Short Assignment 2 Solutions

In [1]:
```
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.metrics import confusion_matrix
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
```

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 firt 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.

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`.)**

In [2]:
```
data = pd.read_csv("crab.txt", delimiter="\t")

data.head()
```

Out[2]:

	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

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()

X_train.shape, X_test.shape, t_train.shape, t_test.shape
```

Out[3]:
```
((140, 7), (60, 7), (140,), (60,))
```

In [4]:
```
list(range(5))
```

Out[4]:
```
[0, 1, 2, 3, 4]
```

In [5]:
```
# This pipeline will apply Standardization to all numerical attributes
# The attributes that are one-hot/interger-encoded (such as gender) will remain as is

scaling_pipeline = ColumnTransformer([('num_attribs', StandardScaler(), list(range(5))
                                      remainder='passthrough')

scaling_pipeline.fit(X_train)
```

Out[5]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('num_attribs', StandardScaler(),
                                 [0, 1, 2, 3, 4])])
```

In [6]:
```
X_train = scaling_pipeline.transform(X_train)
X_test = scaling_pipeline.transform(X_test)
```

Training the Naive Bayes Classifier - estimating its parameters

In [7]:
```
# Prior probabilities

pC0 = np.sum(t_train==0)/t_train.size
pC1 = np.sum(t_train==1)/t_train.size

pC0, pC1
```

Out[7]:
```
(0.5142857142857142, 0.4857142857142857)
```

In [8]:
```
# Means and covariances of the data likelihood

mu0 = np.mean(X_train[t_train==0,:],axis=0)
mu1 = np.mean(X_train[t_train==1,:],axis=0)

cov0 = np.cov(X_train[t_train==0,:].T)
cov1 = np.cov(X_train[t_train==1,:].T)
```

In [9]:
```
# Training Data Likelihood
y0_train = multivariate_normal.pdf(X_train, mean=mu0, cov=cov0) #P(x|C0)
y1_train = multivariate_normal.pdf(X_train, mean=mu1, cov=cov1) #P(x|C1)

# Test Data Likelihood
y0_test = multivariate_normal.pdf(X_test, mean=mu0, cov=cov0) #P(x|C0)
y1_test = multivariate_normal.pdf(X_test, mean=mu1, cov=cov1) #P(x|C1)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Input In [9], in <cell line: 2>()
      1 # Training Data Likelihood
----> 2 y0_train = multivariate_normal.pdf(X_train, mean=mu0, cov=cov0) #P(x|C0)
      3 y1_train = multivariate_normal.pdf(X_train, mean=mu1, cov=cov1) #P(x|C1)
      5 # Test Data Likelihood

File ~\anaconda3\envs\eee3773\lib\site-packages\scipy\stats\_multivariate.py:516, in
multivariate_normal_gen.pdf(self, x, mean, cov, allow_singular)
    514 dim, mean, cov = self._process_parameters(None, mean, cov)
    515 x = self._process_quantiles(x, dim)
--> 516 psd = _PSD(cov, allow_singular=allow_singular)
    517 out = np.exp(self._logpdf(x, mean, psd.U, psd.log_pdet, psd.rank))
    518 return _squeeze_output(out)

File ~\anaconda3\envs\eee3773\lib\site-packages\scipy\stats\_multivariate.py:165, in
_PSD.__init__(self, M, cond, rcond, lower, check_finite, allow_singular)
    163 d = s[s > eps]
    164 if len(d) < len(s) and not allow_singular:
--> 165     raise np.linalg.LinAlgError('singular matrix')
    166 s_pinv = _pinv_1d(s, eps)
    167 U = np.multiply(u, np.sqrt(s_pinv))

LinAlgError: singular matrix
```

Note that if we used all 7 features, the covariance matrix Σ_X would be singular. This is because one of the features is colinear with another feature. In particular, features male and female are the complement of one another.

There are 2 ways to address this issue:

1. Eliminate one of the features (method used here)
2. Diagonally-load the covariance matrix: $\Sigma_X + \lambda I$

Either method will be accepted equally. But note that if one decides to use method 2 (diagonally load the covariance matrix), then we have an extra hyperparameter we need to estimate the value for, λ. It is typically preferred to perform feature selection than distortion the covariance to force it to become full rank (and hence invertible).

Method 1 - Feature Selection

In this first approach, we simply eliminate the feature/s that are colinear with one another. In this dataset, it is easy for us to see that the features male and female are the complement of one another, knowing if a sample is male, then we know it is not female, and vice-versa. So we can eliminate one of then as keep both is redundant.

But, in practice, sometimes this identification is not straightforwardly visible from the data. But we can measure correlation coefficient between features:

In [10]:
```
data.corr()
```

Out[10]:

	Species	FrontalLip	RearWidth	Length	Width	Depth	Male	
Species	1.000000e+00	-0.437966	-0.315751	-0.288333	-0.216180	-0.423716	-2.664535e-17	2.2
FrontalLip	-4.379655e-01	1.000000	0.906988	0.978842	0.964956	0.987627	4.330897e-02	-4.3
RearWidth	-3.157507e-01	0.906988	1.000000	0.892743	0.900402	0.889205	-2.915970e-01	2.9
Length	-2.883330e-01	0.978842	0.892743	1.000000	0.995023	0.983204	1.049828e-01	-1.0
Width	-2.161801e-01	0.964956	0.900402	0.995023	1.000000	0.967812	7.443726e-02	-7.4
Depth	-4.237165e-01	0.987627	0.889205	0.983204	0.967812	1.000000	8.971958e-02	-8.9
Male	-2.664535e-17	0.043309	-0.291597	0.104983	0.074437	0.089720	1.000000e+00	-1.0
Female	2.220446e-17	-0.043309	0.291597	-0.104983	-0.074437	-0.089720	-1.000000e+00	1.0

Now, it is clear that the feature female and male are co-linear.

- We can also see that other features are highly correlated with others (e.g. length-and-width, length-and-depth).
- In practice we can decide to eliminate them or use, for example, Principal Component Analysis (PCA) to decorrelate the features. We will see how to do this later in the course.

In [11]:
```
# Eliminating the last feature ("female")

X_train = data.iloc[:140,1:7].to_numpy()
X_test = data.iloc[140:,1:7].to_numpy()
t_train = data.iloc[:140,0].to_numpy()
t_test = data.iloc[140:,0].to_numpy()

X_train.shape, X_test.shape, t_train.shape, t_test.shape
```

Out[11]:
```
((140, 6), (60, 6), (140,), (60,))
```

In [12]:
```
scaling_pipeline.fit(X_train)

X_train = scaling_pipeline.transform(X_train)
X_test = scaling_pipeline.transform(X_test)
```

In [13]:
```
# Recomputing MLE estimates for mean and covariance

mu0 = np.mean(X_train[t_train==0,:],axis=0)
mu1 = np.mean(X_train[t_train==1,:],axis=0)

cov0 = np.cov(X_train[t_train==0,:].T)
cov1 = np.cov(X_train[t_train==1,:].T)
```

In [14]:
```
# Training Data Likelihood
y0_train = multivariate_normal.pdf(X_train, mean=mu0, cov=cov0) #P(x|C0)
y1_train = multivariate_normal.pdf(X_train, mean=mu1, cov=cov1) #P(x|C1)

# Test Data Likelihood
y0_test = multivariate_normal.pdf(X_test, mean=mu0, cov=cov0) #P(x|C0)
y1_test = multivariate_normal.pdf(X_test, mean=mu1, cov=cov1) #P(x|C1)

y0_train.shape, y1_train.shape
```

Out[14]:
```
((140,), (140,))
```

In [15]:
```
# Posterior for Training Data
pos0_train = (y0_train*pC0)/(y0_train*pC0 + y1_train*pC1) # P(C0|x_train)
pos1_train = (y1_train*pC1)/(y0_train*pC0 + y1_train*pC1) # P(C1|x_train)
pos_train = np.array([pos0_train, pos1_train]).T # Creating a matrix with posterior pr
likelihood_train = np.array([y0_train, y1_train]).T # Creating a matrix with likelihoo

# Posterior for Test Data
pos0_test = (y0_test*pC0)/(y0_test*pC0 + y1_test*pC1) # P(C0|x_test)
pos1_test = (y1_test*pC1)/(y0_test*pC0 + y1_test*pC1) # P(C0|x_test)
pos_test = np.array([pos0_test, pos1_test]).T # Creating a matrix with posterior proba
likelihood_test = np.array([y0_test, y1_test]).T # Creating a matrix with likelihoods

# Prediction for Training Data
predict_train = np.argmax(pos_train, axis=1) # Label prediction for training data
# labels it as the class with largest posterior
predict_likelihood_train = likelihood_train[predict_train] # Likelihood value for the

# Prediction for Test Data
predict_test = np.argmax(pos_test, axis=1) # Label prediction for test set
predict_likelihood_test = likelihood_train[predict_test] # Likelihood value for the as
```

In [17]:
```
print('Confusion matrix in Training')
print(confusion_matrix(t_train, predict_train))

print('Confusion matrix in Test')
print(confusion_matrix(t_test, predict_test))
```

```
Confusion matrix in Training
[[72  0]
 [ 0 68]]
Confusion matrix in Test
[[28  0]
 [ 0 32]]
```

The confusion matrix show that all samples were correctly classified for both training and test sets.

Method 2 - Diagonally-load the Covariance matrix

Recall the multivariate Gaussian probability density function (PDF):

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)\right)$$

where $\mathbf{x} \in \mathbb{R}^d$ is a data sample, $\mu \in \mathbb{R}^d$ is the mean vector, Σ is the covariance matrix, $|\Sigma|$ is the determinant of the covariance matrix and Σ^{-1} is the inverse of the covariance matrix.

In order for us to implement this PDF, the covariance matrix must be invertible (i.e. its determinant must be different than 0). As we saw from before, at least one feature is linearly dependent on another (female and male features), hence the covariance matrix will have dependent column/s, which will make it not full rank, which in turn will produce a determinant of 0 and it is not invertible.

Let's obtain the original data (in the 7-dimensional space):

In [18]:
```
# 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()

X_train.shape, X_test.shape, t_train.shape, t_test.shape
```

Out[18]:
```
((140, 7), (60, 7), (140,), (60,))
```

In [19]:
```
scaling_pipeline.fit(X_train)

X_train = scaling_pipeline.transform(X_train)
X_test = scaling_pipeline.transform(X_test)
```

In [20]:
```
# Visualize the covariance matrix for the training data

pd.DataFrame(np.cov(X_train.T))
```

Out[20]:

	0	1	2	3	4	5	6
0	1.007194	0.922772	0.984102	0.968550	0.995087	0.015013	-0.015013
1	0.922772	1.007194	0.900605	0.905846	0.904054	-0.146339	0.146339
2	0.984102	0.900605	1.007194	1.001785	0.989244	0.050729	-0.050729
3	0.968550	0.905846	1.001785	1.007194	0.972557	0.034288	-0.034288
4	0.995087	0.904054	0.989244	0.972557	1.007194	0.039355	-0.039355
5	0.015013	-0.146339	0.050729	0.034288	0.039355	0.251593	-0.251593
6	-0.015013	0.146339	-0.050729	-0.034288	-0.039355	-0.251593	0.251593

As you can see the last two columns are linearly dependent.

In [21]:
```
# Prior probabilities

pC0 = np.sum(t_train==0)/t_train.size
pC1 = np.sum(t_train==1)/t_train.size

pC0, pC1
```

Out[21]:
```
(0.5142857142857142, 0.4857142857142857)
```

In [22]:
```
# Means and covariances of the data likelihood

mu0 = np.mean(X_train[t_train==0,:],axis=0)
mu1 = np.mean(X_train[t_train==1,:],axis=0)

cov0 = np.cov(X_train[t_train==0,:].T)
cov1 = np.cov(X_train[t_train==1,:].T)
```

Now, let's diagonally-load the covariance before computing the data likelihood for each class, that is:

$$\mathbf{\Sigma} \leftarrow \mathbf{\Sigma} + \lambda \mathbf{I}$$

where λ is a (constant) hyperparameter that must be learned using cross-validation for the training data. I am not going to include that step here but you should implement CV when utilizing this approaches in your research projects.

In [23]:
```
# Diagonally-loading covariance matrices -- injecting small value along the diagonal c
# this forces it to be full rank, and hence invertible.

reg = 0.0001 # this is a tunable parameter, that must be learned using cross-validatic

cov0_reg = cov0 + reg*np.eye(len(cov0))

cov1_reg = cov1 + reg*np.eye(len(cov1))
```

In [24]:
```
# Training Data Likelihood
y0_train = multivariate_normal.pdf(X_train, mean=mu0, cov=cov0_reg) #P(x|C0)
y1_train = multivariate_normal.pdf(X_train, mean=mu1, cov=cov1_reg) #P(x|C1)
```

```
# Test Data Likelihood
y0_test = multivariate_normal.pdf(X_test, mean=mu0, cov=cov0_reg) #P(x|C0)
y1_test = multivariate_normal.pdf(X_test, mean=mu1, cov=cov1_reg) #P(x|C1)
```

No *singular* error message this time!

In [25]:
```
# Posterior for Training Data
pos0_train = (y0_train*pC0)/(y0_train*pC0 + y1_train*pC1) # P(C0|x_train)
pos1_train = (y1_train*pC1)/(y0_train*pC0 + y1_train*pC1) # P(C1|x_train)
pos_train = np.array([pos0_train, pos1_train]).T # Creating a matrix with posterior pr
likelihood_train = np.array([y0_train, y1_train]).T # Creating a matrix with likelihoo

# Posterior for Test Data
pos0_test = (y0_test*pC0)/(y0_test*pC0 + y1_test*pC1) # P(C0|x_test)
pos1_test = (y1_test*pC1)/(y0_test*pC0 + y1_test*pC1) # P(C0|x_test)
pos_test = np.array([pos0_test, pos1_test]).T # Creating a matrix with posterior proba
likelihood_test = np.array([y0_test, y1_test]).T # Creating a matrix with likelihoods

# Prediction for Training Data
predict_train = np.argmax(pos_train, axis=1) # Label prediction for training data
# labels it as the class with largest posterior
predict_likelihood_train = likelihood_train[predict_train] # Likelihood value for the

# Prediction for Test Data
predict_test = np.argmax(pos_test, axis=1) # Label prediction for test set
predict_likelihood_test = likelihood_train[predict_test] # Likelihood value for the as
```

In [26]:
```
print('Confusion matrix in Training')
print(confusion_matrix(t_train, predict_train))

print('Confusion matrix in Test')
print(confusion_matrix(t_test, predict_test))
```

```
Confusion matrix in Training
[[72  0]
 [ 0 68]]
Confusion matrix in Test
[[28  0]
 [ 0 32]]
```

The confusion matrix show that all samples were correctly classified for both training and test sets.

Note that if you go back, and use a very large value for λ, say $\lambda = 10$, the confusion matrices will display a lot more errors. This large regularizer will distort the data significantly to impact the classification results.