Short Assignment 2

This is an individual assignment.

Due: Tuesday, October 4 @ 11:59pm

Crab Dataset Description

The Crab Data Set has 200 samples and 7 features (Frontal Lip, Rear Width, Length, Width, Depth, Male and Female), describing 5 morphological measurements on 50 crabs each of two color forms and both sexes, of the species *Leptograpsus* variegatus collected at Fremantle, W. Australia.

- Dataset Source: Campbell, N.A. and Mahon, R.J. (1974) A multivariate study of variation in two species of rock crab of genus *Leptograpsus*. *Australian Journal of Zoology* 22, 417–425.

The data set is saved in the file "crab.txt": the first column corresponds to the class label (crab species) and the other 7 columns correspond to the features.

Use the first 140 samples as your training set and the last 60 samples as your test set.

In [1]:
```
import pandas as pd
data = pd.read_csv("crab.txt", delimiter="\t")

data
```

Out[1]:

	Species	FrontalLip	RearWidth	Length	Width	Depth	Male	Female
0	0	20.6	14.4	42.8	46.5	19.6	1	0
1	1	13.3	11.1	27.8	32.3	11.3	1	0
2	0	16.7	14.3	32.3	37.0	14.7	0	1
3	1	9.8	8.9	20.4	23.9	8.8	0	1
4	0	15.6	14.1	31.0	34.5	13.8	0	1
...	...	...	...	...	...	...	...	...
195	1	12.3	11.0	26.8	31.5	11.4	1	0
196	1	12.0	11.1	25.4	29.2	11.0	0	1
197	1	8.8	7.7	18.1	20.8	7.4	1	0
198	1	16.2	15.2	34.5	40.1	13.9	0	1
199	0	15.6	14.0	31.6	35.3	13.8	0	1

200 rows × 8 columns

Split the training data and the test data

In [2]:
```
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
```

In [3]:
```
# Partitioning the data into training and test sets

x_train = data.iloc[:140,1:].to_numpy()
t_train = data.iloc[:140,0].to_numpy()

x_test = data.iloc[140:,1:].to_numpy()
t_test = data.iloc[140:,0].to_numpy()
```

In [4]:
```
x_train.shape, x_test.shape, t_train.shape, t_test.shape
```

Out[4]:
```
((140, 7), (60, 7), (140,), (60,))
```

In [5]:
```
# seperate data for different spieces
x_train[t_train == 1].shape
```

Out[5]:
```
(68, 7)
```

In [6]:
```
# features for different spieces
c0_xtrain = x_train[t_train == 0] # spiece 0
c1_xtrain = x_train[t_train == 1] # spiece 1
```

In [7]:
```
c1_xtrain.shape
```

Out[7]:
```
(68, 7)
```

Problem Set

Answer the following questions:

1. Implement the Naive Bayes classifier, under the assumption that your data likelihood model $p(x|C_j)$ is a multivariate Gaussian and the prior probabilities $p(C_j)$ are dictated by the number of samples $n_j \in \mathbb{R}$ that you have for each class. Build your own code to implement the classifier.

2. Did you encounter any problems when implementing the probabilistic generative model? What is your solution for the problem? Explain why your solution works. (Note: There is more than one solution.)

3. Report your classification results in terms of a confusion matrix in both training and test set. (You can use the function `confusion_matrix` from the module `sklearn.metrics`.)

1. Implement the Naive Bayes Classifier

In [8]:
```
# prior probabilities for Class 0 and Class 1

p0 = np.sum(t_train)/ x_train.shape[0]

p1 = 1 - p0
```

Scaling for Class 0

In [9]:
```
# This pipeline will apply Standardization to all numerical attributes
# The attributes that are one-hot/interger-encoded (such as gender) will remain as is

c0_scaling_pipeline= ColumnTransformer([('num_attribs', StandardScaler(), list(range(5
                                    remainder='passthrough')
c0_scaling_pipeline.fit(c0_xtrain)
```

Out[9]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('num_attribs', StandardScaler(),
                                 [0, 1, 2, 3, 4])])
```

In [10]:
```
# mean of each feature column of spiece 0
c0_scaling_pipeline.transformers_[0][1].mean_
```

Out[10]:
```
array([17.10555556, 13.62083333, 34.1375    , 38.10277778, 15.46527778])
```

In [11]:
```
c0_mean = c0_xtrain.mean(axis=0)
```

In [12]:
```
# variance for each feature column of spiece 0
c0_scaling_pipeline.transformers_[0][1].var_
```

Out[12]:
```
array([ 9.1691358 ,  6.76387153, 38.45012153, 48.30138117,  8.32226659])
```

In [13]:
```
# covaraince for Class 0

c0_cov = np.cov(c0_xtrain.T)
```

In [14]:
```
c0_cov.shape
```

Out[14]:
```
(7, 7)
```

Scaling for Class 1

In [15]:
```
# This pipeline will apply Standardization to all numerical attributes
# The attributes that are one-hot/interger-encoded (such as gender) will remain as is

c1_scaling_pipeline= ColumnTransformer([('num_attribs', StandardScaler(), list(range(5
                                    remainder='passthrough')
c1_scaling_pipeline.fit(c1_xtrain)
```

Out[15]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('num_attribs', StandardScaler(),
                                 [0, 1, 2, 3, 4])])
```

In [16]:
```
# mean of each feature column of spiece 0
c1_scaling_pipeline.transformers_[0][1].mean_
```

Out[16]:
```
array([14.03529412, 11.84705882, 30.08676471, 34.73382353, 12.56764706])
```

In [17]:
```
c1_mean = c1_xtrain.mean(axis=0)
```

In [18]:
```
# variance for each feature column of spiece 0
c1_scaling_pipeline.transformers_[0][1].var_
```

Out[18]:
```
array([ 9.08346021,  4.50396194, 48.49261894, 63.22547362,  9.74571799])
```

In [19]:
```
# covaraince for Class 0

c1_cov = np.cov(c1_xtrain.T)
```

In [20]:
```
c1_cov.shape
```

Out[20]:
```
(7, 7)
```

In [21]:
```
from scipy.stats import multivariate_normal

# the probabaility density function (pdf) for each class
train_y0 = multivariate_normal.pdf(x_train, mean=c0_mean, cov=c0_cov, allow_singular=T
train_y1 = multivariate_normal.pdf(x_train, mean=c1_mean, cov=c1_cov, allow_singular=T
```

Predictions on training data

In [22]:
```
# posterior distributions: they represent our classification decision
train_pos0 = train_y0*p0 / (train_y0*p0 + train_y1*p1) # P(C0|x)
train_pos1 = train_y1*p1 / (train_y0*p0 + train_y1*p1) # P(C1|x)
```

In [23]:
```
# prediction of spiece from the train data
train_pre = []

for i in range(len(x_train)):
    if train_pos1[i] > train_pos0[i]:
        train_pre.append(1)
    else:
        train_pre.append(0)

train_pre = np.array(train_pre)
```

Predictions on test data

In [24]:
```
# data likelihoods for the test set

test_y0 = multivariate_normal.pdf(x_test, mean=c0_mean, cov=c0_cov, allow_singular=Tru
test_y1 = multivariate_normal.pdf(x_test, mean=c1_mean, cov=c1_cov, allow_singular=Tru
```

In [25]:
```
# Posterior Probabilities

test_y0_pos = test_y0*p0 / (test_y0*p0 + test_y1*p1) #P(C0|x)
test_y1_pos = test_y1*p1 / (test_y0*p0 + test_y1*p1) #P(C1|x)
```

In [26]:
```
# prediction of spiece from the test data
test_pre = []

for i in range(len(x_test)):
    if test_y1_pos[i] > test_y0_pos[i]:
```

```
        test_pre.append(1)
    else:
        test_pre.append(0)

test_pre = np.array(test_pre)
```

2. Problems Encountered

While implementing the probabilistic generative model, I encountered singular matrix error. This error means that the covariance matrix is non-invertable because its determinant is zero.

The multivariate_normal.pdf() function always performs a test to verify the covariance matrix and this test can be skipped by passing allow_singular=True.

However, a better solution would be to rescale the data. Professor mentioned about rescaling on the day this assignment was due, and I did not had enough time to implement it here.

3. Classification results in terms of a confusion matrix in both training and test set

Confusion Matrix for Training Data

In [27]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(t_train, train_pre)
```

Out[27]:
```
array([[72,  0],
       [ 0, 68]], dtype=int64)
```

Alternate way of evaluating the prediction

In [28]:
```
t_train - train_pre
```

Out[28]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

Confusion Matrix for Test Data

In [29]:
```
confusion_matrix(t_test, test_pre)
```

Out[29]:
```
array([[28,  0],
       [ 0, 32]], dtype=int64)
```

Alternate way of evaluating the prediction

In [30]:
```
t_test - test_pre
```

Out[30]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

Submit Your Solution

Confirm that you've successfully completed the assignment.

Along with the Notebook, include a PDF of the notebook with your solutions.

`add` and `commit` the final version of your work, and `push` your code to your GitHub repository.

Submit the URL of your GitHub Repository as your assignment submission on Canvas.
