Journal of Machine Learning Research 1 (2020) 1-48 Submitted 4/00; Published 10/00

Learning with Naïve Bayes

Seth Bassetti SETH.BASSETTI@STUDENT.MONTANA.EDU
School of Computing
Montana State University
Bozeman, MT, USA

Ben Holmgren BENJAMIN.HOLMGREN1@STUDENT.MONTANA.EDU
School of Computing
Montana State University
Bozeman, MT, USA

Wes Robbins WESLEY.ROBBINS@STUDENT.MONTANA.EDU
School of Computing
Montana State University
Bozeman, MT, USA

Editor: Seth Bassett, Ben Holmgren, and Wes Robbins

Abstract

This paper describes our group's findings when using a Naïve Bayesian approach for learning on various sets of data. We provide our findings on these data sets with and without noise in the data. More specifically, we show how the model performs on each data set using a percent accuracy statistic and two different loss functions. Prior to the project, we were expecting to find that generally the model would perform best on data with more feature variables, and also that the model would perform best on less noisy data. Though our hypotheses were not fully confirmed to the degree of statistical significance we had initially expected, we did find that these trends were present throughout the project, with exceptions for specific cases.

Keywords: Naïve Bayes

1. Problem Statement

Utilizing five datasets- each from unique and differing settings, we implemented a Naïve Bayesian learning model in an attempt to provide insights into the correlation of features within each set of data. More rigorously, each data set came with explicit classifications, and we sought to train our model in order to then guess the classification of a given set of feature data in the absence of a provided classification. Each of the five datasets we use in the project have a variable number of classes and either discrete or real valued attribute values. We then provided an analysis of each dataset with permutations on its feature values. We hypothesized that in general, more attributes would mean a better performance of our model, since more attribute data could be interpreted as gaining greater specificity in pinpointing revealing characteristics of a class. We presumed that this enhanced performance would be statistically significant, in that at least 90 percent of trials would show greater reliability of learning on datasets with more attribute data when compared to datasets with

fewer attributes. In this context, the reliability of learning then was measurable by the percent with which the model correctly guessed a class given only feature data, and also via two loss functions of our choosing. Additionally, we hypothesized that any given dataset would perform more reliably 90 percent of the time when compared to its corresponding permuted data, under the same measurements for reliability as for the first hypothesis.

2. Methods

In order to test our hypotheses, we ran our model on the five original given datasets and introduced noise into each dataset, comparing the results that our learning model was able to attain when comparing the original data to data with permutations. Initially, the datasets we used to test our model included breast cancer screening data, data referring to different properties of various kinds of glass, data on the properties of three different iris plant species, data referring to the properties of different kinds of soybeans, and the 1984 voting data of each congressman in the U.S. house of representatives on the sixteen key votes identified in the Congressional Quarterly Almanac. All of these data were real valued, with the exception of the data corresponding to the House of representatives, which was in the form of strings indicating a vote "y" for yes and "n" for no. We converted these strings into real valued entries by assigning all yes votes to the value 1, and all no votes to the value 0. We also needed to handle missing attribute values in our data, which were existent in two of our datasets. In the congressional voting dataset, missing values corresponded with a "present" vote, or a refusal to take sides. For this, we assigned all missing values 0, which corresponds to a "no", since voting present most likely would indicate that the voter is against the proposal. The breast cancer dataset also had missing values which actually corresponded to data which was unavailable. Because the dataset was already real valued on a range from 1 to 10, we decided to give all missing entries a value of 5 which corresponds to the midpoint of the range. We rationalized that in a medical setting, this should indicate that a queried cancerous bump shouldn't be negligently overlooked, but missing data also need not indicate severe cancer, so we decided that a middle value in this context was appropriate.

For each of these data sets, noise was introduced by permuting approximately 10% of the feature data in any given set. More precisely, for each and every feature value, there was a randomized 10% chance that permutation would occur. Permutation was conducted by computing the midpoint of values for every feature, and then "flipping" every selected data point over this midpoint by subtracting from the midpoint the distance between the midpoint and the data point if the data point were initially larger than the midpoint, and otherwise adding to the midpoint the distance between the midpoint and the data point if the point were initially smaller than the midpoint.

Mathematically, letting F_i be a feature in a dataset and $f_j \in F_i$ be an individual data point of F_i, a midpoint m was computed by $m = (max(f \in F_i) + min(f \in F_i))/2$, and our permutations f_j^* were then conducted by computing on the randomly selected f_j composing approximately 10% of F_i:

$$\begin{cases} f_j^* = m - (f_j - m) \text{ if } f_j > m \\ f_j^* = m + (m - f_j) \text{ if } f_j < m \end{cases}$$

Having accomplished the permutations, another step of preprocessing this data was discretizing real-valued data into categorical data. To do this, we used "binning", which is the processing of dividing values into a number of bins in order to convert it into a finite number of categories. Initially, we set the number of bins to 2 for every dataset before we would go back and fine tune hyperparameters once the algorithm was fully implemented.

After fully preprocessing our data, we were able to begin implementation of the Naïve Bayesian algorithm. After doing so, we needed a metric to determine how well our algorithm performs on each dataset. For this task, we introduced two loss functions to measure performance, a 0/1 loss function, and a log based loss function. These loss functions provide an effective method of determining model performance when certain features of the model are changed. Explicitly, letting c_i be the level of certainty with which a guess was made in our model for the ith line of test data, logarithmic loss for each data point was computed as follows:

$$\begin{cases} ln(c_i) \text{ if correct} \\ ln(1 - c_i) \text{ if incorrect} \end{cases}$$

Letting the value y_i be the value produced by the above equation for the ith line of test data, the logarithmic loss for the entire test set T is computed as:

$$\frac{-1}{|T|} \sum_{t \in T} y_i \tag{1}$$

The second loss function we introduced was the 0/1 loss function, which is the ratio of incorrect guesses to total guesses made on the testing data. Again denoting the test set T, and letting the number of correct guesses made on the test set by the model be denoted g_c, the 0/1 loss is computed with the simple ratio:

$$1 - \frac{g_c}{|T|} \tag{2}$$

Once our algorithm and loss functions had been fully implemented, we began tuning hyperparameters to optimize model performance. Ten fold cross validation proved to be a useful method of training and testing our data. This involved dividing the dataset into ten groups, with nine being used for training and one being used for testing, and rotating so that every group is eventually used for testing. The hyperparameter we were most interested in was the number of bins we used to discretize the real valued datasets. Using cross-validation, we determined that 4 bins offered the best performance for our model, and thus divided the real values into 4 bins to discretize it for features with more than four independent values assigned to any given entry. This was due in part to our findings, that it didn't seem to alter the performance of our model once the bin size was increased above 4, and also due to some limitations in our implementation. Creating more bins seems to bog down our model, and introduces a host of potential problem areas in our implementation that don't seem to be worth minimal performance improvements in the best case, and in practice no real noticeable difference. However, improving our ability to fine tune binning is a clear area for future work for subsequent implementations. Yet, for this project, altering the number of bins seemed by observation to be around optimal with 4 bins for every feature in each dataset excluding the congressional voting data, which for practical purposes was

also convenient in that it limited the number of areas where potential vulnerabilities could arise within our implementation.

3. Results

References