**Department of Electrical and Computer Engineering**

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

# 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:

A picture containing polygon

Description automatically generated

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.

# 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.

Graphical user interface, text

Description automatically generated

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.

Text

Description automatically generated

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.

Text

Description automatically generated

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:  Diagram, venn diagram  Description automatically generated |

|  |
| --- |
| Evaluation Data:  Diagram  Description automatically generated |

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.

Text, chat or text message

Description automatically generated

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:

Chart, scatter chart

Description automatically generated

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.

Graphical user interface, text, application, chat or text message

Description automatically generated

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:

Text

Description automatically generated

# 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.

Text

Description automatically generated

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.

Text

Description automatically generated

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:

Chart

Description automatically generated

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.



# 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.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Data** | **IMLD** | | **JMP** | | **SKLearn** | | **Python** | |
| **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% |

# Summary