DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 01:

HW No. 01: Gaussian Distributions

submitted to:

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

January 22, 2022

prepared by:

Gavin Koma
Email: gavintkoma@temple.edu

A. DESCRIPTION OF TASK 1

The goal of Task 1 is to populate data values given a covariance matrix and a mean for each of four classes. A covariance matrix is a square matrix that describes the covariance—the joint variability of two variables—given a vector. Thus, the primary coding goal of this assignment is to generate data that is described by the given mean and the covariance matrix.

In order to do so, I chose to utilize python and two common imported libraries: pandas and numpy. I then defined the four classes and the four corresponding mean values for each class. This can be seen in the following snippet of code:

```
#define the values as per the homework assignment
classes = [*range(1,5)]

mean1 = [1,1]
mean2 = [1,-1]
mean3 = [-1,-1]
mean4 = [-1,1]

means = (mean1,mean2,mean3,mean4)#condense for ease

#define the diagonal covariance matrix that we will use later
cov = [[0.1,0], [0,0.1]]
```

The means were condensed into a list to facilitate work with later for-loops. In order to create our data distribution, I used numpy's random.multivariate_normal function which allows one to draw random samples from a multivariate normal distribution. This function allows us to generate a dataset while providing a given mean and a covariance matrix (both of which we are given in the homework assignment). I elected to create a dictionary which will allow me to for-loop through both the varying classes and their means instead of repeating the same lines of code. This was performed as follows:

```
#now i guess we should create the data samples
#use the covariance matrix to generate data

#can just use a forloop of sorts to sort this
#but we would need to make a dic for this
d = {}
for val in means:
    for k in classes:
        d["class{0}data".format(k)] = np.random.multivariate_normal(val, cov, size=100)
```

This nested for-loop creates an output of four separate data frames; each of which contains an x-vector and a y-vector. At this point, I was unsure of how to use further loops or even list comprehension to facilitate the creation of our final concatenated data frame. I was able to complete this by using a naïve method to assign column names and create a final data frame.

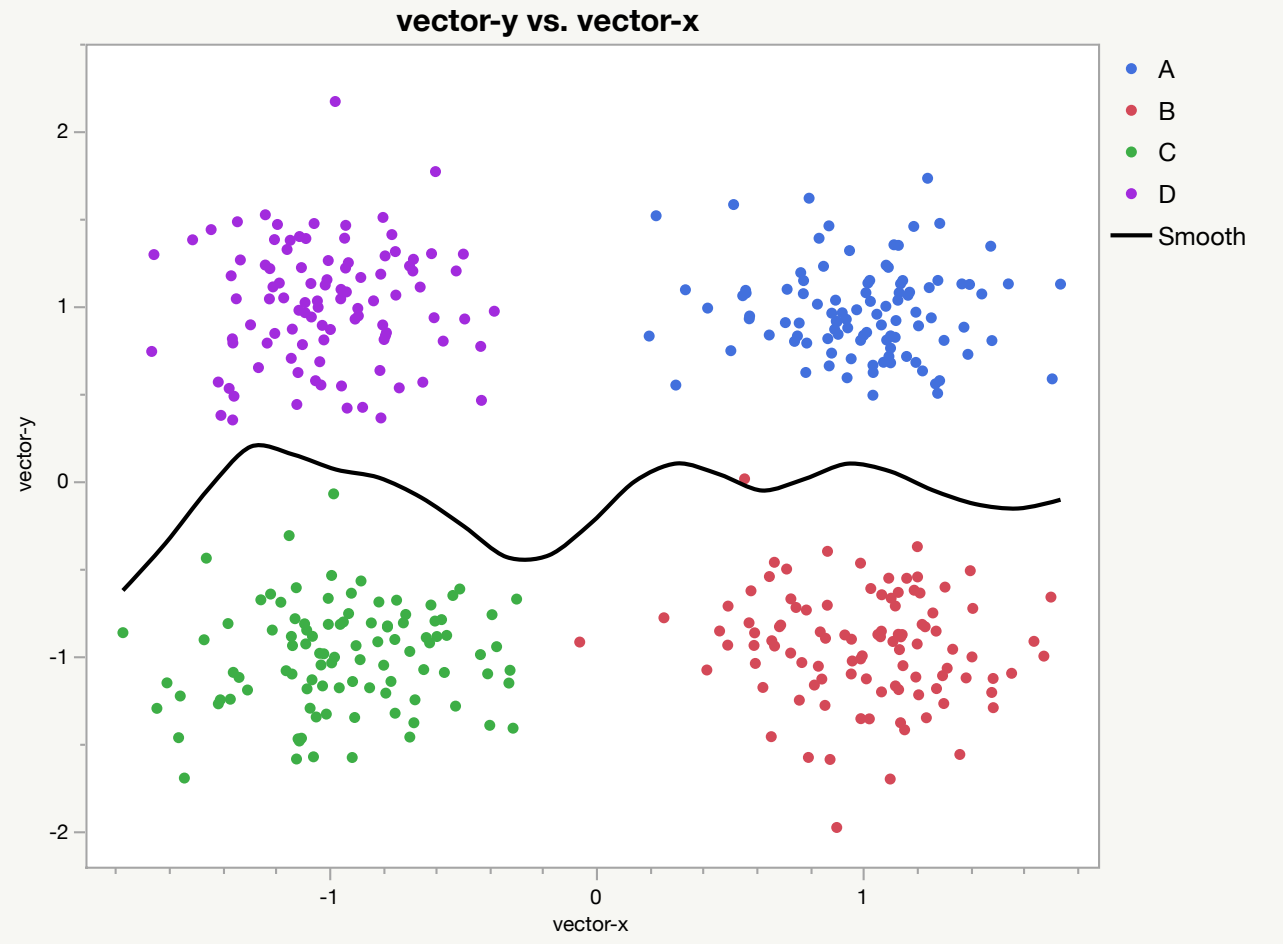
```
dfclass1 = pd.DataFrame(d['class1data'],columns=['class1columnx','class1columny'])
dfclass2 = pd.DataFrame(d['class2data'],columns=['class2columnx','class2columny'])
dfclass3 = pd.DataFrame(d['class3data'],columns=['class3columnx','class3columny'])
dfclass4 = pd.DataFrame(d['class4data'],columns=['class4columnx','class4columny'])

#conjoin the dataframes for ease of use in jmp
df_final = pd.concat([dfclass1,dfclass2],axis=1)
df_final = pd.concat([df_final,dfclass3],axis=1)
df_final = pd.concat([df_final,dfclass4],axis=1)

df_final.to_csv("data_part1.csv",header=True,index=False) #save to csv
```

This final data frame was then saved as a .csv file and was then imported into JMP Pro 16 for further analysis.

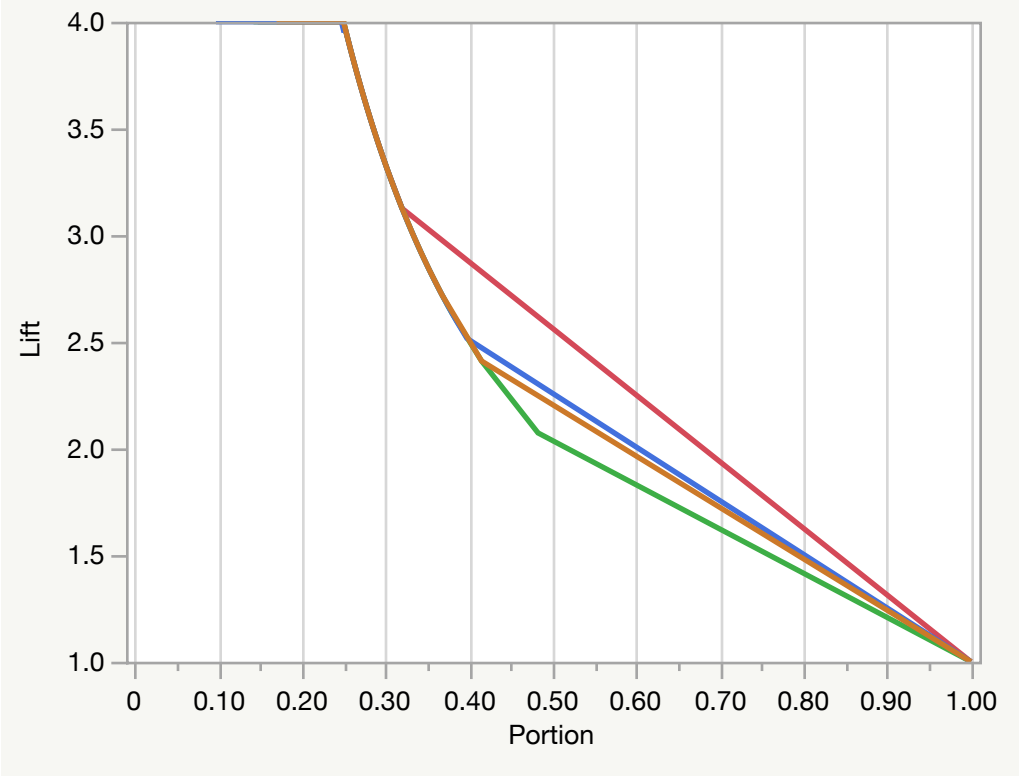
JMP Pro 16 provides an extremely convenient method for performing the Naïve-Bayes Predictive Modeling method. With JMP Pro 16 open, one can select their data and in the Analyze menu, can choose the Naïve-Bayes method. However, before we can do that, we must first organize our data into in three total columns. One column will assign labels of class where Class 1 is A, Class 2 is B, etc…, the second column will contain all x-vector components, and the third column will contain all y-vector components. Once properly organized, the data in JMP should look similar to this image on the right.

When selecting data for the Naïve-Bayes Predictive Modeling, I specified 20% of the data would be used for validation while the remaining 80% will be used for the training of our model. Based on the given data, we will be using our x-vector and our y-vector values (the factors) to be predicted class (the response). Before proceeding with the Naïve-Bayes Classifier, I used graph builder to ensure that all vectors have the proper means. The Graph below shows four separate groups of data that each have very clearly outlined means.

class	vector-x	vector-y
A	0.975547147	0.977666364
A	1.222091529	0.627813342
A	0.514839588	1.579443132
A	1.115096676	1.349354887
A	1.036360854	0.488830827
A	0.198725698	0.827163199
A	1.068130098	0.890839471
A	1.131699575	1.346025433
A	0.548950173	1.05811462
A	1.477434332	1.341165657
A	1.037323851	0.618813992
A	0.766147108	1.190012982
A	0.881724991	0.729580754
A	0.893141449	0.865486435
A	0.742879991	0.796880022
A	1.00126537	0.831054946
A	0.76035754	0.901797398
A	0.573417088	0.926059831
A	1.392055393	0.722615716



Once outputted, we are able to see that of the total 400 sample count, the Misclassification Rate of our data for training and validation with the given covariance matrix of [[0.1,0], [0,0.1]] is 1.0 and 1.0 respectively. The classifier also provides us with a Lift Curve for the class that is calculated via Training Data. The Lift Curve shows us a plot of the lift against the portion of the observations. Our Lift Curve is a partition model that is built to predict the response between 4 binary classifications: Class A, Class B, Class C, and Class D.



The lift chart that we see above is the given ratio of the number of positive observations to the expected number of positive observations. It is important to note that the greater the area between the Lift line and the baseline of the chart, then we can assume the better the model.

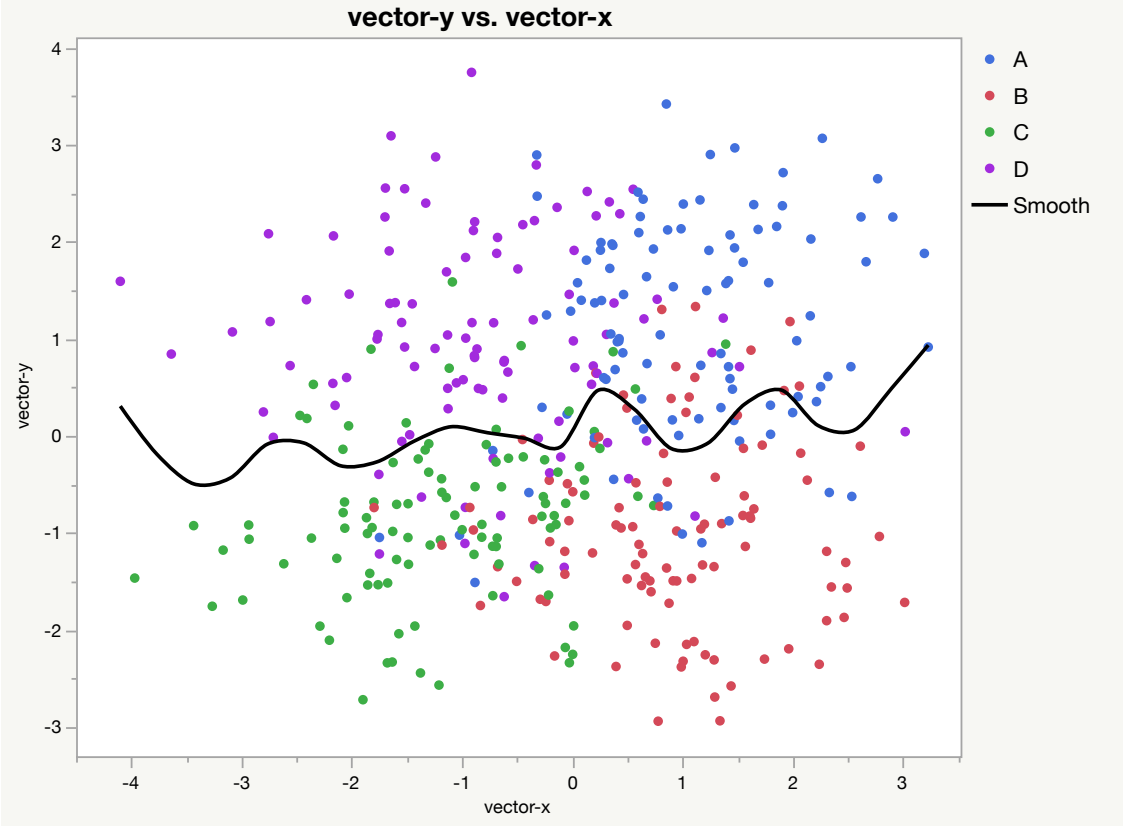
B. DESCRIPTION OF TASK 2

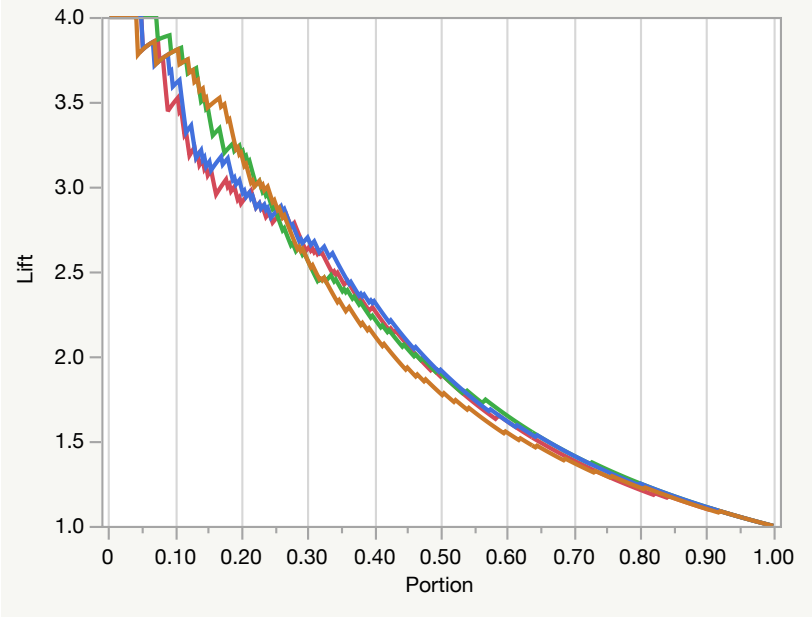
The goal of Task 2 is the same as the goal of Task 1: to generate a dataset of four classes with a given mean and a given covariance matrix. The difference, however, is that the covariance matrix has been altered from [[0.1,0], [0,0.1]] to [[1.0,0], [0,1.0]], thus inducing more variance in our generated dataset. The same code that was utilized in part 1 was recycled for use in part 2. A screenshot of this code was included on the following page for ease of understanding. Note that the covariance matrix values have been changed as indicated but the rest of the code has been unchanged. The data was also reorganized in JMP Pro 16 to fit the needs for a Naïve-Bayes analysis. This screenshot has been excluded and if needed, an idea of the format can be referenced from Task 1.

```
#%%part 2
#increase covar
#define the diagonal covariance matrix that we will use later
cov_2 = [[1.0,0], [0,1.0]]

d = {}
for val in means:
    for k in classes:
        d["class{0}data".format(k)] = np.random.multivariate_normal(val, cov_2, size=100)

dfclass1 = pd.DataFrame(d['class1data'],columns=['class1columnx','class1columny'])
dfclass2 = pd.DataFrame(d['class2data'],columns=['class2columnx','class2columny'])
dfclass3 = pd.DataFrame(d['class3data'],columns=['class3columnx','class3columny'])
dfclass4 = pd.DataFrame(d['class4data'],columns=['class4columnx','class4columny'])

#conjoin the dataframes for ease of use in jmp
df_final = pd.concat([dfclass1,dfclass2],axis=1)
df_final = pd.concat([df_final,dfclass3],axis=1)
df_final = pd.concat([df_final,dfclass4],axis=1)


df_final.to_csv("data_part2.csv",header=True,index=False) #save to csv
```

The Naïve-Bayes classifier is a simple classifying method that applies Bayes' theorem with a strong assumption of independence between features. The essence of Bayes' theorem is that it allows us to find the probability of some event A happening given that another event B has occurred. The independence assumption that was previously mentioned is again that our events are independent of each other.

Once again, I first use the graph builder to visualize the data and check that the data are relatively separated into 'neighborhoods' based on their means. Now that we have increased the covariance of the datasets, we should see notable more overlapping between datasets.





Our Lift Curve for Task 2 data varies much more and shows a less smooth lift curve. We are also able to see more misclassifications than in Task 1. For Task 1 we saw a 100% classification rate which was due to how much less varied our data being used was. However, in Task 2, we see a misclassification rate of 0.27 meaning that 27% of the classifications were incorrect.

C. SUMMARY

The Naïve-Bayes classifier is a strong contender for classification algorithms. The model is easy to implement using JMP Pro 16 and another complimentary computer language to prepare and sort or to generate data. This homework provided us with a general understanding of the use and functions of the Naïve-Bayes model and helped us to understand the effects of higher variance within data.

This model classifies data from vectors into some chosen binary classification. For our project, we classified data into four separate classes based on two generated vectors: once with a small degree of variance and another time with a larger degree of variance. One of the biggest takeaways from this homework assignment was the knowledge that the Naïve-Bayes classifier works extremely well with data that does not necessarily overlap. However, when supplied with data with more variance and therefore a larger degree of overlapping, our model began to fail with the classifications.