Peeraya Khantaruangsakul 63070501054

CPE 393 Machine Learning : Take Home Quiz2

Gaussian Classes

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.

- **[5 pts.]** Plot the 3 classes using the training data.
- **[5 pts.]** Using KNN with K = 5, report the total accuracy of the testing data.

  - Using the minimum distance classifier:

- **[5 pts.]** Report the training data cluster mean for each class of red, blue, and yellow.
- **[5 pts.]** Report the total accuracy of the testing data using this classifier.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import cv2 as cv
from sklearn.model_selection import train_test_split

#Create a 2D cluster of 50 points with mean:
number_points = 50
mean0 = (20, 30) #mean_x, mean_y
cov0 = [[3, 0], [0, 10]] #s_x = 3, s_y = 20
Data0 = np.random.multivariate_normal(mean0, cov0, number_points)   # red = 0
Data0 = np.concatenate((Data0, np.zeros((number_points, 1))), axis=1)
#Create a 2D cluster of 50 points with mean:
mean1 = (40, 40) #mean_x, mean_y
cov1 = [[10, 0], [0, 10]] #s_x = 10, s_y = 10
Data1 = np.random.multivariate_normal(mean1, cov1, number_points)  # blue = 1
Data1 = np.concatenate((Data1, np.ones((number_points, 1))), axis=1)
#Create a 2D cluster of 50 points with mean:
mean2 = (50, 40) #mean_x, mean_y
cov2 = [[15, 0], [0, 15]] #s_x = 15, s_y = 15
Data2= np.random.multivariate_normal(mean2, cov2, number_points)  # yellow = 2
Data2 = np.concatenate((Data2, 2*np.ones((number_points, 1))), axis=1)
#Combine all data into one array and split it into features and labels
Data = np.concatenate((Data0, Data1, Data2), axis=0)
features = Data[:, 0:2]
labels = Data[:, 2]
#Define a custom colormap
```

```
cmap2 = colors.ListedColormap(['r', 'b', 'y'])
#Plot the scatter plot with the custom colormap
plt.scatter(Data[:,0], Data[:,1], c=Data[:,2], cmap=cmap2)
plt.show()
training_features, testing_features, training_labels, testing_labels = train_test_split(features, labels, test_size=0.3, random_state=42)
training_features = np.array(training_features, dtype=np.float32)
training_labels = np.array(training_labels, dtype=np.float32)
testing_features = np.array(testing_features, dtype=np.float32)
#print("\nTraining features: \n", training_features)
#print("\nTraining Labels: \n", training_labels)
#print("\nTesting Labels: \n", testing_labels)
#plot train data
plt.scatter(training_features[:,0], training_features[:,1], c=training_labels, cmap=cmap2)

knn = cv.ml.KNearest_create()
knn.train(training_features, cv.ml.ROW_SAMPLE, training_labels)
k=5
ret, predicted_labels, neighbours, dist = knn.findNearest(testing_features, k)
#print("Predicted Labels: \n", predicted_labels.ravel())
#print("Testing Labels: \n", testing_labels)
number_correct = np.sum(testing_labels == predicted_labels.ravel())
#print ("\nNumber of correct predictions: %d Out of total test cases %d." (number_correct, testing_labels.shape[0]))
# tatal accuracy
from sklearn.metrics import accuracy_score
accuracy_percent = accuracy_score(testing_labels, predicted_labels) * 100
print ("\nKNN Accuracy: %5.2f%%" %accuracy_percent)

# group by target find each mean
df = pd.DataFrame(training_features)
df['target'] = training_labels
df.columns = ['x', 'y', 'target']
df = df.groupby('target').mean()
print(df)
# df.values[0]

min_indices = []
for test_feature in testing_features:
    min_distance = np.inf
    min_index = -1
    for i in range(0, len(df.values)):
        distance = np.linalg.norm(test_feature - df.values[i]) # (x1-x2)^2 + (y1-y2)^2
        if distance < min_distance:
            min_distance = distance
            min_index = i
    min_indices.append(min_index)
# plot test data
plt.scatter(testing_features[:,0], testing_features[:,1], c=min_indices, cmap=cmap2)
```

```
plt.scatter(df.values[:,0], df.values[:,1], c='k', marker='x', s=100)
plt.show()
#print("Predicted Labels: \n", min_indices)
#print("Testing Labels: \n", testing_labels)
number_correct = np.sum(testing_labels == min_indices)
#print ("\nNumber of correct predictions: %d Out of total test cases %d." %(number_correct, testing_labels.shape[0]) )
# tatal accuracy
from sklearn.metrics import accuracy_score
accuracy_percent_mean = accuracy_score(testing_labels, min_indices) * 100
print ("\ncluster mean Accuracy: %5.2f%%" %accuracy_percent_mean)
```

Result : Scatter plot data

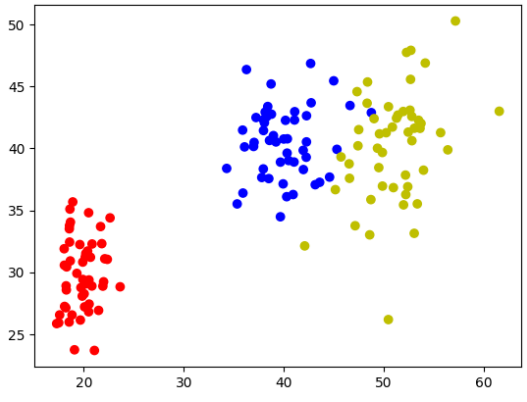
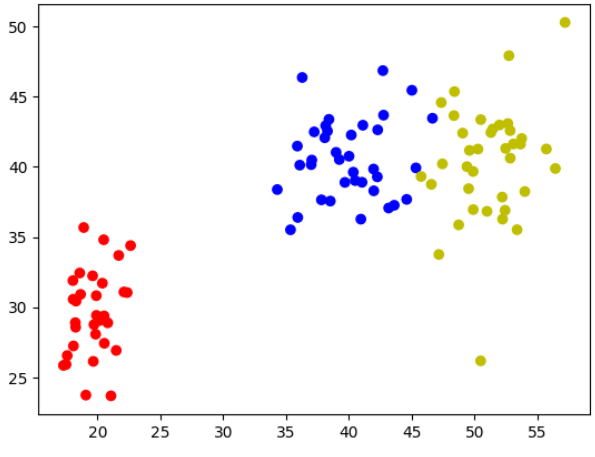
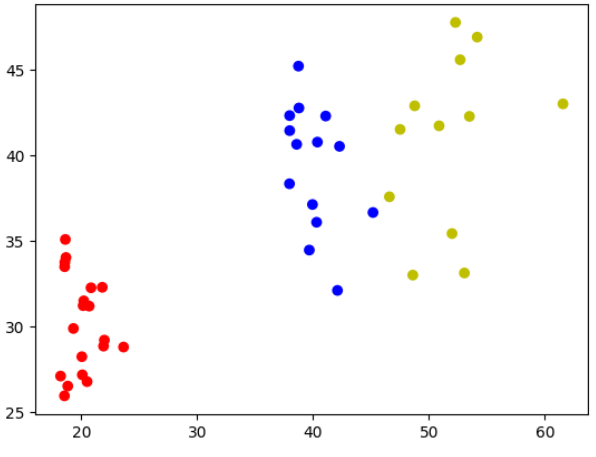


Image1 : Scatter plot all data

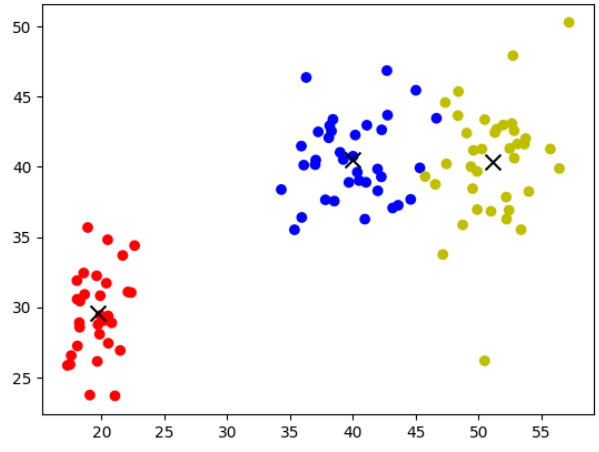
Training data KNN	Testing data KNN

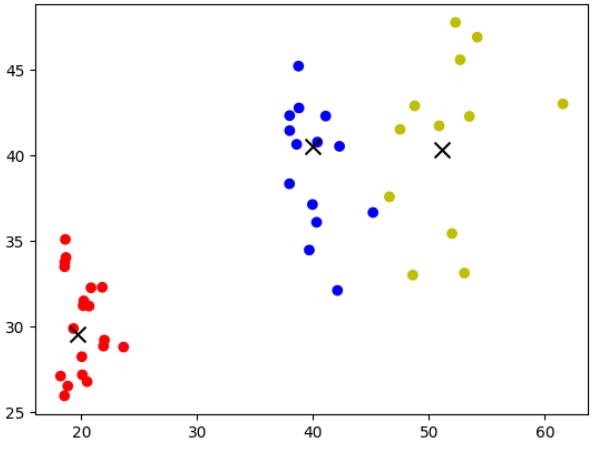




Accuracy : 93.33%

Training data cluster mean	Testing data cluster mean





Accuracy : 93.33%

Notice : The accuracy of the code changes every time it is executed because the input data is randomly generated from a normal (Gaussian) distribution. The KNN algorithm may produce a higher accuracy than the mean of the clusters at times, but this can vary.

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:

- **[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).
- **[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).
- **[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
- **[5 pts.]** Calculate and print the accuracy score from your implementation of Naive Bayes from scratch
- **[5 pts.]** Use sklearn's GaussianNB classifier to report the accuracy score. Compare your result to sklearn's.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

data_iris = load_iris()
df_iris = pd.DataFrame(data_iris.data, columns=data_iris.feature_names)
df_iris['target'] = data_iris.target
df_iris['target_names'] = df_iris['target'].apply(lambda x: data_iris.target_names[x])
#print('df_iris')
#count the number of each target
df_iris['target_names'].value_counts()
features = [col for col in df_iris.columns if col not in ["target", "target_names"]]
df_iris_features = df_iris[features]

training_features, testing_features, training_labels, testing_labels = train_test_split(df_iris_features, df_iris['target'], test_size=0.3, random_state=42)

# find mean and standard deviation for each feature
training_features.groupby(training_labels).agg(['mean', 'std'])
#fine mean and standard deviation for each feature
mean_setosa, mean_versicolor, mean_virginica = training_features.groupby(training_labels).mean().values
std_setosa, std_versicolor, std_virginica = training_features.groupby(training_labels).std().values
def gaussian(x, mean, std):
   return 1/(np.sqrt(2*np.pi)*std)*np.exp(-((x-mean)**2)/(2*std**2))
def predict(sepal_length, sepal_width, petal_length, petal_width):
   return (1/3)*sepal_length*sepal_width*petal_length*petal_width

#setosa probability
p_setosa_sepal_length, p_setosa_sepal_width = gaussian(testing_features['sepal length (cm)'], mean_setosa[0], std_setosa[0]), gaussian(testing_features['sepal width (cm)'], mean_setosa[1], std_setosa[1])

p_setosa_petal_length, p_setosa_petal_width = gaussian(testing_features['petal length (cm)'], mean_setosa[2], std_setosa[2]), gaussian(testing_features['petal width (cm)'], mean_setosa[3], std_setosa[3])

p_setosa = predict(p_setosa_sepal_length, p_setosa_sepal_width, p_setosa_petal_length, p_setosa_petal_width)
#versicolor probability

p_versicolor_sepal_length, p_versicolor_sepal_width = gaussian(testing_features['sepal length (cm)'], mean_versicolor[0], std_versicolor[0]), gaussian(testing_features['sepal width (cm)'], mean_versicolor[1], std_versicolor[1])

p_versicolor_petal_length, p_versicolor_petal_width = gaussian(testing_features['petal length (cm)'], mean_versicolor[2], std_versicolor[2]), gaussian(testing_features['petal width (cm)'], mean_versicolor[3], std_versicolor[3])

p_versicolor = predict(p_versicolor_sepal_length, p_versicolor_sepal_width, p_versicolor_petal_length, p_versicolor_petal_width)
#virginica probability

p_virginica_sepal_length, p_virginica_sepal_width = gaussian(testing_features['sepal length (cm)'], mean_virginica[0], std_virginica[0]), gaussian(testing_features['sepal width (cm)'], mean_virginica[1], std_virginica[1])

p_virginica_petal_length, p_virginica_petal_width = gaussian(testing_features['petal length (cm)'], mean_virginica[2], std_virginica[2]), gaussian(testing_features['petal width (cm)'], mean_virginica[3], std_virginica[3])

p_virginica = predict(p_virginica_sepal_length, p_virginica_sepal_width, p_virginica_petal_length, p_virginica_petal_width)

df_ans = pd.DataFrame()
df_ans['p_setosa'] = p_setosa
```

```
df_ans['p_versicolor'] = p_versicolor
df_ans['p_virginica'] = p_virginica
df_ans['target_names'] = testing_labels.apply(lambda x: data_iris.target_names[x])
# what column has the highest probability
df_ans['my_predicted_labels'] = df_ans[['p_setosa', 'p_versicolor', 'p_virginica']].idxmax(axis=1)
df_ans['my_predicted_labels'] = df_ans['my_predicted_labels'].apply(lambda x: x.split('_')[1])
df_ans['Correct_Naive'] = df_ans['target_names'] == df_ans['my_predicted_labels']
#df_ans

# total accuracy
from sklearn.metrics import accuracy_score
accuracy_percent = accuracy_score(df_ans['target_names'], df_ans['my_predicted_labels']) * 100
print ("Accuracy ['NB from Scratch']: %5.2f%%" %accuracy_percent)

# use Guussian Naive Bayes from sklearn
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
model = classifier.fit(training_features, training_labels)
predictions_labels = model.predict(testing_features)
# create a dataframe to compare the results
df_ans['GaussianNB'] = predictions_labels
df_ans['GaussianNB'] = df_ans['GaussianNB'].apply(lambda x: data_iris.target_names[x])
df_ans['correct_GaussianNB'] = df_ans['target_names'] == df_ans['GaussianNB']
accuracy_percent = accuracy_score(testing_labels, predictions_labels) * 100
print ("Accuracy ['sklearn.naive_bayes']: %5.2f%%" %accuracy_percent)
df_ans['check equal'] = df_ans['Correct_Naive'] == df_ans['correct_GaussianNB']
if all(df_ans['check equal']):
  print('success')
else:
  print('fail')
# df_ans
```

Result : Use NB from Scratch and sklearn.naive_bayes are equal !

Count number each target

```
setosa          50
versicolor      50
virginica       50
Name: target_names, dtype: int64
```

Mean and SD for each feature

	sepal length (cm)		sepal width (cm)		petal length (cm)		petal width (cm)	
	mean	std	mean	std	mean	std	mean	std
target								
0	4.964516	0.340145	3.377419	0.375686	1.464516	0.185380	0.248387	0.109151
1	5.862162	0.531952	2.724324	0.299449	4.210811	0.495975	1.302703	0.206137
2	6.559459	0.658896	2.986486	0.314609	5.545946	0.544464	2.005405	0.297159

Accuracy NB from scratch	Accuracy sklearn NB
Accuracy ['NB from Scratch']: 97.78%	Accuracy ['sklearn.naive_bayes']: 97.78%

Result : Table comparison

	p_setosa	p_versicolor	p_virginica	target_names	my_predicted_labels	Correct_Naive	GaussianNB	correct_GaussianNB	check equal
73	8.495974216373 9e-86	0.246764894330 9137	0.001258100669 5214741	versicolor	versicolor	True	versicolor	True	True
18	0.066835656307 0934	1.547476305633 4375e-14	3.914620890024 895e-21	setosa	setosa	True	setosa	True	True
118	1.379296721751 9334e-278	4.158172376743 635e-15	0.000734813569 3486311	virginica	virginica	True	virginica	True	True
78	2.381833659545 693e-89	0.225239757012 71964	0.006287797494 120535	versicolor	versicolor	True	versicolor	True	True
76	1.611390877537 4873e-101	0.046890332979 053116	0.009701492820 626217	versicolor	versicolor	True	versicolor	True	True
31	0.537412520041 8565	6.225695412373 457e-13	1.058044833405 103e-20	setosa	setosa	True	setosa	True	True
64	2.715182883900 107e-50	0.181071515268 0116	8.446308929486 313e-06	versicolor	versicolor	True	versicolor	True	True
141	1.365069978302 215e-167	5.831445285886 447e-08	0.090283967 7195985	virginica	virginica	True	virginica	True	True
68	5.524061110953 6815e-92	0.048823680624 95299	0.000354640554 97585685	versicolor	versicolor	True	versicolor	True	True
82	3.255379521522 217e-56	0.372504693383 58355	2.253072857126 852e-05	versicolor	versicolor	True	versicolor	True	True
110	4.040842227352 1186e-144	4.697870188439 192e-05	0.142378654379 70798	virginica	virginica	True	virginica	True	True
12	0.655188387649 4634	1.989372767561 8345e-16	2.153565086679 1657e-24	setosa	setosa	True	setosa	True	True
36	0.548226394710 3726	2.909809722415 4143e-16	1.096016654779 5418e-23	setosa	setosa	True	setosa	True	True
9	0.951579399305 036	6.077752910348 969e-16	1.192234962406 2674e-23	setosa	setosa	True	setosa	True	True
19	1.406819320931 4126	6.952941130477 053e-16	5.502883053363 978e-23	setosa	setosa	True	setosa	True	True
56	5.173259614644 493e-103	0.012651590913 17496	0.016722330860 93908	versicolor	virginica	False	virginica	False	True
104	4.786566494829 852e-193	7.492086769550 051e-08	0.181303252286 04033	virginica	virginica	True	virginica	True	True
69	7.463229502740 8155e-53	0.175735225279 20618	2.636066699785 1157e-06	versicolor	versicolor	True	versicolor	True	True
55	4.051360158092 354e-80	0.404388848802 357	0.000851525457 4574084	versicolor	versicolor	True	versicolor	True	True
132	4.456400270317 763e-182	4.579521216278 4873e-07	0.164701813345 8979	virginica	virginica	True	virginica	True	True
29	1.498067063795 065	7.932824480149 442e-15	1.414317986045 8003e-22	setosa	setosa	True	setosa	True	True
127	2.617801617023 01e-121	0.006370695433 587632	0.076840918102 33035	virginica	virginica	True	virginica	True	True
26	0.945907672729 9516	7.201919099781 163e-13	1.163988147968 9134e-20	setosa	setosa	True	setosa	True	True
128	3.809267870284 251e-175	3.363567930321 4086e-06	0.194005073169 28353	virginica	virginica	True	virginica	True	True
131	1.688030164333 8996e-226	1.024734230134 503e-13	0.000327981994 60198934	virginica	virginica	True	virginica	True	True
145	5.686433585062 988e-171	1.110624770223 1617e-07	0.122916626020 68654	virginica	virginica	True	virginica	True	True
108	1.064611595791 7999e-169	3.639068822290 671e-05	0.052582296589 87485	virginica	virginica	True	virginica	True	True
143	1.300773124528 0264e-207	7.769250174870 557e-10	0.092635790128 18615	virginica	virginica	True	virginica	True	True
45	1.476156718316 4224	3.570677015299 23e-14	1.285380721584 6217e-22	setosa	setosa	True	setosa	True	True
30	1.534882750967 7039	1.889257560002 275e-14	2.531052905261 8374e-22	setosa	setosa	True	setosa	True	True
22	0.060567305455 63339	2.094088860362 571e-19	3.173401756435 2787e-27	setosa	setosa	True	setosa	True	True
15	0.002903939533 25554827	1.753073687550 161e-18	2.085368450473 4286e-24	setosa	setosa	True	setosa	True	True
65	1.323776281269 983e-84	0.056791848292 66569	0.003159315454 754126	versicolor	versicolor	True	versicolor	True	True
11	2.012324460846 139	3.254520277088 44e-15	1.138760051006 7437e-22	setosa	setosa	True	setosa	True	True
42	0.450517041563 6648	6.807984394817 06e-17	5.588561326430 609e-25	setosa	setosa	True	setosa	True	True
146	1.914533951027 938e-133	0.001182486015 0072881	0.040033866451 46921	virginica	virginica	True	virginica	True	True
51	6.669949795600 425e-91	0.046994274448 43515	0.007223169351 06481	versicolor	versicolor	True	versicolor	True	True
27	2.168287540882 159	1.661662137814 3807e-15	7.791795879213 691e-23	setosa	setosa	True	setosa	True	True
4	2.324771487984 7083	1.256842422603 4073e-16	5.653565674525 631e-24	setosa	setosa	True	setosa	True	True
32	0.157503259231 97364	8.251823932837 359e-20	6.877267037410 008e-26	setosa	setosa	True	setosa	True	True
142	1.847633628773 076e-135	0.001545994529 86237 23	0.057468221511 5415	virginica	virginica	True	virginica	True	True
85	9.596119050166 38e-94	0.011723670664 946102	0.004610508955 755377	versicolor	versicolor	True	versicolor	True	True
86	1.106142068812 845e-100	0.026551578191 788893	0.016236009348 796095	versicolor	versicolor	True	versicolor	True	True
16	0.140563022031 35	3.631178628558 084e-16	2.259728547848 433e-23	setosa	setosa	True	setosa	True	True
10	0.885686271175 0445	3.506486854905 4204e-16	4.029290337228 28e-23	setosa	setosa	True	setosa	True	True

What is the best value of K in KNN?

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?

```
import numpy as np
import pandas as pd
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.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

dataset_type = input('Enter iris for the Iris dataset, cancer for the Breast Cancer dataset and digits for the Digits dataset:')
if dataset_type.lower() == 'iris':
  the_data = load_iris() #get the data from sklearn
elif dataset_type.lower() == 'cancer':
  the_data = load_breast_cancer() #get the data from sklearn
```

```
elif dataset_type.lower() == 'digits':
  the_data = load_digits() #get the data from sklearn
else:
  print('Invalid dataset type')
  exit()
print('The dataset is ', dataset_type)
def check_type(name):
    if dataset_type.lower() == 'iris':
        feature = the_data.data
        target = the_data.target
    elif dataset_type.lower() == 'cancer':
        feature = the_data.data
        target = the_data.target
    elif dataset_type.lower() == 'digits':
        feature = the_data.data
        target = the_data.target
    return feature, target

feature, target = check_type(dataset_type)
training_features, testing_features, training_labels, testing_labels  = train_test_split(feature, target, test_size=0.2)

def Gaussian_NB(feature, target):
    classifier = GaussianNB()
    model = classifier.fit(training_features, training_labels)
    # use the model obtained in previous step to predict labels for testing features
    predicted_labels = model.predict(testing_features)
    from sklearn.metrics import accuracy_score   # evaluate the model
    accuracy_percent = accuracy_score(testing_labels, predicted_labels) * 100
    print("\nThe " + dataset_type +" use GaussianNB Accuracy: %5.2f%%\n" % accuracy_percent)

def best_K(feature, target):
    neighbor_size = []
    errors_list = []
    best_k = 0
    for k in range (2, 20):
        classifier = KNeighborsClassifier(n_neighbors = k)
        # or can also use:   predicted_labels = classifier.fit(train_features,train_labels)
        model = classifier.fit(training_features,training_labels)
         #use the model obtained in previous step to predict labels for testing features
        predicted_labels = model.predict(testing_features)
         # Calculating the % Accuracy of the prediction.
        accuracy_percent = accuracy_score(testing_labels, predicted_labels) * 100
        #%% escapes the formatting % to print '%
        print("Accuracy for k = %2d : %5.2f%%" % (k, accuracy_percent))
        neighbor_size.append(k)
        errors_list.append(101-accuracy_percent)
        if min(errors_list):
```

```
        best_k = neighbor_size[errors_list.index(min(errors_list))]
    #print (" K = ", neighbor_size, "\n", "Errors = ", errors_list)
    return best  k
def Knn(k, feature, target):
    classifier = KNeighborsClassifier(n_neighbors=k)
    model = classifier.fit(training_features, training_labels)
    # use the model obtained in previous step to predict labels for testing features
    predicted_labels = model.predict(testing_features)
    accuracy_percent = accuracy_score(testing_labels, predicted_labels) * 100   # evaluate the model
    print("\nThe " + dataset_type +" use KNN Accuracy: %5.2f%%" % accuracy_percent)

feature, target = check_type(dataset_type)
Gaussian_NB(feature, target)
k = best_K(feature, target)
Knn(k, feature, target)
print("\nThe best K is ", k)
```

Result : The best value of K in KNN and accuracy for Different datasets.

Iris Dataset	Breast Cancer Dataset	Digits Dataset
The iris use GaussianNB Accuracy: 93.33%	The cancer use GaussianNB Accuracy: 96.49%	The digits use GaussianNB Accuracy: 85.56%
Accuracy for k = 2 : 96.67%	Accuracy for k = 2 : 92.98%	Accuracy for k = 2 : 98.33%
Accuracy for k = 3 : 96.67%	Accuracy for k = 3 : 92.11%	Accuracy for k = 3 : 98.33%
Accuracy for k = 4 : 96.67%	Accuracy for k = 4 : 92.98%	Accuracy for k = 4 : 98.33%
Accuracy for k = 5 : 96.67%	Accuracy for k = 5 : 92.98%	Accuracy for k = 5 : 98.61%
Accuracy for k = 6 : 96.67%	Accuracy for k = 6 : 92.98%	Accuracy for k = 6 : 97.78%
Accuracy for k = 7 : 96.67%	Accuracy for k = 7 : 92.98%	Accuracy for k = 7 : 97.78%
Accuracy for k = 8 : 96.67%	Accuracy for k = 8 : 92.98%	Accuracy for k = 8 : 97.50%
Accuracy for k = 9 : 96.67%	Accuracy for k = 9 : 93.86%	Accuracy for k = 9 : 97.78%
Accuracy for k = 10 : 93.33%	Accuracy for k = 10 : 92.98%	Accuracy for k = 10 : 97.78%
Accuracy for k = 11 : 96.67%	Accuracy for k = 11 : 93.86%	Accuracy for k = 11 : 97.50%
Accuracy for k = 12 : 96.67%	Accuracy for k = 12 : 93.86%	Accuracy for k = 12 : 96.94%
Accuracy for k = 13 : 96.67%	Accuracy for k = 13 : 92.98%	Accuracy for k = 13 : 97.22%
Accuracy for k = 14 : 96.67%	Accuracy for k = 14 : 92.98%	Accuracy for k = 14 : 97.50%
Accuracy for k = 15 : 96.67%	Accuracy for k = 15 : 92.11%	Accuracy for k = 15 : 97.22%
Accuracy for k = 16 : 96.67%	Accuracy for k = 16 : 91.23%	Accuracy for k = 16 : 97.22%
Accuracy for k = 17 : 96.67%	Accuracy for k = 17 : 91.23%	Accuracy for k = 17 : 96.94%
Accuracy for k = 18 : 93.33%	Accuracy for k = 18 : 92.98%	Accuracy for k = 18 : 96.94%
Accuracy for k = 19 : 96.67%	Accuracy for k = 19 : 92.11%	Accuracy for k = 19 : 96.67%
The iris use KNN Accuracy: 96.67%	The cancer use KNN Accuracy: 93.86%	The digits use KNN Accuracy: 98.61%
The best K is 2	**The best K is 9**	**The best K is 5**

Notice : Accuracy may vary with each execution due to the split of training and testing data without specifying the random state. and the best K is defined as the value that results in the highest accuracy while being as low as possible.

Normalize data

[10 pts.] 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.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris, load_breast_cancer
from sklearn.model_selection import train_test_split
```

```
dataset_type = input('Enter iris for the Iris dataset and cancer for the Breast Cancer dataset ')
if dataset_type.lower() == 'iris':
  the_data = load_iris() #get the data from sklearn
elif dataset_type.lower() == 'cancer':
  the_data = load_breast_cancer() #get the data from sklearn
else:
  print('Invalid dataset type')
  exit()
#print('The dataset is ', dataset_type)
```

```
def check_type(name):
    if dataset_type.lower() == 'iris':
        feature = the_data.data
        target = the_data.target
    elif dataset_type.lower() == 'cancer':
        feature = the_data.data
        target = the_data.target
    return feature, target
```

```
def z_score(feature):
    mean = int(0)
    std = int(1)
    z = (feature - mean) / std
    return z
```

```
feature, target = check_type(dataset_type)
training_features, testing_features, training_labels, testing_labels = train_test_split(feature, target, test_size=0.2)
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
def best_K(feature, target):
    neighbor_size = []
    errors_list = []
    best_k = 0
```

```
    for k in range (2, 20):
        classifier = KNeighborsClassifier(n_neighbors = k)
        # or can also use:   predicted_labels = classifier.fit(train_features,train_labels)
        model = classifier.fit(training_features,training_labels)
        #use the model obtained in previous step to predict labels for testing features
        predicted_labels = model.predict(testing_features)
        accuracy_percent = accuracy_score(testing_labels, predicted_labels) * 100
        # 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(101-accuracy_percent)
        if min(errors_list):
            best_k = neighbor_size[errors_list.index(min(errors_list))]
    #print (" K = ", neighbor_size, "\n", "Errors = ", errors_list)
    return best_k
```

```
def Knn(k, feature, target):
    classifier = KNeighborsClassifier(n_neighbors=k)
    model = classifier.fit(training_features, training_labels)
    # use the model obtained in previous step to predict labels for testing features
    predicted_labels = model.predict(testing_features)
    # evaluate the model
    accuracy_percent = accuracy_score(testing_labels, predicted_labels) * 100
    print("\nThe " + dataset_type +" use KNN Accuracy: %5.2f%%" %accuracy_percent)
    return accuracy_percent
```

```
feature, target = check_type(dataset_type)
print("Unnormalized " + dataset_type + " dataset:")
k_unnormalized = best_K(feature, target)
accuracy_percent_unnormalized  = Knn(k_unnormalized, feature, target)
print("\nThe best 'unnormalized' K is ", k_unnormalized)
print("-------------------------------------------------\n")
print("Normalized " + dataset_type + " dataset:")
feature_z = z_score(feature)
k_normalized = best_K(feature_z, target)
accuracy_percent_normalized = Knn(k_normalized, feature_z, target)
print("\nThe best 'normalized' K is ", k_normalized)
print("\n compare accuracy of unnormalized : normalized == %5.2f%% : %5.2f%%" %
(accuracy_percent_unnormalized, accuracy_percent_normalized))
```

Result : Compare the accuracy results normalized vs. unnormalized data.

	Iris Dataset	Breast Cancer Dataset
Normalized	Normalized iris dataset: The iris use KNN Accuracy: 100.00% The best 'normalized' K is 7	Normalized cancer dataset: The cancer use KNN Accuracy: 97.37% The best 'normalized' K is 5
Unnormalized	Unnormalized iris dataset: The iris use KNN Accuracy: 100.00% The best 'unnormalized' K is 7	Unnormalized cancer dataset: The cancer use KNN Accuracy: 97.37% The best 'unnormalized' K is 5
Compare	compare accuracy of unnormalized : normalized == 100.00% : 100.00%	compare accuracy of unnormalized : normalized == 97.37% : 97.37%

Notice : The results demonstrate that normalizing and unnormalizing result in equal values of accuracy and K. Normalization helps reduce the range of data and is typically scaled to the range of [0, 1].