Pattern Recognition with Discrete Bayes

Enis Berk Çoban

Abstract—Bayes theorem is widely used in machine learning for probabilistic classification. In this paper we explore using Discrete Bayes' rule for classifying continuous data with binary output. Since our data is continuous and our classifier is discrete, we quantize data. Quantization process start with calculating information carried by each feature, and number of quantization levels for each feature is determined accordingly. Discrete Bayes used to calculate decision rule and optimization applied to quantization process until it decision rule's expected gain on test data cannot be improved. Data with 10 million points and 5 features is used. Accuracy is 96.7 percent and expected gain is 1.93 out of 1.99.

Keywords—Discrete Bayes Rule, probabilistic classification

1 Introduction

B-AYES theorem enables us to calculate probability of a cause given effects. Statistically dependent variables provide us information for predicting the others. This can be formulated for two variables as following which is conditional probability of c given D:

$$P(c|\mathbf{D}) = \frac{P(c,\mathbf{D})}{P(\mathbf{D})} \tag{1}$$

$P(D)$ can be calculated by summing joint probabilities P(D,c) with all possible values of c variable. This is known as Law of Total Probability and written as:

$$P_d(\mathbf{D}) = \sum_{c \in C} P(\mathbf{D}, c) \tag{2}$$

With this law, conditional probability can be written as following which is called Bayes' rule

$$P(c|\mathbf{D}) = \frac{P(c)P(\mathbf{D}|c)}{\sum_{c \in C} P(\mathbf{D}|c)P_c(C)} \tag{3}$$

Denominator of the Bayes' rule is a normalization factor, it makes sure that posterior probability sums to 1. With this rule we can transition from $P(c|D)$ to $P(D|c)$, this is helpful because an effect might be result of many cause but effects of a cause are known in general. In classification context, c represents a class and d represents a data point to be classified. Since prior probabilities might given by an expert or can be calculated with $P(D|c)$ from training data. Then we can use $P(c|D)$ to classify a given data point. [1]

Supervised classification's purpose is assigning labels or categories to examples defined by related variables or features. Bayes' rule enables development of explicit and interpretable decision rules for supervised classification. Decision rule is learned from data-set which is labeled with expert help or labels known in data collection process. Also discrete Bayes Classifier outputs probabilistic results, allowing users to use this metric as a confidence measure.

In classification, a data point d from measurement space D, can be classified correctly as c^j or with a wrong class as c^k, $c \in C$. For each class, true or false classification of data points carries an economical consequence. To represent all possible economical consequences, a matrix with dimensions of $|C| * |C|$ can be used. Each row and column representing classes in the same order, rows being correct class for a data point and columns are assigned classes. Each cell of the matrix stores expected economical gain related with corresponding assigned and correct classes $e(c^j, c^k)$. This matrix is called gain matrix and filled with an expert or user guide from the field of the specific task. Another matrix similar to gain matrix is confusion matrix, rows and columns represents classes however each cell stores probability of a machine learning model assigning a class to data points in a class. So sum of diagonal entries of the confusion matrix, gives us probability of true classification. [2]

Discrete bayes' rule can be used with continuous data sets as well, however data points will be mostly unique and learning from them will not be possible. To solve this problem we quantize each feature range to quantised intervals. This quantization will divide data space to regions and each data point in the data-set will be represented by corresponding region in discrete bayes' rule application. For this reason, regions should not be random but covering similar data points. We optimize quantization boundaries to maximize expected gain, eventually creating regions covering related data points. Depending on quantization levels of features, number of regions increases exponentially. When there are too many regions, regions with few or no data points increases. In that case over-fitting might occur. As a solution we apply smoothing to number of elements in those regions to increase models' generalization ability.

2 QUANTIZATION

Classifying real valued or sparse data-sets with Discrete Bayes is not suitable. From equation 3, we know that, conditional probability of data point given class $P(D|c)$ needs to be known to be able to calculate conditional probability of class given data point. This means, when number of times a data point exists in training set increases, we learn more about this data point's closeness to different classes. With real valued data set,correctly approximating this probability will not be possible since most of the data points will exists once or few times in training set. Even worse, most of the data points to be classified will be unseen. Such data's dimensions can be quantized to create a denser data set, representing each data point with its corresponding region. Quantization used transforming analog signals to digital in engineering for further processing. It is as assignment of an integer value (between 0 and $K-1$) to a continuous physical quantity. [7] Q quantization resulting with level k can be written as following:

$$Q_k = \{x|q(x) = k\} \tag{4}$$

When each J dimensions (features) quantized to L quantization levels, it generates J^L bins in discrete measurement space. If $Z = (z_1, z_2 ... z_j)$ is a data point tuple and q_j is the quantization function for the jth dimension, quantised Z will be represented as $(q_1(z_1), q_2(z_2)...q_j(z_j))$. Since quantization function maps data point to a region in a space, we will store those regions in a memory table, each tuple will be memory address of corresponding region. So $Q(Z)$ will be address for data point Z in the memory. With this mapping if we revise bayes' rule equation 3, now we replace data point D with it's quantized version:

$$P(c|Q(\mathbf{D})) = \frac{P(c)P(Q((\mathbf{D}))|c)}{\sum_{c \in C} P(Q(\mathbf{D})|c)P_c(C)} \tag{5}$$

Deciding number of quantization levels K, is first step of quantization. We should spare more bins to features with higher amount of information. If data points are spread over range of the feature, there would be more information to differentiate them from one another. Because information gain increases with number of low probability events' occurrence. Measurement of information is called entropy, and it is formulated as following:

$$\hat{H} = -\sum_{x=0}^{N-1} p_x log_2(p_x) \tag{6}$$

If $x \in \{0, ...x_{N-1}\}$, p_x is probability of x occurring in a sample from a data-set which is constructed from set of x values. H bits (entropy) is average information per symbol in the sample. To calculate entropy in a feature, we divide it's range to T number of bins. Each bin is as a symbol in entropy equation(eq:6) and each data point falling into that bin is interpreted as occurrence of corresponding symbol in the sample. If sample list is $(z_1, z_2 ... z_N)$, then first component of the each data point is $(z_{11}, z_{21} ... z_{N1})$. When they are mapped by quantization, we get the list $(Q_1(z_{11}), Q_1(z_{21}) ... Q_1(z_{N1}))$ where each value is $\in \{0...K-1\}$. Then x values in entropy function are $\#\{n|Q_1(z_{1n}) = k\}$ and $p_x = \frac{x}{N}$.

Statistical variation of small samples brings both statistical and systematic deviations of entropy estimates. There are publications on decreasing estimation error by adding terms of corrections [5]. We are going to use the term added by Miller, allowing us to include empty bins to entropy estimation. [4] Given that $n_0 = \#\{k|x_k = 0\}$, entropy formula becomes:

$$\hat{H} = -\sum_{x=0}^{N-1} p_x log_2(p_x) + \frac{n_0 - 1}{2Nlog_e 2} \quad (7)$$

With close approximation of entropy we can compare each feature by amount of information they carry and allocate number of bins accordingly to each one. Number of bins (L) is bounded by amount of memory space (M) and we calculate it for j'th feature L_j as following:

$$f_j = \frac{\hat{H}_j}{\sum_{i=1}^{J} \hat{H}_i}$$

$$L_j = \lceil M^{f_j} \rceil$$

After calculating number of bins to be used in quantization, we initialize their borders with values that makes probability of each bin p_x equal. To achieve this effect we sort each feature list $(z_{1j}, z_{2j}...z_{Nj})$ in ascending order, and elements in the following indexes becomes left quantizing interval boundaries:

$$(0, \frac{N}{L_j} + 1, ..., \frac{kN}{L_j} + 1, ..., \frac{(L_j - 1)N}{L_j} + 1)$$

However classes in the data not necessarily spread in regions that are spaced equally. For this reason we will optimize boundaries to increase expected gain. [3]

We are going to use letter b to represent a boundary,as an example following list is set of boundaries belongs to j'th feature for K quantization levels:

$$(b_{1j}, b_{2j}...b_{kj}...b_{Kj})$$

3 Optimization and Smoothing

When we quantize J number of features (dimensions) to L_j levels,we get $M = \prod_{i=1}^{J} L_j$ number of regions (memory cells) which possibly a data point can fall into. When we train our classifier with quantized data, all data-points falling into the same region will be assumed to have same value. When data-points falling into the same region have different classes, region is called to be non-uniform. If a region is not uniform then classifier cannot create a good decision rule for this region. We optimize boundaries of each bin to make regions more uniform, resulting a better decision rule and higher expected gain. In a epoch we optimize all boundaries once in random order. In optimization process, we perturb a boundary of a bin in a pre-determined range which is not exceeding previous and next boundary. Perturbation amount (δ) should be chosen small enough to find boundaries that have high precision. For a boundary $(b_{i,j})$, total number of values that are tested in one epoch is equal to distance between neighbour boundaries divided by perturbation amount:

$$\frac{b_{i+1,j} - b_{i-1,j}}{\delta}$$

To test new value of a boundary, data is quantized with new boundaries, decision rule re-calculated and new expected gain is recorded. Then new boundary becomes the one with the highest expected gain. This process is continued until there is no more improvements on the expected gain. Details of optimization implementation explained in dataset and experiments section.

Making no assumptions on formal data and making strong assumptions for scarce data points brakes generalization capability of ML model. Smoothing allows us to connect those two edge cases and create ML model with a better generalization behaviour [6]. There are M number of regions, each one of them have a volume V_m and number of data points in the region is k_m. Density of a region is mass divided by volume, so density is $\frac{k/M}{V}$. Lets assume optimal k is $k*$. We smooth each region that have less number of elements than $k*$. First we find indexes $(m_1..., m_I)$ of closest I neighbours of the region that satisfies following equations:

$$\sum_{i=1}^{I} k_{m_i} \geq k^*$$

$$\sum_{i=1}^{I-1} k_{m_i} < k^*$$

Let the sum of the number of elements in the region and it's neighbours $b_m = \sum_{i=1}^{I} k_{m_i}$ and sum of the volumes to be $V_m^* = \sum_{i=1}^{I} v_{m_i}$. Then density becomes $\alpha b_m / V_m^*$, we set α in a way that density integrates to 1.

$$\text{Probability of each bin:} p_m = (\alpha b_m / V_m^*) v_m$$

$$\text{total probability:} 1 = \sum_{m=1}^{M} p_m = \alpha b_m v_m / V_m^*$$

$$\alpha = \frac{1}{\sum_{m=1}^{M} b_m v_m / V_m^*}$$

$$p_m = \frac{1}{\sum_{m=1}^{M} b_k v_k / V_k^*} b_m v_m / V_m^*$$

$$\text{if } v_m = v, m = 1, ..., M, \text{then } V_m^* = I_m v$$

$$p_m = \frac{b_m v / I_m V}{\sum_{m=1}^{M} b_k v / I_k v}$$

$$= \frac{b_m / I_m}{\sum_{m=1}^{M} b_k / I_k}$$

After finding closest neighbours accumulating total elements equal or more than $k*$, we use probability formula we derived above to calculate smoothed values. [3]

4 Dataset and Experiments

Dataset I use in our experiments have 1 million data points and 5 features, each one of them had range of 0 to 1. Precision of numbers is 5 after the decimal point. For each data point we have a binary class label 0 or 1. I divided data-set to three equal parts, training, development and test set. Economic gain matrix provided with data-set is:

$$\begin{bmatrix} 1 & -1 \\ -2 & 3 \end{bmatrix}$$

Also class conditional probabilities are given with data-set and they are $P(0) = 0.4$ and $P(1) = 0.6$. Previously I explained following procedures sorted in order:

1) Calculation of required number of bins
2) Initial boundary calculations
3) Quantisaiton of data
4) Training Bayesian Model
5) Calculating gain and accuracy scores of model
6) Boundary Optimisation

In the first procedure, I calculate number of bins required for each feature with entropy information. Second step is calculation of boundary values for fixed-sized bins. Then I quantize data with initial boundaries. In the training bayesian model procedure, I compute conditional probability of a data point given class $P(d|c)$ from quantized data points of training set and I compute decision rule from $P(d|c)$. Finally, I use decision rule to predict class of data points with the development set and calculate gain and accuracy scores of model. To increase expected gain, I optimize bins' boundaries for each feature. I follow those procedures in given order. However optimization procedure (6) involves 3th,4th and 5th steps. When I chance value of a boundary, I go through those steps to calculate new results. One epoch involves executing boundary optimization for all boundaries in random order. In a epoch, for a boundary we test values that are between in between value of it's neighbours with precision of δ. Optimization continues until there are no improvements in the expected gain after an epoch.

This optimization procedure takes around 4 days with computer that have a 2.6 GHz Intel Core i5 processor and 8 GB of Ram. This time is quite long when we consider fact that it takes many trials to design, code and debug a ML model. Also makes it hard to re-produce the results. To solve this problem, I introduced three solutions. First one is sampling data to create a smaller data set for optimizing boundaries with higher speed. Since big enough sample will have similar boundaries with total data, we start optimization with small sample and increase the size of the sample until I use all of it. With trial and error I found that 0.001 percent of the data is big enough to start optimizing boundaries. I follow optimization procedure I explained previously until there are no improvements for each sample and then use optimized boundaries as starting boundaries for the next data sample. I increase sampling rate by power of 10, and I use 0.001,0.01,0.1 percent of data and eventually all of the data in that order.

Second solution to increase speed of training is using Hill Climbing algorithm to find optimal boundaries. Hill Climbing algorithm is a optimization technique for local search. It starts in a random point, and calculates gain of neighbour solutions and picks the one with highest gain to continue the process. If a solution have higher gain than it's neighbours then it stops and returns it as the maximum value. Since it cannot guarantee finding global maximum, it is ran for many times to increase probability of finding best solution. When I check the gain in the process of optimizing a boundary, I realized that for that particular boundary, solution space was convex. However since I cannot guarantee that would be the case all the time, I use hill climbing and original optimization technique together. I start with hill climbing optimization and when there is no more improvement in the gain, I run original optimization one time. And then I continue with the hill climbing. If original optimization do not improve gain after hill climbing, I stop optimization.

Sampling increases optimization speed by reverse of sampling rate. And hill climbing on average 10 times faster than original optimization algorithm. Even if those speeds up optimization, there is another cause of slow conversion to best possible boundaries. And this cause is local minimums generated by set of boundaries being shifted from their best possible positions. Lets assume we have set of boundaries for a feature and following values are best for highest gain:

$$(b_{1j}, b_{2j}, b_{3j}, b_{4j}, b_{5j})$$

If optimizer finds boundaries b_{3j}, b_{4j}, b_{5j} but places them in correct order to in a wrong places such as:

$$(b_{3j}, b_{4j}, b_{5j}, X, X)$$

Since they are correct boundaries and they are next to each other, quantizer have a high expected gain score. Optimizer also moves other two boundaries shown with Xs to left or right side as much as it can to eliminate their effect. As a result creates a quantizer with 3 boundaries and optimization gets stuck to this local minimum. And it takes so many iterations even with original optimization algorithm to get rid of this local minimum. To detect this problem, in my optimization procedure after original optimization step I check for boundaries that are equal in the precision of δ. If two boundaries are same in the precision of δ, it means that one of them is a pushed boundary as shown with an X in my example. I accept that as a sign of local minimum and I initialize boundaries of this quantizer with random values. After initialization, I use my combined optimization method to optimize only this quantizer's values. I continue to this process until expected gain is higher than expected gain of the local minimum found in the first place. This re-initialization prevents optimization to get stuck at local minimums.

Total running time of the program was 4.2 hours and there was 52 epochs.

For testing, I generated a simple input file with simple data, and tested all functions with that file. I used examples from lectures to create the data.

I used development set for tuning parameters such as smoothing. Also development set is used for optimization of boundaries. All reported results are generated with test data. Training, development and test data are all one third of the total data.

After optimization completed, we move to smoothing probabilities. Equation for smoothed probabilities is:

$$\frac{b_m/I_m}{\sum_{m=1}^{M} b_k/I_k} \qquad (8)$$

Hyper-parameter in this equation is $k*$. Optimal number of elements in each region(memory cell) can be found with following equation:

$$k* = \frac{jN}{20M}, j \in 1, 2, 3, 4 \tag{9}$$

We apply smoothing for all values of $k*$ and pick the one with highest expected gain.

5 Results

Before smoothing accuracy is 96.7 percent, and expected gain is 1.9315 on the test set. If accuracy was 100 percent, expected gain would be 1.9983 on this data set. Following table shows the confusion matrix:

0.4907	0.0091
0.0101	0.4900

TABLE 1
Confusion Matrix

After smoothing,best accuracy is 97.9, and expected gain is 1.92 on test set. While accuracy increases with smoothing, expected gain decreases.

k	j	Accuracy	Expected Gain
0	0	96.7%	**1.9315**
20	1	**97.9%**	1.9241
40	2	97.6%	1.9146
60	3	97.3%	1.9041
80	4	97.0%	1.8961

TABLE 2
Accuracy and Expected Gain after smoothing with different j values

Following table shows quantization borders for each feature:

Table 3 shows boundaries determining bins and generated with optimization processes. And table 4 provides boundaries used to generated data we use for this experiment. So they are the best boundaries one can achieve.

0	0.02	0.18	0.56	0.79	0.91	1
0	0.47	0.4918	0.5374	0.5896	0.6196	1
0	0.0860	0.1518	0.2418	0.6060	0.9460	1
0	0.03	0.2262	0.2553	0.3244	0.8544	1
0	0.0635	0.1035	0.1912	0.5112	0.6035	1

TABLE 3
Optimized Boundary Values

0	0.0203	0.1811	0.5637	0.9067	0.9871	1
0	0.4743	0.4915	0.5360	0.5894	0.6202	1
0	0.08321	0.1519	0.2415	0.6065	0.9449	1
0	0.026	0.2257	0.2549	0.3245	0.8531	1
0	0.06348	0.1035	0.1916	0.5135	0.6037	1

TABLE 4
Boundary values used to generate the data

When two tables are compared, one can see that all boundaries except last two boundary in the first row are same with the two decimal precision. We consider two decimal precision because our δ was 0.01. However, to achieve identical results one can use smaller δ value to continue optimizing.

I achieved those results in 4.2 hours and there was 52 epochs. Computer had 2.6 GHz Intel Core i5 processor and 8 GB of Ram.

6 Conclusion

In this paper, we showed that discrete bayes can be used classify a continuous dataset with correct level of quantization and optimization. Improved suggested optimization strategy to increase speed and save time with hill climbing method. Best expected gain we achieved is 1.93 over 1.98 with 96.7 percent accuracy.

Appendix

User Documentation

Code is written in Python 3.6. To run the code, following python 3.x libraries are required:

- Pandas 0.23
- Numpy 1.14

There is a main.py file,one needs to run it only to reproduce the results. ”Pr_data.txt” is required to be in the same folder with exact name. When it is run, it generates a csv file with information about each epoch. Each epoch generates 10 lines in csv file, first 5 is boundaries, 6ht one is sampling rate, 7th one is best possible gain for current sampled data,8th one is gain, 9th is accuracy and last one is elapsed time in that epoch.

Normally, code calculates bin_counts itself. However a small sample from data set is used for optimization so in that case bin counts can be smaller as well. We want to keep bin_counts fixed according to total data so they are hard coded to be 6.

All higher parameters are in the main file, such as class_probabilities and gain matrix. One can change them accordingly in there.

Code is commented, and aim of each code file is explained in the ReadMe file.

Partner Project

My partner is Shweta Garg who has done her project with Panda Subhadarshi. Their code is minimalist and they are taking advantage of numpy library more than I did. For example, I did not know there was a function called digitize which is a quantization function for given boundaries. Probably, my code would run faster if I used it.

They are using original optimization strategy suggested in the project so their code takes more than 3 days to complete. Our results are almost same, I have a slightly higher expected gain with 1% difference. Also with smoothing, they also found that expected gain is decreasing.

References

[1] Richard O. Duda, Peter E. Hart, David G. Stork, C R. O. Duda, P. E. Hart, and D. G. Stork. Pattern classification, 2nd ed, 2001.

[2] Robert M. Haralick. Discrete bayes. University Lecture, 2018.

[3] Robert M. Haralick. Quantization. University Lecture, 2018.

[4] GA Miller. Note on the bias of information estimates. information theory in psychology: Problems and methods, 1955.

[5] Thomas Schrmann. Bias analysis in entropy estimation. *Journal of Physics A: Mathematical and General*, 37(27):L295, 2004.

[6] Jeffrey S. Simonoff. *Introduction*, pages 1–12. Springer New York, New York, NY, 1996.

[7] Bernard Widrow. A study of rough amplitude quantization by means of nyquist sampling theory. *IRE Transactions on Circuit Theory*, 3(4):266–276, 1956.