CSC 74020 Machine Learning Midterm project
Bayes Rule Classifier

Xinyue Chang

May 2021

1 Abstract

This model is created for the midterm project for machine learning class at CUNY graduate center, fall semester, 2020. The purpose of this model is to explore the implementation of Bayes rules in machine learning. This model is mainly based on Bayes rule to train the dataset and consider the expected gain at the same time while building up the decision rule. The program language used for coding this model is python 3.0. The data in the dataset used in this model is followed by the cumulative probability distribution created randomly by the dataset generator built in the model. There are totally 5 dimensions, and 3 different classes in the dataset and two possible values for each dimension. Besides cresting the decision rule, the model also performed a cross validation and optimization, both helped to improve the performance of the model and produce a better result.

2 Introduction

The Bayes rules are widely used in calculating joint probabilities and conditional probabilities. And it is a very basic and important concept in statistical and data analyst machine learning. It is very common to use Bayes rule in both regressions and classification to calculate all the different probabilities and use for training the dataset to generate the information needed for the decision rule. With enough training data, using Bayes' Theorem can be used to build a very powerful model for classifiers and prediction for the unknown.

I am building this classifier for my midterm project and I think this whole process really helps me have a deeper understanding about the basic concept of Bayes rules and also applied the theory into building up the model to solving the real problem. The main goal of the classifier is to create a decision rule that can achieve the total economic expected gains as high as possible. The inputs for this model are a dataset which created by the dataset generator; a 3 by 3 economic gain matrix, because there are a total 3 classes in the dataset; the prior probabilities for each class, and the data set for class i ($< x_1, ..., x_z >$

$, < c_1, ..., c_z >$). And the outputs from this model are the Discrete Bayes Rule $fd(C)$, a 3 by 3 confusion matrix, as well as the expected gain.

After the model was built, I also did data shuffling, cross valuations and optimization to explore the data and to discover what effects can improve the performance of the model to achieve a better result. By shuffling the data set and doing cross valuations, allows the model to decrease the noise created by the order of dataset, and exposed the learning process to different part of the dataset. By doing the optimization, it helps the model to modify and adjust the conditional probability by a small adjustment, so that increases the performance of the model. I will elaborate more details about my project, and the problem that I had during the process in the following paper.

3 Technical Process

3.1 The data set generator

The first step for this project is to create the dataset. The dataset I generated has 5 dimensions, size of 100,000, and classified into 3 classes. To do so, I created a one-dimensional sample data generator first, and repeat the generating process 5 times to create 5 one-dimensional sample data and combine them together as final data set, and then randomly assign those data into 3 classes. This one-dimensional sample generator first will create a cumulative probability distribution based on how many possible values are in the sample, and then generate a certain size of sample data from the cumulative probability distribution that was created before.

Here are the mathematical definitions of the above procedure: let $X_i(a, n)$ represents there are a numbers of values for component Xi, and with total n numbers of data, and the result of the whole generating process is a length n list $< x_1, x_2, ...x_n >$, and all value of those elements are in [1, a]. Notice that a and n are user input parameters, and n should be the same for all dimensions. First, I load two list of uniform [0,1] pseudo random numbers: $< v_1, v_2...v_a >$, and $< k_1, k_2, ...k_n >$. Then normalized the sum of $< v_1, v_2...v_a >$ to create the probabilities $< q_1, q_2...q_a >$ by:
For $i \in 1...a$:

$$q_i = \frac{v_i}{\sum_{i=1}^{a} v_i} \tag{1}$$

Then we can form the Cumulative Distribution $< p_0, ..., p_j >$:
For $j \in 1...a$:

$$P_j = \sum_{i=1}^{j} q_i \tag{2}$$

After the Cumulative Distribution are calculated, I find the value of x.
For $m \in 1...n$: if $p_j < k_m < p_j + 1$, then x_m=$j + 1$. Notice that p_0 equals to 0.

For example, in my model, there are 2 possible values for the first component of the measurement tuple, and the total length of the data set is 100,000. The

first component is represented by X_1 in the following essay, which represented as $X_1(2, 100, 000)$. The generator first creates 2 uniform[0,1] pseudo random numbers, and by normalizing this set of numbers, it creates the Cumulative Probability Distribution for X_1. Then the generator randomly creates a list of 100,000 number of uniform [0, 1] pseudo random numbers, and for each number in this list, if it falls in the interval [0,0.5) it would be given a value 1 and if it falls in the interval (0.5,1], it would be given a value 2. I used this 1-D sample generator to create a total 5 lists of one-D data lists, each dimension has 2 possible values, and combined them together as my 5-dimension data set.

For the data set class tag, I used the same generator to create a list of class tags which has 3 values with the size same as the data set I created in the last step. And by concatenating this class tag list to the 5-dimensional data. Now the data set generating process is completed.

Meanwhile, I have the prior probabilities $Pr(C_i)$ for each class, which can be generated from class tag's Cumulative Distribution: $Pr(C_i) = P_i - P_(i-1)$.

The next step is to separate the training data and testing data, in my model I used the first 70% of the data I created as my training data set and the rest 30% as my testing data set.

3.2 The data training process

The calculation in this training process is based on the Bayes Theorem equation:

$$Pr(Ci, d) = Pr(d|Ci) * Pr(Ci), \tag{3}$$

To assign any C_k to d such that the $\sum_{j=1}^{k} e\ (C_J,\ C_K\ )Pr(C_J, d)$ is maximized, it has to find out the largest project of joint probability $Pr(Ci, d)$ for each d and expected gain $e(C_J, C_K)$. The prior probabilities $Pr(C_i)$ and expected gain $e(C_J, C_K)$ are known. The conditional probabilities are the probabilities of each tuple in the given class. Categorize dataset into subsets based on the classes. In each class's subset, I count the frequency of each d , and divided it by the total subset length, which is $Pr(d|C_i)$, by doing this for every d in each class, I have all the conditional probabilities for each d in each of each class. I make my data set to a DataFrame and use the groupBy.() function in pandas to separate the class, and collections function from counter to organize the summarize all the d in each class.

3.3 Bayes decision rule

Following the rule of picking the class that maximize the product of $e(C_J, C_K) * Pr(C_J, d)$, the model will go through each data point and find the same data point in the 3 different data subsets to find the conditional probabilities, and the multiply by each class's prior probabilities, and the true assignment expected gain value. After this calculation, the model will pick the class which has the largest product and assign that class to the measurement d. After the model go through all the testing measurements, it created a new assigned classes for each

d in the testing data set. Since the 3 classes data subsets are all data frame, I use the loc.() function in panda to locate the data point location and find the conditional probability for each d in each class.

3.4 Confusion matrix

The confusion matrix shows how the decision rule performed. The model combines the true class list and assigned class list that decision rule created in last step together, and create a new list of class tag combination for testing set, and then the model counts the frequency for each combination and divided it with totally length of the list to find out the probabilities for each combination to conduct the confusion matrix. The sum of the diagonal entries of the confusion matrix is the identification accuracy rate, which indicates how accurate the decision rule is.

3.5 Expected gain

The expected economic gain is the value of the possible gain for all situations. and the confusion matrix shows all the probabilities for each combination, so we take the probabilities from the confusion matrix and multiply by the economic gain under that combination of true class and assign the class. The economic Gain Matrix used in my example has value of 2 for $e(c_j, c_k)$ when $c_j = c_k$, otherwise the value is 0. The expected economic gain is 0.704 at this point.

4 Model improvement

4.1 Data Shuffling

In the real world, sometimes given data has some certain order, and this can cause bias since the training data set and testing data set are not evenly mixed, so training dataset and testing dataset might both contain some special feature, that causes the training process to be less efficient and decision rule less accurate. Under this circumstance, shuffling data can help to evenly distribute dataset and increase the possibility of training the data set including as much information as possible. In my experience, even the dataset in my model was generated randomly, after shuffling, and repeating the training process, it still shows an improvement of the model: expected economic gain increased from original 0.704 to 0.708.

I used the DataFram.sample(frac=1) function to return a random sample of items from an axis of object to shuffle the entire data set.

4.2 Cross Validation

Cross validation allows us to use all the data for training at least once, the model has 5 folders, and chooses one at each time as the testing set and rest of them as the training set. The result shows when using the first folder as testing

set and rest of the data as training set, the model performs the best, which increase the expected gain to 0.746.(figure 1)

```
[[0.0015, 0.147, 0.111], [0.004, 0.19, 0.1785], [0.0035, 0.183, 0.1815]]
in fold 0 ecnomic gain is: 0.746
[[0.0, 0.0755, 0.1795], [0.0, 0.131, 0.2615], [0.0, 0.1145, 0.238]]
in fold 1 ecnomic gain is: 0.738
[[0.0, 0.114, 0.138], [0.0, 0.182, 0.1845], [0.0, 0.2025, 0.179]]
in fold 2 ecnomic gain is: 0.722
[[0.006, 0.114, 0.1545], [0.0055, 0.162, 0.1845], [0.01, 0.163, 0.2005]]
in fold 3 ecnomic gain is: 0.7370000000000001
[[0.0055, 0.126, 0.1285], [0.008, 0.1625, 0.1975], [0.0075, 0.17, 0.1945]]
in fold 4 ecnomic gain is: 0.7250000000000001
```

Figure 1: Results from cross validation

This model has a 5-fold cross validation, and the number of the fold can be defined by users based on needs.

The shuffling and cross validation are both improving the model's performance by making adjustment within data set itself.

4.3 Optimization

The other optimization I did for the model is modifying the class conditional probabilities. I pick a small iteration Δ, which equals 0.015 in my case, and check the true class tag and assigned class tag. For each wrong assignment, I increase the conditional probability for the d given the true class, for example, the true class for di is 2, but it's assigned class is 3, the model will increase the $Pr(di \mid C_2)$ by a small iteration d. After adjusting all wrong assignments, the model normalizes the sum of conditional probabilities for each class to 1 again, and then performs the decision rule again with the new conditional probabilities.

Here are the mathematical definitions of the above procedure: let c_j represent the true class, and c_k represents the assigned class. For each wrong assignment of di, where $cj! = ck$, add iteration to conditional probabilities $Pr(d_i|c_j)' = Pr(d_i|c_j) + \Delta$. Then normalized the new conditional probabilities so that for each $c \in C$, $\sum_P r(d_i|c)' = 1$. And this is done by following equation: for $c \in C$, and for each d:

$$Pr(d_i|C)'' = \frac{Pr(d_i|c)'}{\sum_D Pr(d_i|c)'} \tag{4}$$

This step increases the conditional probabilities in the right class for all wrong assigned measurement data, and the identification accuracy is increased at the same time. I also used DataFrame.loc() function to locate the conditional probabilities in each class. Here shows how expected gain changes with the different numbers of iteration added. (Figure 2).

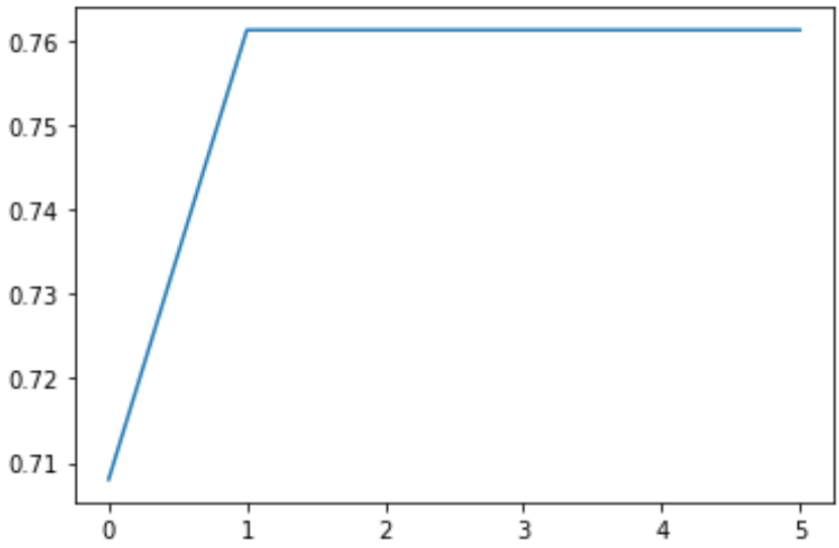

Figure 2: Relationship between expected gain and numbers of delta be added

5 Conclusion

Following the Bayes rule and learning from the data set, one can build a model for the data analysis and help to predicted unknown. there are several effects that affect the performance of the model, for example the data quality, data size, the distribution of the data, and the technique that used during the discover process. The Data shuffling, cross validation and modifying the conditional probabilities are all helpful for improving the model's performance, however this encasement of the improvement will slow down and stop when the performance reaches a certain point. I believe this limitation is caused by the information that carried in the data set itself. In the future I would love to explore more of how Bayes rule works in regression.