     mean_Y = np.mean(train_Y, axis=0)

     centroid = [mean_R.tolist(), mean_B.tolist(), mean_Y.tolist()]
     centroid = np.array(centroid, dtype=np.float32)
     responses = [0, 1, 2]
     responses = np.array(responses, dtype=np.float32)

     print("mean of red =", centroid[0], "mean of blue =", centroid[1], "mean of yellow =", centroid[2])
```

```
mean of red = [19.931755 29.114399] mean of blue = [39.626453 40.266296] mean of yellow = [50.753605 41.74431 ]
```

D. Report the total accuracy of the testing data using this classifier.

```python
knn = cv.ml.KNearest_create()
knn.train(centroid, cv.ml.ROW_SAMPLE, responses)
ret, results, neighbors, dist = knn.findNearest(test_data, 1)

correct = 0
for i in range(len(test_data)):
  resultColor = colorName[results[i].astype(int)]
  neighborColors = colorName[neighbors[i].astype(int)]

  if (results[i].astype(int) == expect_result[i].astype(int)):
    correct += 1

accuracy_percent = correct / len(expect_result)
print("Minimum distance classifier accuray percent is",accuracy_percent)
```

```
Minimum distance classifier accuray percent is 0.9777777777777777
```

2. 4 hrs. Code Naive Bayes from Scratch. Write a program to read the Iris dataset, split into 2 parts: training and testing just like it was done in the example. Then write your own code to:

    a. 5 pts. Find the mean and standard deviation for each of the 4 features of each of the 3 classes from the training data. μ_ik and σ_ik for i = 1..4, k = 1..3. You will get pdfs P(x_i | c_k) for each class using the Gaussian distribution equation with μ_ik and σ_ik for i = 1..4, k = 1..3. This gets you pdfs: P(x_1, x_2, x_3, x_4 | c_1), P(x_1, x_2, x_3, x_4 | c_2), P(x_1, x_2, x_3, x_4 | c_3).

```
[1] import numpy as np
    import math
    from sklearn.datasets import load_iris
    from sklearn.model_selection import train_test_split

    the_data = load_iris()
```

```
[2] label_names = the_data['target_names']
    feature_names = the_data['feature_names']
    all_labels = the_data['target']
    all_features = the_data['data']
```

```
#see https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
# Spliting our dataset into 2 parts for training and testing
training_features, testing_features, training_labels, testing_labels = train_test_split(all_features,all_labels,test_size=0.2,random_state=0)
```

    b. 5 pts. Find the P(c_k) by counting the percent frequency of each class in your training data. Now we have P(c_k | x_1, x_2, x_3, x_4) ∝ P(x_1, x_2, x_3, x_4 | c_k) * P(c_k).

```
num_Label = len(label_names)
num_feature = len(feature_names)

mean = np.zeros((num_Label, num_feature))
std = np.zeros((num_Label, num_feature))

for k in range(num_Label):
  for i in range(num_feature):
    mean[k, i] = np.mean(training_features[np.where(training_labels == k), i])
    std[k, i] = np.std(training_features[np.where(training_labels == k), i])

# print(mean, "\n\n", std)

pdfs = np.zeros((num_Label, len(testing_features)))

for i in range(testing_features.shape[0]):
  for k in range(num_Label):
    pdf = 1.0
    for j in range(num_feature):
      # Calculate the probability of each test data matching each label
      pdf *= (1/(math.sqrt(2*math.pi)*std[k, j])) * math.exp(-(testing_features[i, j] - mean[k, j])**2/(2*(std[k, j]**2)))
    pdfs[k, i] = pdf

# print(pdfs)
```

    c. 5 pts. Then for each (x_1, x_2, x_3, x_4) in your test data: find the class k of 1, 2, or 3 for which P(c_k | x_1, x_2, x_3, x_4) is maximum, put that k into array my_predicted_labels

```
[17] # Classify the testing data
     my_predicted_labels = np.zeros(len(testing_labels))

     for i in range(len(testing_labels)):
         max_index = np.argmax(pdfs[:, i])
         my_predicted_labels[i] = max_index

     print(my_predicted_labels)

     [2. 1. 0. 2. 0. 2. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 2. 1. 0. 0.
      2. 0. 0. 1. 1. 0.]
```

d. 5 pts. Calculate and print the accuracy score from your implementation of Naive Bayes from scratch

```
[18] true_positive = 0
     for i in range(len(testing_labels)):
       if testing_labels[i] == my_predicted_labels[i]:
         true_positive += 1

     my_predict_accuracy = true_positive / len(testing_labels)
     print("My predict accuracy", my_predict_accuracy)

     My predict accuracy 0.9666666666666667
```

e. 5 pts. Use sklearn's GaussianNB classifier to report the accuracy score. Compare your result to sklearn's.

```
true_positive = 0
for i in range(len(testing_labels)):
  if testing_labels[i] == my_predicted_labels[i]:
    true_positive += 1

my_predict_accuracy = true_positive / len(testing_labels)
print("My predict accuracy", my_predict_accuracy)

My predict accuracy 0.9666666666666667
```

```
[19] # Training our Classifier
     from sklearn.naive_bayes import GaussianNB
     classifier = GaussianNB() #classifier is now an object of the Gaussian Naive Bayes class
     model = classifier.fit(training_features,training_labels)
     predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features

     true_positive = 0
     for i in range(len(testing_labels)):
       if testing_labels[i] == predicted_labels[i]:
         true_positive += 1

     GaussianNB_predict_accuracy = true_positive / len(testing_labels)
     print("GaussianNB predict accuracy", GaussianNB_predict_accuracy)

     GaussianNB predict accuracy 0.9666666666666667
```

3. 1 hr. Try the digits datasets. Change the "**Naive Bayes and KNN Iris and Cancer.ipynb**" program to allow the user to also select the digits dataset by entering "digits", in addition to "iris" and "cancer". Present the output results for both Naive Bayes and KNN classifiers using Sklearn. What is the best value of K in KNN?

```
[ ] #see https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

    dataset_type = 'none'
    while dataset_type not in {'iris', 'cancer', 'digits'}:
      dataset_type = input('Enter iris for the Iris dataset and cancer for the Breast Cancer dataset ')
      if dataset_type not in {'iris', 'cancer', 'digits'}:
        print("Invalid response: '%s'. Please try again." % dataset_type)

    Enter iris for the Iris dataset and cancer for the Breast Cancer dataset digits
```

```
[ ] from sklearn.datasets import load_iris, load_breast_cancer, load_digits
    if dataset_type == 'iris':
      the_data = load_iris() #get the data from sklearn
      print("Iris data set selected")
    elif dataset_type == 'cancer':
      the_data = load_breast_cancer() #get the data from sklearn
      print("Cancer data set selected")
    else:
      the_data = load_digits() #get the data from sklearn
      print("Digits data set selected")

    Digits data set selected
```

```
[34] # Training our Classifier
     from sklearn.naive_bayes import GaussianNB
     classifier = GaussianNB() #classifier is now an object of the Gaussian Naive Bayes class
     model = classifier.fit(training_features,training_labels)
     predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features
```

```
[35] print ("\nPredicted class labels: \n", predicted_labels)
     print("\nCorrect Testing class labels: \n", testing_labels)
     print("\nPredicted Class Names: \n", label_names[predicted_labels])
     print("\nCorrect Class Names: \n", label_names[testing_labels])
```

```
print ("Test data where predicted label equals the test label: \n", testing_labels == predicted_labels)
number_correct = (testing_labels == predicted_labels).sum()
print ("\nNumber of correct predictions: %d. Out of total test cases %d." %(number_correct, testing_labels.shape[0]) )
```

```
Test data where predicted label equals the test label:
 [ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True False  True  True
  True  True  True  True  True  True False  True False False  True  True
  True  True  True False  True False  True  True False  True  True  True
  True  True  True  True  True False  True  True  True  True  True  True
  True  True  True False  True  True False  True False  True  True  True
 False  True  True False  True  True  True False  True False  True  True
 False  True False  True False  True  True  True False  True False False
 False  True  True  True  True  True  True  True False  True  True False
  True  True  True  True False  True  True  True  True  True  True  True
  True  True False  True  True  True  True  True  True  True  True  True
 False  True  True  True  True False  True  True  True  True  True  True
  True False  True  True  True False  True  True  True  True  True  True
  True  True  True  True False  True  True  True False  True  True  True
  True  True  True False  True  True  True False False  True  True False
  True  True  True  True  True  True  True  True  True  True False  True
  True  True  True  True False  True  True False  True  True  True  True
  True  True False  True False  True False  True  True  True  True False
 False  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True False
  True  True  True  True  True  True  True  True  True False False  True
 False  True  True  True  True  True False  True  True  True  True False
  True  True  True False  True False  True  True  True False  True  True
 False  True  True False  True  True  True False  True  True  True  True
 False  True False  True  True  True  True False False  True  True  True
  True  True False  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True False False  True  True]

Number of correct predictions: 297. Out of total test cases 360.
```

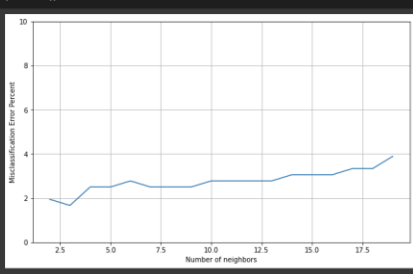
```
[37] from sklearn.metrics import accuracy_score
     # Calculating the % Accuracy of the prediction. For Iris dataset random_state = 0 gives 97%, random_state 40 gives 100%, random_state 5 gives 90%.
     accuracy_percent = 100*accuracy_score(testing_labels,predicted_labels)
     print("Prediction Accuracy : %5.2f%%" % accuracy_percent) #%% escapes the formatting % to print '%'
```

```
Prediction Accuracy : 82.50%
```

```
[38] from sklearn.neighbors import KNeighborsClassifier
     neighbor_size = []
     errors_list = []
     for k in range (2, 20):
       classifier = KNeighborsClassifier(n_neighbors = k)
       model = classifier.fit(training_features,training_labels)  # or can also use:   predicted_labels = classifier.fit(train_features,train_labels)
       predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features
       accuracy_percent = 100*accuracy_score(testing_labels,predicted_labels)
       # Calculating the % Accuracy of the prediction.
       print("Prediction Accuracy for k = %2d : %5.2f%%" % (k, accuracy_percent)) #%% escapes the formatting % to print '%'
       neighbor_size.append(k)
       errors_list.append(100-accuracy_percent)

     print (" K = ", neighbor_size, "\n", "Errors = ", errors_list)
```

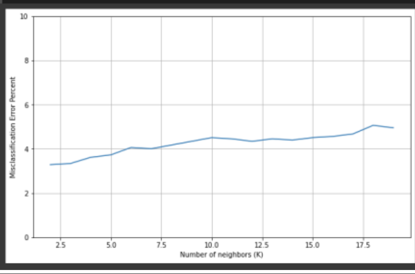
```
Prediction Accuracy for k =  2 : 98.06%
Prediction Accuracy for k =  3 : 98.33%
Prediction Accuracy for k =  4 : 97.50%
Prediction Accuracy for k =  5 : 97.50%
Prediction Accuracy for k =  6 : 97.22%
Prediction Accuracy for k =  7 : 97.50%
Prediction Accuracy for k =  8 : 97.50%
Prediction Accuracy for k =  9 : 97.50%
Prediction Accuracy for k = 10 : 97.22%
Prediction Accuracy for k = 11 : 97.22%
Prediction Accuracy for k = 12 : 97.22%
Prediction Accuracy for k = 13 : 97.22%
Prediction Accuracy for k = 14 : 96.94%
Prediction Accuracy for k = 15 : 96.94%
Prediction Accuracy for k = 16 : 96.94%
Prediction Accuracy for k = 17 : 96.67%
Prediction Accuracy for k = 18 : 96.67%
Prediction Accuracy for k = 19 : 96.11%
 K =  [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
 Errors =  [1.9444444444444429, 1.6666666666666714, 2.5, 2.5, 2.7777777777777857, 2.5, 2.5, 2.5, 2.7777777777777857, 2.7777777777777857, 2.7777777777777857, 2.7777777777777857, 3.055555555555557, 3.055555555555557, 3.055555555555557,
```

```
# plot misclassification error versus k
import matplotlib.pyplot as plt
plt.figure(figsize = (10, 6 ))
plt.plot(neighbor_size, errors_list) #x list and y list
plt.xlabel('Number of neighbors')
plt.ylabel('Misclassification Error Percent')
plt.ylim((0,10))
plt.grid(True)
plt.show()
```



```
[40] from sklearn.model_selection import cross_val_score
     # perform k-fold cross validation
     neighbor_size = [] #value of k used
     errors_list = [] #accuracy for k used
     num_folds = 5 #number of folds in k-fold. Usually 5 is a good number.
     for k in range (2, 20):
         classifier = KNeighborsClassifier(n_neighbors = k)
         #try the experiment, splitting data into num_fold folds, with 1 fold as test set
         scores = cross_val_score(classifier, all_features, all_labels, cv = num_folds, scoring = 'accuracy')
         print("scores = ", scores, "average = ", scores.mean())
         accuracy_percent = 100*scores.mean() #use the mean of the cv = num_folds scores as accuracy.
         neighbor_size.append(k)
         errors_list.append(100-accuracy_percent)

     print (" K = ", neighbor_size, "\n", "Error Percent = ", errors_list)

     scores =  [0.96111111 0.96666667 0.96935933 0.97771588 0.96100279] average =  0.9671711544413494
     scores =  [0.95555556 0.95833333 0.96657382 0.98607242 0.96657382] average =  0.966621788919839
     scores =  [0.94722222 0.95833333 0.96657382 0.98050139 0.96657382] average =  0.9638409161250386
     scores =  [0.94722222 0.95555556 0.96657382 0.98050139 0.9637883 ] average =  0.9627282575054161
     scores =  [0.94444444 0.95833333 0.96657382 0.97493036 0.95264624] average =  0.9593856391210152
     scores =  [0.93611111 0.96111111 0.96935933 0.98050139 0.95264624] average =  0.9599458372021046
     scores =  [0.93611111 0.95833333 0.96935933 0.97771588 0.94986072] average =  0.9582760755184154
     scores =  [0.93055556 0.95277778 0.97214485 0.97771588 0.94986072] average =  0.9566109563602601
     scores =  [0.93055556 0.94722222 0.96935933 0.97771588 0.94986072] average =  0.954942742185082
     scores =  [0.93611111 0.94444444 0.96935933 0.97771588 0.94986072] average =  0.9554982977406375
     scores =  [0.93611111 0.95277778 0.96935933 0.97493036 0.94986072] average =  0.9566078613432374
     scores =  [0.93333333 0.95       0.97214485 0.97493036 0.94707521] average =  0.9554967502321263
     scores =  [0.93611111 0.95277778 0.96935933 0.97214485 0.94986072] average =  0.9560507582791706
     scores =  [0.93611111 0.95       0.96935933 0.97214485 0.94707521] average =  0.9549380996595481
     scores =  [0.93055556 0.95       0.96935933 0.96935933 0.95264624] average =  0.9543840916125038
     scores =  [0.93611111 0.95       0.9637883  0.96657382 0.94986072] average =  0.9532667904673475
     scores =  [0.92777778 0.94444444 0.9637883  0.9637883  0.94707521] average =  0.9493748065614362
     scores =  [0.93055556 0.95       0.9637883  0.9637883  0.94428969] average =  0.950484370164036
      K =  [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
      Error Percent =  [3.282884555865067, 3.337821108016101, 3.615908387496134, 3.727174249458386, 4.061436087898485, 4.005416279789543, 4.172392448158462, 4.338904363973995, 4.505725781491805, 4.4501702259362474, 4.339213865676257, 4.45
```

```
# plot misclassification error versus k
plt.figure(figsize = (10, 6))
plt.plot(neighbor_size, errors_list) #x list and y_list
plt.xlabel('Number of neighbors (K)')
plt.ylabel('Misclassification Error Percent')
plt.ylim((0,10))
plt.grid(True)
plt.show()
```




```
[60] best_k = 0
     best_accuracy = 0
     for k in range(1, len(training_labels)):
       classifier = KNeighborsClassifier(n_neighbors = k)
       model = classifier.fit(training_features,training_labels)  # or can also use:   predicted_labels = classifier.fit(train_features,train_labels)
       predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features
       accuracy_percent = 100*accuracy_score(testing_labels,predicted_labels)
       if best_accuracy < accuracy_percent:
         best_k = k
         best_accuracy = accuracy_percent
         if best_accuracy == 100:
           break

     print("Best accuracy percent for KNN with k = %2d is %5.2f" %(best_k, best_accuracy))

     Best accuracy percent for KNN with k =  1 is 98.89
```

Ans: The best K is 1 with 98.89% accuracy

4. 10 pts. 1.5 hr. Normalize data option. Add an option to “Naive Bayes and KNN Iris and Cancer.ipynb” to ask the user whether to normalize the dataset by converting each feature into a Z-distribution by making mean = 0, and standard deviation = 1. For this problem, compare the accuracy results for the breast cancer dataset on the sklearn's KNN classifier using normalized vs. unnormalized data.

```
#see https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import math

dataset_type = 'none'
while dataset_type not in {'iris', 'cancer', 'digits'}:
  dataset_type = input('Enter iris for the Iris dataset and cancer for the Breast Cancer dataset ')
  if dataset_type not in {'iris', 'cancer', 'digits'}:
    print("Invalid response: '%s', Response must be only iris or cancer or digits. Please try again." % dataset_type)
```

```
Enter iris for the Iris dataset and cancer for the Breast Cancer dataset cancer
```

```
[3] if dataset_type == 'iris':
      the_data = datasets.load_iris() #get the data from sklearn
      print("Iris data set selected")
    elif dataset_type == 'cancer':
      the_data = datasets.load_breast_cancer() #get the data from sklearn
      print("Cancer data set selected")
    else:
      the_data = datasets.load_digits() #get the data from sklearn
      print("Digits data set selected")
```

```
Cancer data set selected
```

```
# the_data is an extension of the distionary data type into a sklearn type called bunch.  The Iris data is a dictionary data type.
# The dicitionay has the following key -> values:
# 'data': array n x 4 of 4 lengths
# 'target': array n x 1 of target classes with values 0, 1, or 2
# 'target_names': array for the 3 names of iris species values or array (['setosa', 'versicolor', 'virginica'], dtype='<U10')
# 'feature_names': list of 4 feature names.  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
# 'frame': ...
# 'DESCR': ...
# 'filename': 'iris.csv'
# 'data_module': ...
label_names = the_data['target_names']
feature_names = the_data['feature_names']
all_labels = the_data['target']
all_features = the_data['data']
# Looking at our data
print("\n the_data data type: ", type(the_data))
print("Feature Names: \n", feature_names)
print("\nTarget Label Names: \n", label_names)
print("\nfeatures data type: ", type(all_features))
print("\nAll Features in the data: \n", all_features)
print("\nAll Labels in the data: \n", all_labels)
```

```
 the_data data type:  <class 'sklearn.utils.Bunch'>
Feature Names:
 ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

```
#see https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
# Spliting our dataset into 2 parts for training and testing
training_features, testing_features, training_labels, testing_labels = train_test_split(all_features,all_labels,test_size=0.2,random_state=0)
# print("\nTraining features: \n", training_features)
# print("\nTraining Labels: \n", training_labels)
# print("\nTesting Labels: \n", testing_labels)
normalize = "none"
while normalize not in {'Y', 'N'}:
  normalize = input("Do you want to normalize the data (Y/N) ")
  if normalize == 'Y':
    scaler = StandardScaler()
    model = scaler.fit(training_features)
    training_features = model.transform(training_features)
    testing_features = model.transform(testing_features)
  elif normalize != 'N':
    print("Invalid response: '%s', Response must be only Y or N. Please try again.  " % dataset_type)
```

```
Do you want to normalize the data (Y/N) N
```

```
[7] # Training our Classifier
    from sklearn.naive_bayes import GaussianNB
    classifier = GaussianNB() #classifier is now an object of the Gaussian Naive Bayes class
    model = classifier.fit(training_features,training_labels)
    predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features
```

```
print ("\nPredicted class labels: \n", predicted_labels)
print("\nCorrect Testing class labels: \n", testing_labels)
print("\nPredicted Class Names: \n", label_names[predicted_labels])
print("\nCorrect Class Names: \n", label_names[testing_labels])
```

```
Predicted class labels:
 [0 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1
 0 1 0 1 1 0 1 0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1
 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1
 0 0 1]

Correct Testing class labels:
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1
 0 1 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0
 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1
 0 0 1]

Predicted Class Names:
 ['malignant' 'benign' 'benign' 'benign' 'benign' 'benign' 'benign'
 'benign' 'benign' 'benign' 'malignant' 'benign' 'benign' 'malignant'
 'malignant' 'malignant' 'benign' 'malignant' 'malignant' 'malignant'
 'malignant' 'malignant' 'benign' 'benign' 'malignant' 'benign' 'benign'
 'malignant' 'benign' 'malignant' 'benign' 'malignant' 'benign'
 'malignant' 'benign' 'malignant' 'benign' 'malignant' 'benign'
 'malignant' 'benign' 'benign' 'malignant' 'benign' 'malignant'
 'malignant' 'benign' 'benign' 'benign' 'malignant' 'malignant'
 'malignant' 'malignant' 'benign' 'benign' 'benign' 'benign' 'benign'
 'benign' 'malignant' 'malignant' 'malignant' 'benign' 'benign'
 'malignant' 'benign' 'malignant' 'malignant' 'malignant' 'benign'
 'benign' 'malignant' 'benign' 'benign' 'malignant' 'benign' 'benign'
 'benign' 'benign' 'benign' 'malignant' 'malignant' 'malignant' 'benign'
 'malignant' 'benign' 'benign' 'benign' 'malignant' 'malignant' 'benign'
 'benign' 'benign' 'malignant' 'benign' 'benign' 'malignant' 'benign'
 'benign' 'benign' 'benign' 'benign' 'benign' 'benign' 'malignant'
```

```
[9] print ("Test data where predicted label equals the test label: \n", testing_labels == predicted_labels)
    number_correct = (testing_labels == predicted_labels).sum()
    print ("\nNumber of correct predictions: %d. Out of total test cases %d." %(number_correct, testing_labels.shape[0]) )
```

```
Test data where predicted label equals the test label:
 [ True  True  True  True  True  True  True  True  True  True False  True
  True False False  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True False  True  True  True False  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True False  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
 False  True  True  True  True  True]

Number of correct predictions: 106. Out of total test cases 114.
```

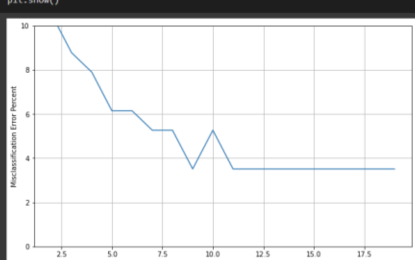
```
from sklearn.metrics import accuracy_score
# Calculating the % Accuracy of the prediction. For Iris dataset random_state = 0 gives 97%, random_state 40 gives 100%, random_state 5 gives 90%.
accuracy_percent = 100*accuracy_score(testing_labels,predicted_labels)
print("Prediction Accuracy : %5.2f%%" % accuracy_percent) #%% escapes the formatting % to print '%'
```

```
Prediction Accuracy : 92.98%
```

```
from sklearn.neighbors import KNeighborsClassifier
neighbor_size = []
errors_list = []
for k in range (2, 20):
  classifier = KNeighborsClassifier(n_neighbors = k)
  model = classifier.fit(training_features,training_labels)  # or can also use:   predicted_labels = classifier.fit(train_features,train_labels)
  predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features
  accuracy_percent = 100*accuracy_score(testing_labels,predicted_labels)
  # Calculating the % Accuracy of the prediction.
  print("Prediction Accuracy for k = %2d : %5.2f%%" % (k, accuracy_percent)) #%% escapes the formatting % to print '%'
  neighbor_size.append(k)
  errors_list.append(100-accuracy_percent)

print (" K = ", neighbor_size, "\n", "Errors = ", errors_list)
```

```
Prediction Accuracy for k =  2 : 89.47%
Prediction Accuracy for k =  3 : 91.23%
Prediction Accuracy for k =  4 : 92.11%
Prediction Accuracy for k =  5 : 93.86%
Prediction Accuracy for k =  6 : 93.86%
Prediction Accuracy for k =  7 : 94.74%
Prediction Accuracy for k =  8 : 94.74%
Prediction Accuracy for k =  9 : 96.49%
Prediction Accuracy for k = 10 : 94.74%
Prediction Accuracy for k = 11 : 96.49%
Prediction Accuracy for k = 12 : 96.49%
Prediction Accuracy for k = 13 : 96.49%
Prediction Accuracy for k = 14 : 96.49%
Prediction Accuracy for k = 15 : 96.49%
Prediction Accuracy for k = 16 : 96.49%
Prediction Accuracy for k = 17 : 96.49%
Prediction Accuracy for k = 18 : 96.49%
Prediction Accuracy for k = 19 : 96.49%
 K =  [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
 Errors =  [10.526315789473685, 8.771929824561411, 7.89473684210526, 6.1403508771929864, 6.1403508771929864, 5.26315789473685, 5.26315789473685, 3.5087719298245617, 5.26315789473685, 3.5087719298245617, 3.5087719298245617, 3.50877192
```

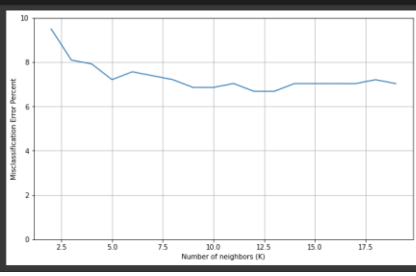
```
[12] # plot misclassification error versus k
     import matplotlib.pyplot as plt
     plt.figure(figsize = (10, 6 ))
     plt.plot(neighbor_size, errors_list) #x list and y list
     plt.xlabel('Number of neighbors')
     plt.ylabel('Misclassification Error Percent')
     plt.ylim((0,10))
     plt.grid(True)
     plt.show()
```



```
[14] from sklearn.model_selection import cross_val_score
     # perform k-fold cross validation
     neighbor_size = [] #value of k used
     errors_list = [] #accuracy for k used
     num_folds = 5 #number of folds in k-fold. Usually 5 is a good number.
     for k in range (2, 20):
         classifier = KNeighborsClassifier(n_neighbors = k)
         #try the experiment, splitting data into num_fold folds, with 1 fold as test set
         scores = cross_val_score(classifier, all_features, all_labels, cv = num_folds, scoring = 'accuracy')
         print("scores = ", scores, "average = ", scores.mean())
         accuracy_percent = 100*scores.mean() #use the mean of the cv = num_folds scores as accuracy.
         neighbor_size.append(k)
         errors_list.append(100-accuracy_percent)

     print (" K = ", neighbor_size, "\n", "Error Percent = ", errors_list)
```

```
scores =  [0.89473684 0.9122807  0.92982456 0.89473684 0.89380531] average =  0.9050768514205869
scores =  [0.87719298 0.92105263 0.94736842 0.93859649 0.91150442] average =  0.9191429902189101
scores =  [0.89473684 0.92105263 0.94736842 0.93859649 0.90265487] average =  0.9208818506443098
scores =  [0.88596491 0.93859649 0.93859649 0.94736842 0.92920354] average =  0.9279459711224964
scores =  [0.89473684 0.93859649 0.93859649 0.92982456 0.92035398] average =  0.9244216736531594
scores =  [0.87719298 0.93859649 0.94736842 0.94736842 0.92035398] average =  0.9261760596180716
scores =  [0.87719298 0.93859649 0.94736842 0.94736842 0.92920354] average =  0.9279459711224964
scores =  [0.87719298 0.93859649 0.94736842 0.95614035 0.9380531 ] average =  0.9314702685918336
scores =  [0.87719298 0.93859649 0.94736842 0.95614035 0.9380531 ] average =  0.9314702685918336
scores =  [0.87719298 0.93859649 0.94736842 0.95614035 0.92920354] average =  0.9297003570874087
scores =  [0.88596491 0.93859649 0.94736842 0.95614035 0.9380531 ] average =  0.9332246545567457
scores =  [0.87719298 0.93859649 0.94736842 0.95614035 0.94690265] average =  0.9332401800962584
scores =  [0.87719298 0.93859649 0.93859649 0.95614035 0.9380531 ] average =  0.9297158826269213
scores =  [0.86842105 0.93859649 0.93859649 0.95614035 0.94690265] average =  0.9297314081664337
scores =  [0.86842105 0.93859649 0.94736842 0.95614035 0.9380531 ] average =  0.9297158826269213
scores =  [0.86842105 0.93859649 0.92982456 0.95614035 0.95575221] average =  0.9297469337059463
scores =  [0.87719298 0.93859649 0.92105263 0.95614035 0.94690265] average =  0.9279770222015216
scores =  [0.86842105 0.93859649 0.93859649 0.95614035 0.94690265] average =  0.9297314081664337
 K =  [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
 Error Percent =  [9.49231485794131, 8.085700978108989, 7.911814935569012, 7.205402887750353, 7.557832634684061, 7.382394038192842, 7.205402887750353, 6.852973140816644, 6.852973140816644, 7.029964291259134, 6.677534544325425, 6.6759
```

```
# plot misclassification error versus k
plt.figure(figsize = (10, 6))
plt.plot(neighbor_size, errors_list) #x list and y_list
plt.xlabel('Number of neighbors (K)')
plt.ylabel('Misclassification Error Percent')
plt.ylim((0,10))
plt.grid(True)
plt.show()
```




```
best_k = 0
best_accuracy = 0
for k in range(1, math.ceil(math.sqrt(len(training_labels)))):
  classifier = KNeighborsClassifier(n_neighbors = k)
  model = classifier.fit(training_features,training_labels)  # or can also use:   predicted_labels = classifier.fit(train_features,train_labels)
  predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features
  accuracy_percent = 100*accuracy_score(testing_labels,predicted_labels)
  if best_accuracy < accuracy_percent:
    best_k = k
    best_accuracy = accuracy_percent
    if best_accuracy == 100:
      break

print("Best accuracy percent for KNN is k =%2d is %5.2f (normalize = %s)" %(best_k, best_accuracy, normalize))
```

```
Best accuracy percent for KNN is k = 9 is 96.49 (normalize = N)
```

```
best_k = 0
best_accuracy = 0
for k in range(1, math.ceil(math.sqrt(len(training_labels)))):
  classifier = KNeighborsClassifier(n_neighbors = k)
  model = classifier.fit(training_features,training_labels)  # or can also use:   predicted_labels = classifier.fit(train_features,train_labels)
  predicted_labels = model.predict(testing_features) #use the model obtained in previous step to predict labels for testing features
  accuracy_percent = 100*accuracy_score(testing_labels,predicted_labels)
  if best_accuracy < accuracy_percent:
    best_k = k
    best_accuracy = accuracy_percent
    if best_accuracy == 100:
      break

print("Best accuracy percent for KNN is k =%2d is %5.2f (normalize = %s)" %(best_k, best_accuracy, normalize))
```

```
Best accuracy percent for KNN is k =16 is 97.37 (normalize = Y)
```

Ans: The best K of normalized data is 16, different from unnormalized that is 9