Homework Assignment No. 02:

# HW No. 02: Bayesian Decision Theory

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

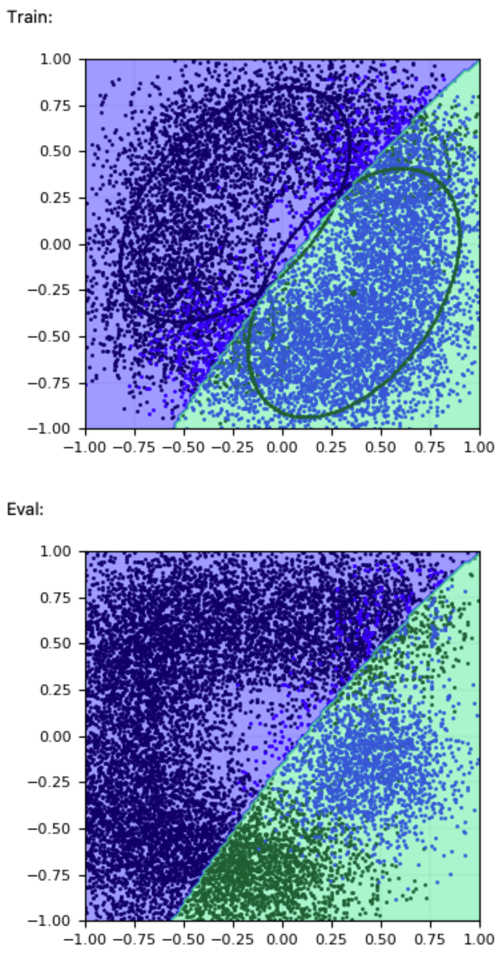
January 24th, 2022

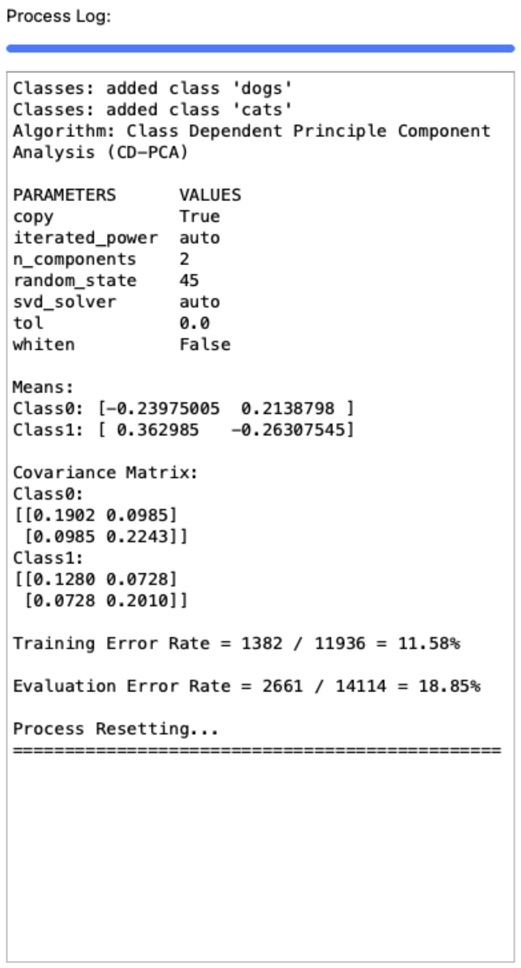
prepared by:

Gavin Koma
Email: gavintkoma@temple.edu

## A. IMLD.PY

imld.py is a custom GUI that was designed by Dr. Picone's lab and offers a simple and effective way to implement various machine learning algorithms. The first task of this assignment is to utilize this executable and perform "class-dependent principal components analysis". Doing so, provides a window that looks like this:



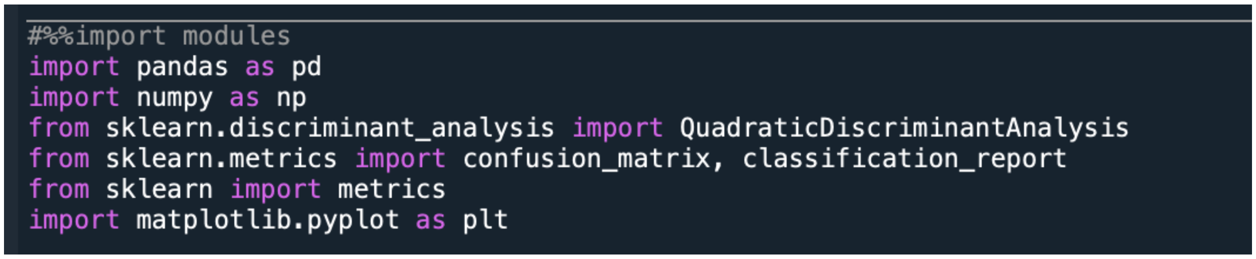


```
Classes: added class 'dogs'
Classes: added class 'cats'
Algorithm: Class Dependent Principle Component
Analysis (CD-PCA)

PARAMETERS      VALUES
copy            True
iterated_power  auto
n_components    2
random_state    45
svd_solver      auto
tol             0.0
whiten          False

Means:
Class0: [-0.23975005  0.2138798 ]
Class1: [ 0.362985   -0.26307545]

Covariance Matrix:
Class0:
[[0.1902 0.0985]
 [0.0985 0.2243]]
Class1:
[[0.1280 0.0728]
 [0.0728 0.2010]]

Training Error Rate = 1382 / 11936 = 11.58%

Evaluation Error Rate = 2661 / 14114 = 18.85%

Process Resetting...
============================================
```

One can see the training error rate and the evaluation error rate which are 11.58% and 18.85%, respectively. In order to maintain similarity between imld.py and my own python scripts, I have altered the report and instead of reporting the miscalculation rate, I have included the percent of properly classified data. This was done by subtracting the error rate from 100. Thus, my accuracy rate was 88.42% and 81.15% for the training data and evaluation data.

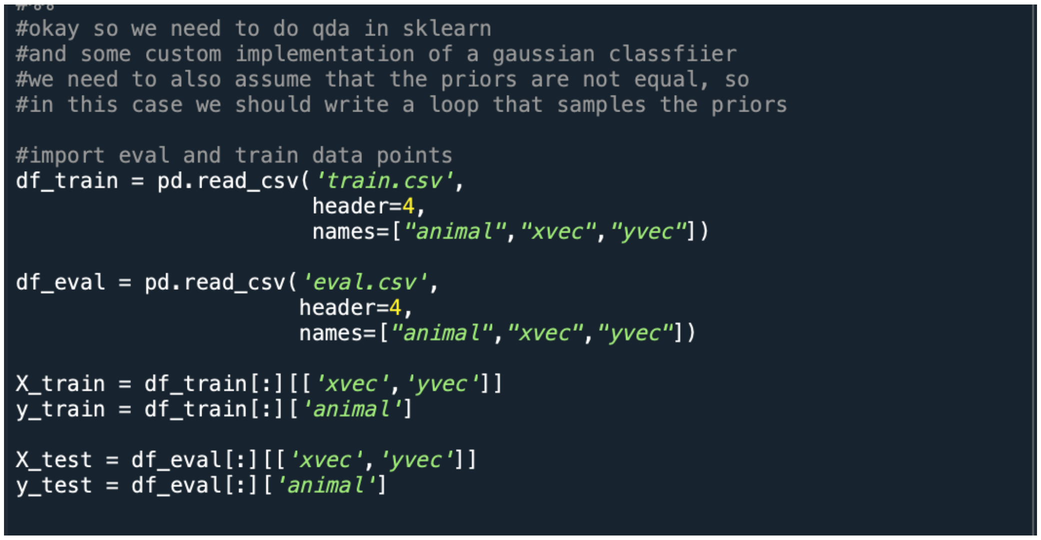
## B. QDA PERFORMED IN JMP, SKLEARN, AND SOME SIMPLE GAUSSIAN CLASSIFIER

The first step of any coding project requires the implementation of the proper python modules. To complete this assignment, I utilized pandas, numpy, various portions of sklearn, and matplotlib for basic plotting.



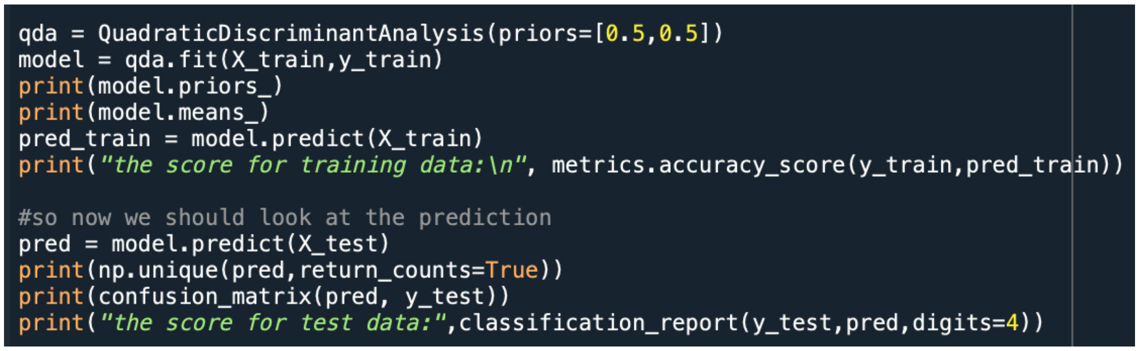
```
#%%import modules
import pandas as pd
import numpy as np
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, classification_report
from sklearn import metrics
import matplotlib.pyplot as plt
```

In addition to proper modules, the data must first be imported into one's workspace. SKLearn offers great methods for splitting data into test and evaluation sets. Luckily, we did not need to use this tool as our data was provided in two separate data files: train.csv and eval.csv. I did, however, decide to read in my csv files into a manner that allowed me to easily differentiate between train and test data. This is not a necessary step but it makes the data easier for me to work.



```
#okay so we need to do qda in sklearn
#and some custom implementation of a gaussian classfiier
#we need to also assume that the priors are not equal, so
#in this case we should write a loop that samples the priors

#import eval and train data points
df_train = pd.read_csv('train.csv',
                       header=4,
                       names=["animal","xvec","yvec"])

df_eval = pd.read_csv('eval.csv',
                      header=4,
                      names=["animal","xvec","yvec"])

X_train = df_train[:][['xvec','yvec']]
y_train = df_train[:]['animal']

X_test = df_eval[:][['xvec','yvec']]
y_test = df_eval[:]['animal']
```

The assignment specifies that we should assume that the priors are equal in this case and therefore require us to assign the priors of dogs and cats to be [0.5,0.5]. I also have a variety of print statements that I use to check my own work as I progress, this print statements can be ignored if desired. The assignment also dictates that the error rate be reported for *both* evaluation data and training data.



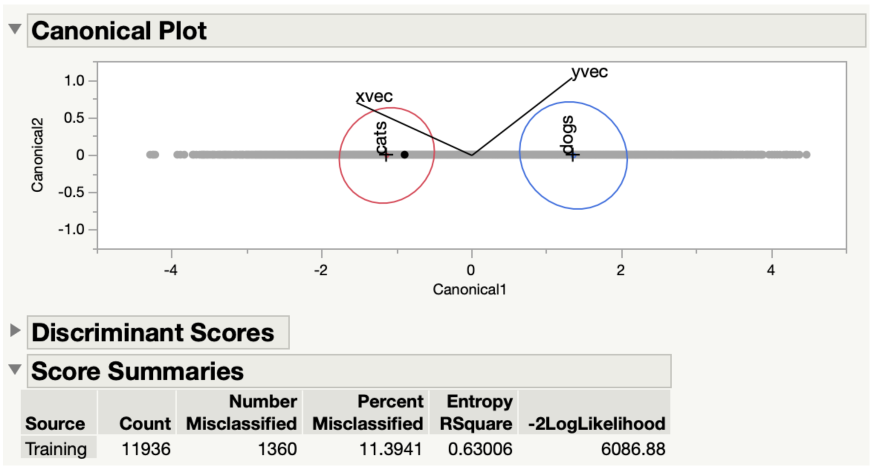
```
qda = QuadraticDiscriminantAnalysis(priors=[0.5,0.5])
model = qda.fit(X_train,y_train)
print(model.priors_)
print(model.means_)
pred_train = model.predict(X_train)
print("the score for training data:\n", metrics.accuracy_score(y_train,pred_train))

#so now we should look at the prediction
pred = model.predict(X_test)
print(np.unique(pred,return_counts=True))
print(confusion_matrix(pred, y_test))
print("the score for test data:",classification_report(y_test,pred,digits=4))
```

One can see that using QDA in SKLearn only requires three lines of code. The first is the calling of the quadratic discriminant analysis with predefined priors. We then use the x vector and y vector from our training data to fit the model. Our prediction for training is the third line of code where we have our model predict for the training data. To predict for the test data, we provide out model with the X_test vector.

To obtain the scores, we include the print functions seen above. I specified a string within each print statement to better determine which number corresponded to the desired output in my terminal. SKLearn outputted a score of 88.61% and 80.09% for training data and evaluation data, respectively.

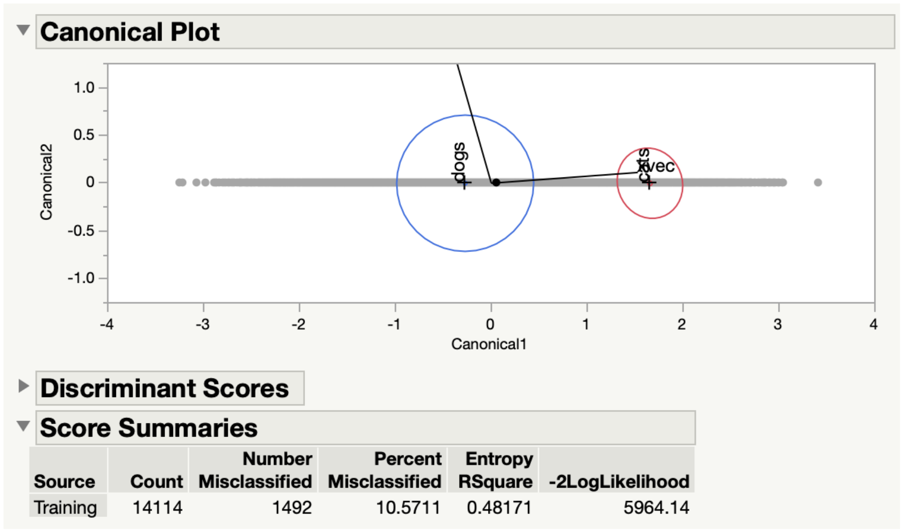
Performing the same procedure in JMP Pro is notably easier; however, I am unsure of its effectiveness when working with a dataset that has not already been organized to facilitate such a procedure. In order to perform this, one must load their data and then go to Analyze > Multivariate Methods > Discriminant. When the discriminant pop-up shows, it is important to change the Discriminant method which is by default, Linear, Common Covariance to Quadratic, Different Covariances; as per our assignment instructions. At this point, your continuous vectors will be placed in the Covariates box while the animal category will be placed in Categories. Your outputs for training data and evaluation data will be as follows:

Training Data:



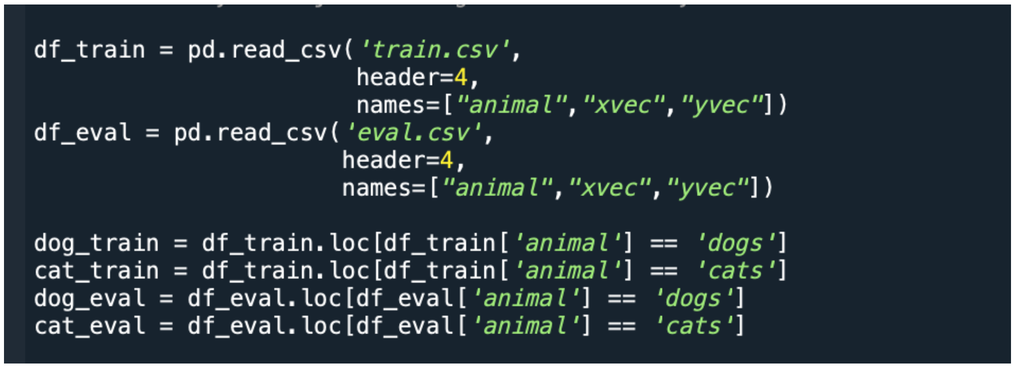
| Source | Count | Number Misclassified | Percent Misclassified | Entropy RSquare | -2LogLikelihood |
|---|---|---|---|---|---|
| Training | 11936 | 1360 | 11.3941 | 0.63006 | 6086.88 |

Evaluation Data:



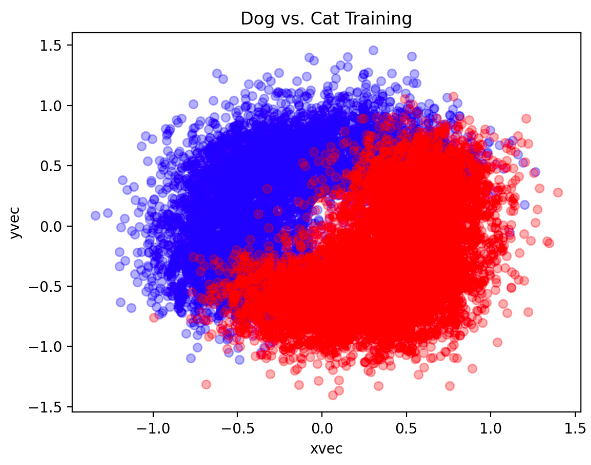
| Source | Count | Number Misclassified | Percent Misclassified | Entropy RSquare | -2LogLikelihood |
|---|---|---|---|---|---|
| Training | 14114 | 1492 | 10.5711 | 0.48171 | 5964.14 |

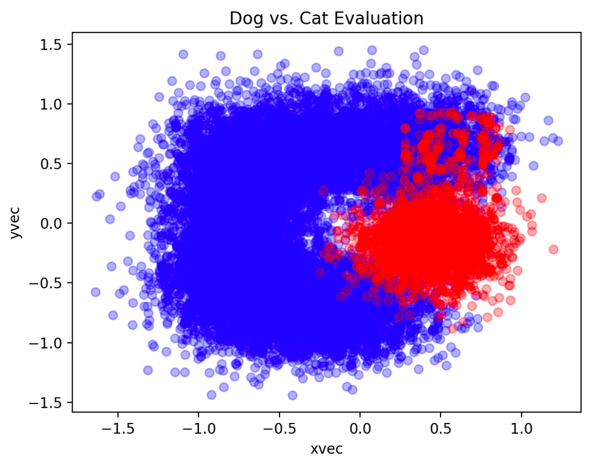
The final objective in this task is to utilize some self-chosen simple Gaussian classifier. We have spent a good portion of time discussing Naïve-bayes and therefore, I have elected to implement a Naïve-bayes Gaussian classifier. To do so, I prefer to segment my code, and reimport data points to ensure a clean and fresh start for my algorithm. Therefore, I import all of my data and rename my variables to fit my needs.



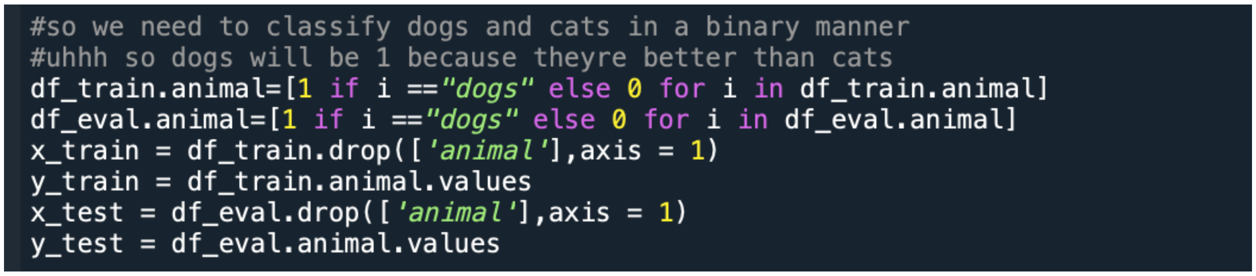
```
df_train = pd.read_csv('train.csv',
                       header=4,
                       names=["animal","xvec","yvec"])
df_eval = pd.read_csv('eval.csv',
                      header=4,
                      names=["animal","xvec","yvec"])

dog_train = df_train.loc[df_train['animal'] == 'dogs']
cat_train = df_train.loc[df_train['animal'] == 'cats']
dog_eval = df_eval.loc[df_eval['animal'] == 'dogs']
cat_eval = df_eval.loc[df_eval['animal'] == 'cats']
```

Another precaution that I have taken is to plot the training data and the evaluation data. I was able to briefly visualize the data when using imld.py but I have chosen to perform this action again as a safety measure to be sure that my data has not be drastically altered. I have chosen not to include a snippet of this code as it is relatively simple and can be performed with little to no coding experience. The two graphs look as such:





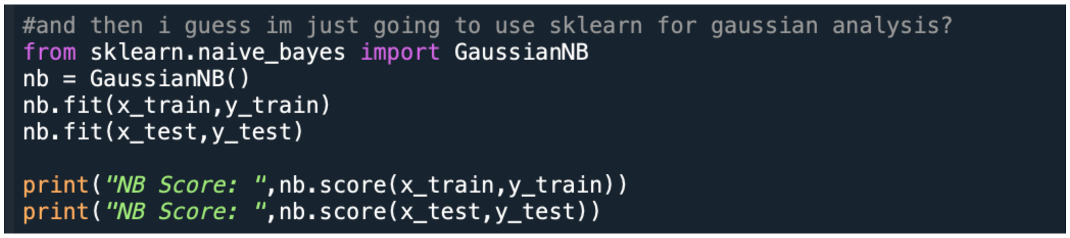
From here, I take time to binary classify the animals. I have chosen 1 for dogs and 0 for cats. I am, unfortunately, a bigger fan of dogs and that is why I gave them 1. However, these numbers can be chosen arbitrarily as they only represent a categorical value for the two animals.



```
#so we need to classify dogs and cats in a binary manner
#uhhh so dogs will be 1 because theyre better than cats
df_train.animal=[1 if i =="dogs" else 0 for i in df_train.animal]
df_eval.animal=[1 if i =="dogs" else 0 for i in df_eval.animal]
x_train = df_train.drop(['animal'],axis = 1)
y_train = df_train.animal.values
x_test = df_eval.drop(['animal'],axis = 1)
y_test = df_eval.animal.values
```

The code functions by iterating through the animal column of both the evaluation data and the training data. The values in the animal column become 1 if the string is "dogs" and 0 if the string is anything else. In this case, all dogs become 1 and all cats become 0. Our variables then become y_train which has all of the categorical representations of the dogs and cats and x_train which is a two column dataframe which contains the original vectors corresponding to each animal. The same can be said for y_test and y_train which will be used to evaluate our algorithm after training.

In order to obtain the values for accuracy, we finish our code by calling the SKLearn Naïve-Bayes Gaussian Classifier. This is done as follows and gives us a score of 70.29% and 93.86% for training and evaluation data:



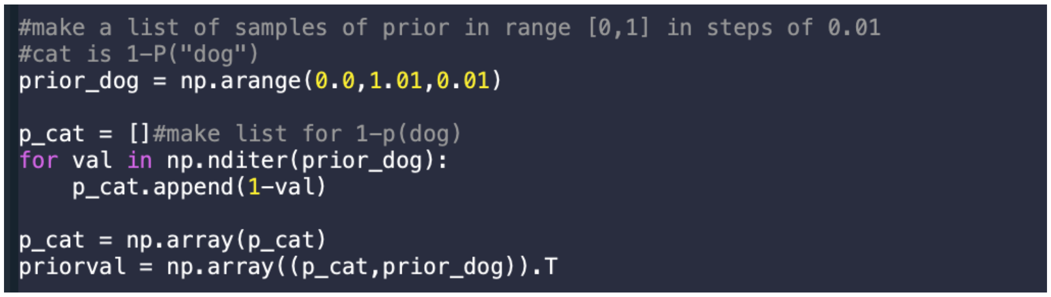
```
#and then i guess im just going to use sklearn for gaussian analysis?
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train,y_train)
nb.fit(x_test,y_test)

print("NB Score: ",nb.score(x_train,y_train))
print("NB Score: ",nb.score(x_test,y_test))
```

We see that these data values (also found in the table below) are slightly different but overall quite similar when comparing the JMP Pro 16 QDA and the QDA performed by SKLearn. The differences between these classifiers can be a byproduct of several reasons. For these classifiers, we are utilizing a Gaussian Multivariate Distribution in order to obtain an estimation of the mean and the covariance for each given class. This method also incorporates a small degree of randomness when we perform our estimation. I believe that this is the main reason between the small differences in accuracy. Another reason for the discrepancy between accuracy between QDA and the Naïve-Bayes Gaussian Classifier is the assumptions that each one makes. QDA assumes that correlation is a minimal factor to be considered while the Naïve-Bayes Gaussian Classifier assumes that all features are independent of each other.

## C. GRAPH COMPARISON WITH VARYING PRIORS

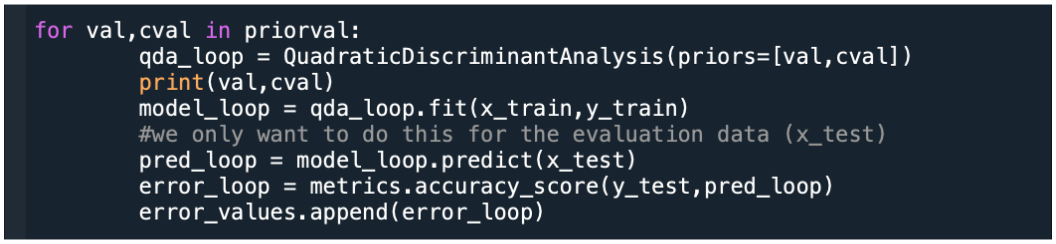
This task requires us to assume that the priors are not equal. To do so, I implemented a loop that populates a list of 101 values over the range of [0,1] with a step-size increment of 0.01. It was necessary to make the range command go from 0.0 to 1.01 to ensure that 1.0 was included in the final output.

I then created another list for the priors of the cat (p_cat) by writing loop that will subtract the each value in the dog priors list and append the given value to p_cat. Finally, I concatenated the two numpy arrays into one array and transposed the matrix to have two columns.



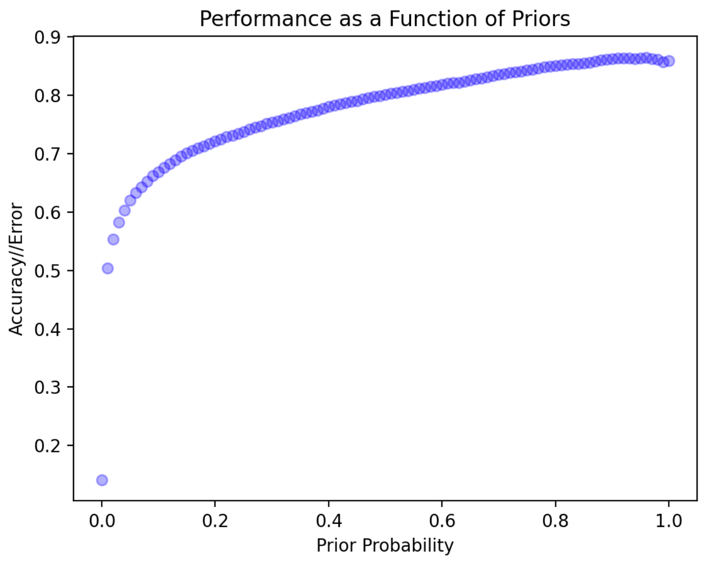
```
#make a list of samples of prior in range [0,1] in steps of 0.01
#cat is 1-P("dog")
prior_dog = np.arange(0.0,1.01,0.01)

p_cat = []#make list for 1-p(dog)
for val in np.nditer(prior_dog):
    p_cat.append(1-val)

p_cat = np.array(p_cat)
priorval = np.array((p_cat,prior_dog)).T
```

I concluded this task by calling on the SKLearn Quadratic Discriminant Analysis in a loop that pulled two values at a time from my concatenated numpy array and utilized them as priors.

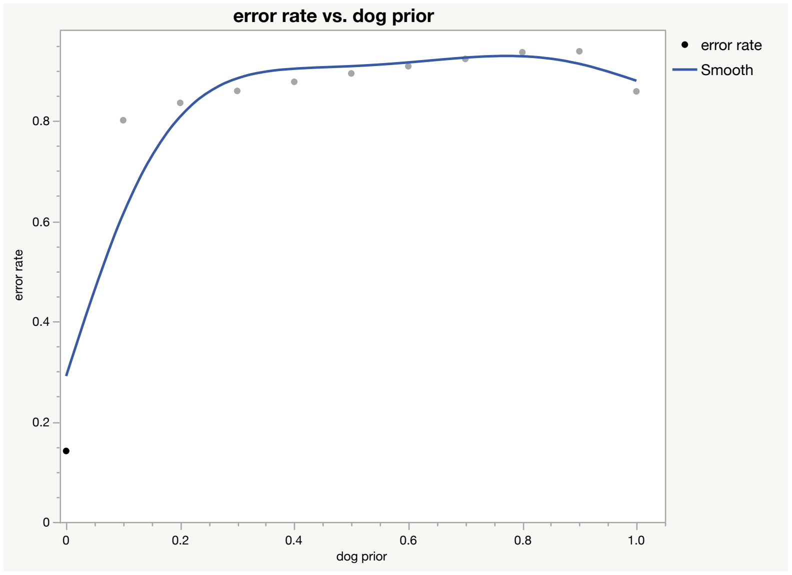


```
for val,cval in priorval:
        qda_loop = QuadraticDiscriminantAnalysis(priors=[val,cval])
        print(val,cval)
        model_loop = qda_loop.fit(x_train,y_train)
        #we only want to do this for the evaluation data (x_test)
        pred_loop = model_loop.predict(x_test)
        error_loop = metrics.accuracy_score(y_test,pred_loop)
        error_values.append(error_loop)
```

By calling the QDA in this manner, it allowed me to easily perform 100 analyses in a short time span and record the errors. The QDA loop uses each pair of numbers from my dog and cat priors array and utilizes them as the priors. I use the same training data for each training and use the same data as previously to evaluate the work as well. I append all of the error values at the end of the loop into an empty array that was previously defined as error_values = []. Utilizing a few short lines of code, I was able to obtain the following plot of error values:



This assignment requires me to also vary the priors in JMP Pro 16 and compare the graphs. In order to do this in JMP, I elected to save myself some time and varied the priors by 0.1 instead of 0.01. This gives me enough data values that I can compare the graph without having to manually change the priors 100 times. I am not advanced enough in using JMP to know if it is possible to write a script to do this efficiently. The values for the dog priors were placed on the x-axis and the error rate was placed on the y-axis. The smoothed line looks different from the python graph but that is because it is accounting for the lack of data points. The points themselves, however, look extremely similar, if not identical, to the data values that were plotted by python.



The main difference is that although the graphs look the same in terms of shape, the error rates provided by JMP are all above .8 albeit two of the data points. SKLearn, however, outputs error rates that are anywhere between 0.65 to 0.85. We see a notably wider range but with a near exact same shape being output from SKLearn when compared to JMP Pro. This could be because we are only using the evaluation data when calculating the priors in JMP Pro where as in SKLearn, we are accounting for *both* training and evaluation data.

## D. TABLE SUBMISSION

The following table was filled out during the coding process and was included below. Some data cells were not required to be filled out and instead of leaving them blank, I have elected to fill their spaces with N/A.

| | | **IMLD** | | **JMP** | | **SKLearn** | | **Python** | |
|---|---|---|---|---|---|---|---|---|---|
| **Algorithm** | **Data** | **Train** | **Eval** | **Train** | **Eval** | **Train** | **Eval** | **Train** | **Eval** |
| CD-PCA | Set No. 13 | 88.42% | 81.15% | N/A | N/A | N/A | N/A | N/A | N/A |
| QDA | Set No. 13 | N/A | N/A | 88.61% | 89.43% | 88.61% | 80.09% | N/A | N/A |
| Naïve Bayes | Set No. 13 | N/A | N/A | N/A | N/A | N/A | N/A | 70.29% | 93.86% |

## E. SUMMARY

This assignment's goal was to further explore Bayesian Decision theory as well as dabble in Gaussian distributions. The assignment helped greatly to apply the math that we have studied in lecture. I regularly find myself struggling to make connections between high-level mathematics and real-world applications. Although I understand that this is not the main goal of this course, it greatly helps with my overall understanding.

I am particularly curious as to whether or not there is a way to perform some of these classifiers in JMP Pro if we do not have them already separated into organized datasets. As I progressed through this assignment, I was unable to see a way to apply a training dataset and a test dataset in JMP Pro, unlike in python where one is easily able to split data as necessary.

One aspect of my approach to solving these homework assignments going forward will be to implement more generalized functions. There were a variety of times where instead of writing a script that I could have defined a function and passed variables to it. I understand that as someone who would not describe themselves as “experienced” in coding that I need to start practicing these techniques to better incorporate and tidy my code.

Overall, this assignment helped to further cement the Naïve-Bayes classifier into my mind and helped to better understand some of the variations that these programs might cause in data analysis. I had not previously considered how using JMP Pro or python may result in small discrepancies that could affect future decisions with a dataset.