Discrete Bayes Classifier*

*Midterm Report

Bi Wu
1^{st} year Master In Data Science
Graduate Center, CUNY
bwu1@gradcenter.cuny.edu

***Abstract*—This paper shows the construction, performance testing and optimization of a discrete naive Bayes classifier that combines the discrete Bayes probability model with a decision rule. The main methods applied in the construction involve generating class conditionals and saving them in a linear space, as well as designing a decision rule based on an economic gain matrix through a user's input. The methods applied in the performance testing of the classifier involve synthetic data generation, K-fold cross validation and the optimization of the classifier is done through modification of the class conditionals based on the previous testing results.**

***Index Terms*—machine learning model, probability model, naive bayes model, discrete naive bayes classifier**

I. Introduction

In this paper, a discrete Naive Bayes classifier is constructed to test on a synthetic dataset of k classes and I dimensions. Each dimension of the tuple can take on T_i values. (T_i can be different for each i in I)

Each possible combination of T_i of I is assigned to a unique linear address a. The classifier then saves a in a space called M. The length of M is : $\prod_{i=1}^{I} T_i$.

The classifier then generates k conditional probabilities for each address in the space M - the conditional probability for measurement d for class c is $P(d \mid c)$. The classifier also generates a prior probabilities $P(c)$ for each c in C. The number of $P(c)$ is k. Having $P(d \mid c)$ and $P(c)$, the classifier computes the posterior probabilities $P(c\ midd)$ for eahc class c of a tuple d.

The classifier takes a user input of an economic gain matrix that decides the economic gain E_{AT} of assigning a class c_A when the true class is c_T.

The classifier then constructs a decision rule that decides on an assigned class c_A when c_A produces the maximum economic gain.

Using cumulative probability distribution, a synthetic dataset is generated to test the performance of the discrete Bayes classifier. The synthetic dataset is split into v folds for cross-validation. In each validation, one fold is taken out from the v folds for testing and the rest $v-1$ folds is used for training. In each round of validation, classification accuracy and total economic gains from the training set and the testing set are reported.

When generating the synthetic data, no relationship between the measurement tuples and class tags is established, thus the classifier performs poorly on this dataset. In order to optimize the performance of the classifier on this dataset, a parameter $delta$ is introduced to the testing process to modify the class conditional probabilities for any wrong assignment, so that the conditional probability $P(d\ midc)$ of d is increased when the true class is c. Eventually, the classifier is able to increase assignment accuracy to close to 1 with enough iteration of $delta$.

II. Assumptions

- Independence among features: no pair of features are dependent. The value of one feature has nothing to do with the value of another feature
- Equal contribution to the outcome: each feature is given the same weight, thus each feature is assumed to be contributing equally to the outcome of predicting a class.

III. Technical

A. The Naive Bayes model

The mathematical equation of Bayes Theorem is:

$$P(c \mid d) = \frac{P(d \mid c) * P(c)}{P(d)} \tag{1}$$

where, c is class variable and d is a dependent feature vector (of size n) where:$d = (d1, d2, d3, ..., dn)$.

Because independence among features is assumed in a Naive Bayes model, we can rewrite the formula by splitting the evidence d into independent parts:

$$P(c \mid (d1, d2, d3, ..., dn)) = \frac{P(d1 \mid c)P(d2P(c)...P(dn \mid c)P(c)}{P(d1)P(d2)...P(dn)} \tag{2}$$

As the denominator $P(d1)P(d2)...P(dn)$ remains constant for a given input, we can remove the denominator, so that we have:

$$P(c \mid (d1, d2, d3, ..., dn)) \propto P(d1 \mid c)P(d2P(c)...P(dn \mid c)P(c) \tag{3}$$

Given a sample vector d, the classifier should calculate the probabilities of assigning the measurement tuple d to each class c - $P(c \mid d)$. The classifier does so by multiplying the class conditional probability of sample tuple d given class c - $P(d \mid c)$, and the prior probability of class c - $P(c)$.

B. Measurement Space

Given a dataset of k classes and I features, each feature can take on T_i values (T_i can be different for each i in I). Each possible combination of T_i for I is saved in a space called called M, the length of M is : $\prod_{i=1}^{I} T_i$. Each unique tuple in the space is then assigned to a unique linear address a. a is an integer. Each unique tuple in the measurement space and its corresponding linear address is saved in an address book so that it can be accessed later.

C. Priors

The number of classes is k.
The set of all classes is C, $C = c_0, c_1, ..., c_k$.
Generate an array of size k with uniform [0,1] pseudo random numbers.
Normalize the sum to create prior probabilities $P(c)$ for each c in C.

D. Class conditionals

The last address in the measurement space array is S.
The length of the measurement space array is $S + 1$.
For each c in C, generate an array of size $S + 1$ with uniform [0,1] pseudo random numbers. Normalize the sum to create conditional probabilities $< q_0, q_1, ..., q_S$. Copy $< q_0, q_1, ..., q_S$ to the address space as $P(d \mid c)$.
The resulting class conditionals are stored in a dictionary where each address from the space is stored as a key and its corresponding array of size k class conditionals is stored as the value.

E. Economic gain matrix

The economic gain matrix is a 2 dimensional square matrix of k rows and k columns. The rows correspond with assigned classes and the columns correspond with true classes. For each c_A when the true class is c_T, the economic gain matrix gives an economic gain of E_{AT}.

IV. Training and testing on synthetic data

The classifier uses a synthetic data to train and test the performance of the classifier.

- The dataset should have k classes.
- The dataset should have I features ($I >= 5$).
- Each feature i in I can have T_i different values.
- The dataset should have at least N rows.$N >= 10 * \prod_{i=1}^{I} T_i$.

A. Generate synthetic tuples

The dataset should have I features, feature i should have T_i possible values.
For each feature i in I, repeat the following process to generate a one-dimensional sample if size N:

- Generate an array of size T_i with uniform [0,1] pseudo random numbers.
- Normalize the sum to create probabilities $< q_0, q_1, ..., q_{Ti}$.
- Form cumulative distribution $< p_0, p_1, ..., p_{Ti} >$ following the below process:
  For $i \in 0, ..., T_i$:
  generate $p_i = \sum_{j=0}^{i} q_j$.
- Each value in the sample will be in the set $T = 0, ...T_i$.
- For $i \in 1, ..., N$:
  Generate x a uniform $[0, 1]$ pseudo random number.
  If $x <= p_0$, generate a sample value $v_i = 0$.
  If for any $j \in JT_i$, $p_j < x <= p_j + 1$ generate a sample value of $v_i = j + 1$.

B. Generate class tags

An array of size N class tags should be generated independently from the measurement tuples. The class tags should have no relationship with the measurement tuples.
To generate the class tags, do the following:

- Generate an array of size k cumulative distribution $< p_1, p_2, ..., p_k >$
- Generate the class assignment function $h : M \to C$:
  For $x \in M$:
  Generate a uniform $[0, 1]$ number x.
  If $x < p1$, set $h(i) = 1$.
  If for any $k \in Ck, p_k <= x < p_k + 1$, set $h(i) = k + 1$

C. Shuffle the data

After generating the synthetic dataset, which contains I-dimensional measurement tuples of size N and an array of class tags of size N, a random permutation $s_1, ...s_n$ is used to permute the sequence of measurement tuples and the sequence of class tags. Doing so will ensure that each class tagged tuple has the same probability of being positioned at each position of the sequence.

D. K-fold cross validation

Partition the shuffled dataset (including the measurement tuples and the class tags) into K folds (K has nothing to do with the number of classes k), each fold should have approximately the same size.

1) Round Robin: For each round of the cross-validation, take a different fold of the data to be the testing set, and the rest $K - 1$ folds to be the training set.
Using the classifier, for each measurement tuple d in the training set, calculate the posterior probabilities $P(c \mid d)$ for each class c, then compute the economic gains E_d for each assigned class c_A, the classifier should decide on the assigned class with the highest economic gain.
The expected gain and accuracy for the training and testing sets are reported for each round of the validation.

2) Optimization: Set parameter $\delta < 0.05$. For each tuple d in the measurement space M, use the classifier to assign a class. For each d, compare the assigned class with the true class tag, if the assigned class c_A is not the true class c_T, then

modify the class conditional probability $P(d \mid c_T)$ by adding δ.
Re-normalize the class conditionals so that:
For each $c \in C, \sum_{d \in M} P(d \mid c) = 1$.

3) Repeat K-fold cross validation: Use the modified class conditionals to repeat the K-fold validation process and report expected gains and accuracy on both training and testing sets.

V. Conclusions

A. Setting parameters for the data

In the experiment, we set

- number of classes $k = 3$
- index(name) set of the classes is $C = [0, 1, 2]$
- number of features = 5
- the value set of each feature is: $[[0, 1], [0, 1, 2], [0, 1, 2, 3], [0, 1], [0, 1, 2]]$
- economic gain matrix is the identity matrix which looks like:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- number of rows for the synthetic dataset is 100000.

B. Expected gains and accuracy from cross validation

In K-fold cross validation, K is set to 5.
The dataset has 100000 rows, thus in each round of the validation, the training set has 80000 rows and the testing set has 20000 rows.
In the initial testing of the performance of the classifier, the training set reports expected gains around 326 and the testing set reports expected gains around 82. Both the training and testing set reports accuracy at around $16\% - 17\%$.

C. Expected gains and accuracy vs. δ iterations

After the initial 5-fold cross validation, use the classifier again to assign a class for each measurement tuple in the dataset without splitting the dataset into training and testing set. Compare the assigned classes with the true classes (initially generated class tags), for any measurement tuple whose assigned class is not equal to the true class, modify its class conditional probability of the true class ($P(d \mid c_T)$ by adding $\delta = .02$.
The resulting expected gain and accuracy from the entire dataset is seen to increase with the increase of number of iterations of δ modification.
However, the accuracy reaches its ceiling at around 25% no matter the number of δ iterations added afterwards. (As seen in the graph below).

References

[1] Robert M. Haralick, *Discrete Bayes Pattern Recognition*
http://haralick.org/ML/midterm_project.pdf

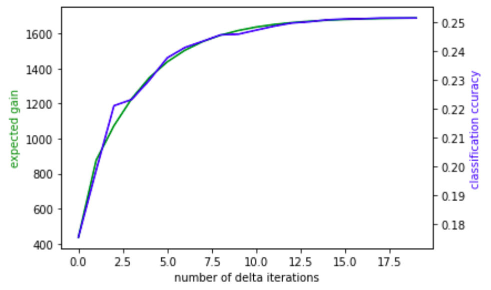

Fig. 1. Expected gains and accuracy vs. δ iterations

Appendix

Code
The code is available here: Discrete Bayes Classifier