Quantization for Discrete Bayes Decision Rule Classification

Daniel Ladner

May 27, 2018

Abstract

Discrete Bayes decision rules can be used to classify data in order to maximize expected value. Although this method is effective when applied to discrete data, applying Bayes decision rules to real valued data makes finding the probability of any specific observation challenging. To overcome this challenge, the dataset can be quantized by creating intervals for each feature and associating each datapoint with its corresponding interval. Each feature can be given a number of intervals based on its proportion of entropy out of all of the features. These techniques were applied to a two class five dimensional dataset of size ten million. In order to find optimal boundaries for the intervals, a brute force, greedy algorithm was used. Using an identity matrix for the economic gain matrix, an unbiased estimate for true positive classification rate of 96.6 percent was found.

Contents

1 Introduction **3**
- 1.1 Bayes Rule . . . 3
- 1.2 Expected Value/Decision Rules . . . 3

2 Problem Formulation **3**

3 Experimental Protocol **4**
- 3.1 Determining Number of Bins and Boundaries . . . 4
- 3.2 Applying Discrete Bayes Rule . . . 5
- 3.3 Optimization . . . 6
- 3.4 K Smoothing Probability Estimates . . . 7

4 Results and Data **7**
- 4.1 Number of Boundaries . . . 7
- 4.2 Deciding on Batch Training set size . . . 7
- 4.3 Boundary Optimization . . . 8

5 Discussion and data analysis **9**

6 Conclusions **10**

7 References **10**

8 Appendix **10**

8.1 User Documentation 10

1 Introduction

1.1 Bayes Rule

Over time, the natural world changes and unfolds in a systematic way. The way that nature unfolds can be categorized into types of events. As these events are observed, information about the probability of these events is revealed. Given a sequence of events, the probability of a specific event can be estimated by taking the sum of the events and dividing it by the total number of events. However, some events either affect or are affected by other events. Given knowledge of these other events, Bayes Rule has given us the ability to update probabilities. The probability of an event c given observation d is given by [1]:

$$P(c|d) = \frac{P(cd)}{P(d)} = \frac{P(d|c)p(c)}{P(d)} \tag{1}$$

By using this principle, and estimating the $P(d|c)$ for all mutually exclusive events $c \in C$ that are dependent on all mutually exclusive events $d \in D$, the probabilities of c can be predicted using only information in d.

1.2 Expected Value/Decision Rules

Knowing the only probabilities tells us which events c are more or less likely. However, this alone does not tell us how to change our behavior. For this reason, humans can assign an economic gain value to each behaving as if event c_j (a member of set C) is true when event c_k is true. These economic gains can be represented as a matrix E where economic gain e in column j and row k represents the quantitative consequence of behaving as if c_j is true when c_k is true.
Using these economic gain values, decision rules can be determined that would maximize expected gain [1]. These rules are called Bayes' decision rules. If K is the number events in C and Ejk is the expected value of choosing c_j when c_k occurs, then a decision rule for $d \in D$ is found by the finding the event c_j that maximizes this formula[1]:

$$e = \sum_{j=1}^{K} E_{jk} P(c_j|d) \tag{2}$$

A confusion matrix of the same shape as the gain matrix can be made where the probability of assigning c_j to c_k is located in the j column and k row. By taking the dot product of these two matrices, a weighted average of the expected values can be found called the expected gain [1].

2 Problem Formulation

In Section 1 we discussed how to compute optimal decision rules given conditional probabilities. Discrete datasets allow us to find probabilities for each combination of data points. However, the probabilities of real dataset observations can not be calculated, since the probability of any particular observation approaches zero. Our dataset is 5 dimensional and real valued between 0 and 1. In order to apply Bayes' Rule, we will employ a discretization and optimization method outlined in Haralick's quantization presentation [2].

Real data is often discretized by creating boundaries for each feature in the observation vector and replacing each feature observation with a discrete number. For example, our dataset has features valued between 0 and 1. By setting a boundary at .5, all values below .5 in the dataset can be replaced with the discrete number 0 and all values above .5 can be replaced by discrete number 1. This assumes that two quantization levels is sufficient and observations that are close to each other provide similar information about the data.

Although this method works, the question of where the boundaries should be placed, how many boundaries there should be, and how to deal with low probability observations is a challenging problem that this project attempts to address. If a generalized method for finding optimal boundaries can be found, then this method could be applied to countless other datasets to find useful underlying structure.

Knowing optimal boundaries of a dataset can also reduce a huge amount of the noise in real datasets. The discretization of the data can be thought of as a form of lossy compression. It takes real values of high amounts of variation and groups together all values that are close together into discrete values of low variation. Once a useful discretization is made, huge datasets can be reduced dramatically by recording only the discretization. Although massive amounts of information will be lost, if the boundaries are optimal, we will retain the information that is necessary for the purposes that we optimize them for. All datasets are technically discretized, because a number can not be stored with infinite precision. So, discretizing data further should not invoke too much worry over loss of information. It is only necessary to retain the amount of information necessary for our purposes.

3 Experimental Protocol

This section will describe the steps that were taken to maximize expected gain for our dataset. First, an entropy calculation was made to decide how many boundaries should be given to each dimension of the data. Second, with the number of boundaries fixed, the boundaries for each dimension were equally spaced out to provide an equal probability quantization of the data. These bounds were used to quantize the dataset. With the data quantized, conditional probabilities of each discrete observation of the dataset was estimated with both Maximum Likelihood Estimates and K smoothing estimates. With those probabilities, Bayes decision rules were calculated. Finally, the boundaries were optimized through brute force. The data was divided into three equal sets. the first set was used for training, the second set was used for optimization, and the third set was used for gaining a final unbiased estimate of the expected gain.

3.1 Determining Number of Bins and Boundaries

For each combination of discretized observations, a probability needs to be estimated and a decision rule need to be calculated. We can think of this combination as a hyper dimensional cube in the measurement space in which real valued observations fall. We call this a bin. For this project, the maximum number of bins was limited to $M = 10,000$, because the dataset has 10,000,000 observations. Using too many bins may overfit, contain too many 0 probability measurements, and take up excessive amounts of computer memory (memory complexity here is exponential with respect to bins per dimension). The entropy of each dimension was estimated by discretizing the dimension into 10,000 equally spaced bins (an equal interval quantization) and using the standard Shannon entropy formula[2]:

$$entropy_F = \sum_{f \in F} -p(f)log_2(p(f)) \tag{3}$$

Where f is a feature observation of dimension F. Next we find the fraction of entropy that a dimension has out of the total of all the dimensions. By raising M to each of these fractions we get a number of bins that multiplies with the other numbers of bins to get close to the desired M. If one dimension has much more information than the others, this formula allocates more bins to it [2].

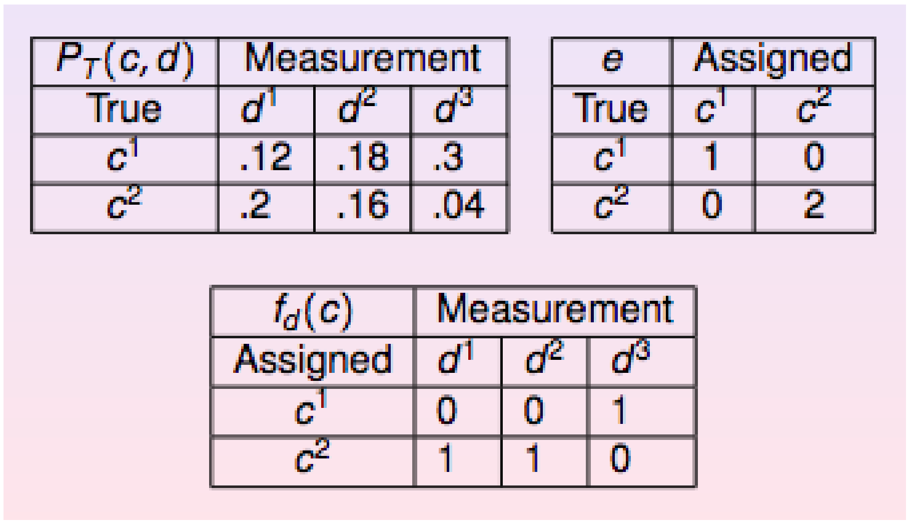
3.2 Applying Discrete Bayes Rule

The boundaries were initialized to create an equal probability quantization and used to quantize the data. A two dimensional array was used to store the boundaries. Each sub array represents a dimension and holds real numbers for each boundary. To quantize a number of the f^{th} feature the f^{th} subarray was binary searched for the boundary closest to and less than the number to quantize. The index of this boundary became the quantized number.

Iterating through the data, the counts of each bin were stored in a custom Python multi-dimensional array that maps each discrete, multi-dimensional coordinate to a linear array address. These counts were divided by the total class conditional observations to get a maximum likelihood estimate for $p(d|c)$ where d is the bin and c is the class. To get p(d), the counts of d given c were summed for all $c \in C$ for each d and divided by the total number of d with class c. To get the $p(c|d)$, the prior probabilities were inputted and formula (1) was used. A decision rule for each $d \in D$ was computed that maximizes equation (2).

To estimate the expected gain of the decision rules, first a confusion matrix was calculated by iterating through the testing set of data and counting the probabilities that each bin is assigned to each class C when using the Bayes' decision rules. Finally a dot product is taken with the gain matrix to find the expected gain.

To test the discrete bayes classifier, the probabilities of a simple example was taken from professor Haralick's slides [2]. Here are the input probabilities, economic gains, and resulting decision rules of the example:



$P_T(c,d)$	Measurement		
True	d^1	d^2	d^3
c^1	.12	.18	.3
c^2	.2	.16	.04

e	Assigned	
True	c^1	c^2
c^1	1	0
c^2	0	2

$f_d(c)$	Measurement		
Assigned	d^1	d^2	d^3
c^1	0	0	1
c^2	1	1	0

Figure 1: The top left table shows the joint probabilities of each datapoint and class observation. The top right table shows the economic gain matrix to be used. The bottom table shows the resulting decision rules. This example was taken from professor Haralick's lecture slides [2]

By plugging in the above probabilities, the exactly the same set of decision rules, confusion matrix, and expected gain was computed. The computed expected gain was 1.02 which matched the above example. In addition, the optimal boundaries that were used to generate the data were used to quantize the data. The resulting probabilities were used with the classifier and achieved over 99.99% accuracy.

3.3 Optimization

The final step is to optimize the boundaries. In this phase, a random dimension was chosen. Then a random boundary in that dimension was chosen. Finally, a random point between the boundary below and above it was chosen. The boundary was changed to this point, and the data was quantized with the new boundaries. With the new quantized values, new probabilities, decision rules, and expected gains were calculated as outlined in section 3.2. The boundary was kept only if the new boundaries produced a higher expected value. This process was repeated many times to find optimal boundaries.

Using all of the training and testing set at the same time proved to be very time consuming (1 minute per evaluation). So, the training set was divided into smaller batches. Every ten evaluations, the training batch would be switched with the next batch. The testing set was broken up into smaller groups as well. This led to much faster evaluations of a boundary set (about 3 seconds). The full training and validation datasets were used after the boundaries were optimized to get a final unbiased estimate of the performance.

Two optimizations were performed. In the first optimization, only the smaller training batch was used during each evaluation of the bounds. Once a high expected gain was achieved, the best bounds found in the previous step were used to start and the full training set was used to estimate the probabilities of the bins for optimization.

Several random seeds were used, but the best result came from using the random seed: 58298.

3.4 K Smoothing Probability Estimates

An alternative to MLE was attempted using the technique outlined in Haralick's quantization slides[2]. For each bin, the N closest bins were taken whose counts added up to more than parameter $k*$. $k*$ was chosen to be the square root of the observations. The counts of these N bins were divided by the volume of the N bin and multiplied by the volume of the chosen bin to estimate the counts in that bin. Afterward, the new counts were summed up, and each bin was divided by this sum to get the probabilities of each bin. This approach was used to smooth out the probabilities of the bins and estimate probabilities of bins that were too small for any observations to fall into them. To limit the complexity of this operation, only the bins that were one manhattan distance away from the chosen bin were considered regardless of whether or not the $k*$ count requirement was met.

4 Results and Data

4.1 Number of Boundaries

The entropy for each dimension was estimated equation (3) by using 10,000 equally spaced quantizing boundaries for each dimension. They all had the same entropy of around 12.4 bits per feature observation. Since the entropies were equal, the method outlined in 3.1 suggests 6.3 boundaries per dimension. Considering this, 7 boundaries per dimension was chosen.

4.2 Deciding on Batch Training set size

The training set was broken into batches of 50,000. 50,000 was chosen as a good compromise between speed and probability approximation after observing that the classification rate improved as the number of training observations N increased, but they did improve not by much after 50,000 observations. Below shows a graph of expected value vs N given equally spaced bounds:

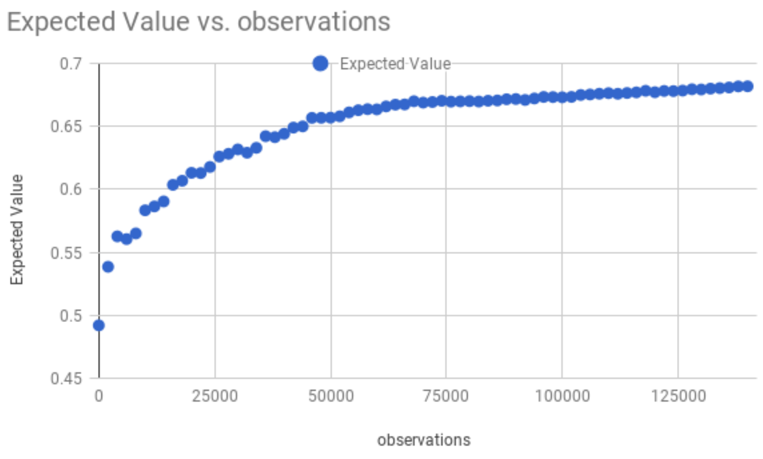


Figure 2: This figure shows how adding more observations for training discrete bayes has diminishing returns.

Including the whole training dataset improved the expected gain by only a few percent beyond the 50,000 observation estimate.

4.3 Boundary Optimization

After several optimizations and varying different parameters, these were the best boundaries that were found:

boundaries	1	2	3	4	5	6	7	8
dimension 1	0	0.022	0.182	0.569	0.844	0.906	0.987	1
dimension 2	0	0.471	0.472	0.492	0.537	0.591	0.620	1
dimension 3	0	0.084	0.153	0.195	0.237	0.605	0.939	1
dimension 4	0	0.028	0.254	0.323	0.827	0.851	0.853	1
dimension 5	0	0.062	0.101	0.194	0.511	0.518	0.602	1

Figure 3: Best boundaries from optimization algorithm

boundaries	1	2	3	4	5	6	7
dimension 1	0	0.020	0.181	0.564	0.907	0.987	1
dimension 2	0	0.474	0.492	0.536	0.589	0.620	1
dimension 3	0	0.083	0.152	0.242	0.607	0.945	1
dimension 4	0	0.027	0.226	0.255	0.325	0.853	1
dimension 5	0	0.063	0.104	0.192	0.514	0.604	1

Figure 4: Optimal boundaries used to generate the data.

The optimization took a total of 11.51 hours to terminate. These results were compared with the optimal boundaries used to generate the data shown in figure 4.

Below is the confusion matrix for the bounds shown in figure 3.

	True 0	True 1
Assigned 0	0.4824	0.0184
Assigned 1	0.0149	0.4842

Figure 5: The confusion matrix obtained when using the bounds in figure 3

The following table shows various parameters and resulting expected gain:

k	expected gain	accuracy
0	1.886847879	0.9666524242
21	1.779435152	0.9118224242
42	1.753380606	0.8989133333
223	1.627299697	0.8387206061

Figure 6: Optimization results

In figure (3), the Kstar parameter represents the parameter used for approximating smoothed probabilities as outlined in section 3.4. Using the smoothed probabilities reduced the expected gain of the optimized boundaries. However, it was noticed that smoothing the probabilities did improve expected gain when boundaries were more spread out. K smoothing may only be effective on datasets that are not concentrated so heavily in such tight boundaries. Note: the K smoothing algorithm used here only considered bins that were one manhattan distance away. The d parameter represents the smallest unit that the optimizer could move a boundary. Reducing this parameter increased the expected gain.

5 Discussion and data analysis

The data shows that a greedy brute force hill climbing algorithm can be used to optimize quantization levels for real valued datasets. Interestingly, the brute force algorithm often pushed some boundaries very close together while spacing other boundaries far apart. This indicates that the important information may be concentrated in some areas more than others. In addition, some boundaries were pushed very close together. Most likely, a lower d parameter will lead to a higher expected gain, because the boundaries to fluctuate more precisely.

Comparing the boundaries used to generate the data and the boundaries found through optimization, we can see that the boundaries found through optimization were very close to the boundaries used to generate the data. For every bound in figure 4, there is a corresponding boundary that is very close in figure 3. The boundaries in figure 3 that don't correspond to the ones in figure 4 are between two boundaries that do. This is because 7 intervals were used instead of 6 when finding the boundaries through optimization.

Unexpectedly, the algorithm for smoothing probabilities decreased as $k*$ increased. It may be the case that only using bins that are one manhattan distance away was affecting the algorithm. However, the optimal boundaries contained many boundaries with 0 counts, because they were too small. In this case, smoothing the probabilities might simply redistribute too much probability from high density areas to near 0 density areas. It would be interesting to attempt to use a more refined K smoothing algorithms by using a k-d tree to find nearest neighbors.

Limiting the training set to size 50,000 led to drastically increased performance, as the algorithm could iterate much more in the same amount of time without sacrificing too much accuracy. Adding more data to the Discrete Bayes' rule followed a logarithmic model. This is expected behavior, because the probabilities should slowly converge on the true probabilities.

6 Conclusions

This report demonstrates the effectiveness of discrete Bayes' rules on real valued datasets. Although, this report did not achieve over 99% accuracy, it seems possible to increase the accuracy by tweaking the d parameter and adjusting the smoothing algorithm. It is interesting to see that real valued data can be grouped together in artificial box-like structures. Perhaps a kind of curved boundary would be interesting for further research.

The git repository is located here: https://github.com/djlad/discrete-bayes-machine-learning-algorithm.git

7 References

[1] Haralick, R. (2018). Discrete Bayes [Power Point slides]. Retrieved from http://haralick.org/ML/discrete_bayes.pdf

[2] Haralick, R. (2018). Quantization [Power Point slides]. Retrieved from http://haralick.org/ML/quantization.pdf

8 Appendix

8.1 User Documentation

The code for this project consists of 3 main files for 3 main functions. discrete bayes.py, quantizer.py, and optimizer.py. quantizer.py handles all of the conversions from real values to quantized values. The discrete bayes.py file takes quantized probability values and computes the discrete bayes rule. The optimizer.py file uses both the quantizer and discretebayes modules to find and evaluate optimized bin boundaries.

csv2sqlite.py simply converts the pr data csv file to an sqlite database by changing the file name parameter in this file and running the file with python it will generate the sqlite database from the specified file.

The tests directory contains test files with various unit tests used to verify the correctness of the program. They can be run by using the python unittest modules command line functionality.

To run an optimization, an input file must be created using the various parameters for the optimization. A text file must be created in the inputs directory. This text file is written as a python dictionary with the following attributes:

- seed: an int representing the seed to use for the random number generator.
- rounds: number of iterations to run optimization for. (put 0 to just evaluate the bounds)
- d: float use to determine the distance between points that can be selected when randomly perturbing the bounds.
- nrows: number of rows to use in a batch when optimizing the boundaries.
- offset: number of rows to skip after training set to before starting the evaluation set
- trainoffset: number of rows to skip in the data before starting training set.
- best_gain: best expected gain to start off with. (default is 0)
- kstar: parameter kstar to use for K smoothing algorithm
- start_bounds: 2 dimensional array representing the boundaries. (see example inputs for example)
- gain matrix: 2 dimensional array representing gain matrix.

To run the optimization, run python optimizer.py file_name where file_name is the name of the file in the inputs folder that contains the parameters for the optimization. The optimizer will periodically save its progress in the outputs directory under a file with the same name as the input file except for a '-output' suffix.