Naive Bayesian Classification

Turner Haugen TURNER.HAUGEN712@GMAIL.COM
Machine Learning Student
Montana State University
Bozeman, MT 59718, USA

Jackson Thorn JTHORN2448@GMAIL.COM
Machine Learning Student
Montana State University
Bozeman, MT 59718, USA

Abstract

This paper aims to provide a custom implementation of the Naive Bayesian Classification algorithm with evaluation on a variety of datasets, provided by the UCI Machine Learning Repository. Our method handles datasets with continuous and discrete variables through equal-width discretization, along with k-fold cross-validation to improve model robustness. We also explore randomly adding noise to 10 percent of the data to further refine the model. Precision, recall and F1 scores are computed from confusion matrices as measures of model performance.

1 Introduction

In an era where machine learning has transformed industries from finance to healthcare, choosing the right model is paramount to any organization's decisions and goals. Though overshadowed by recent advancements in generative artificial intelligence, probabilistic classifiers like Naive Bayes provide value due to their simplicity, efficiency and generally low cost of compute. Traditionally designed to work with categorical data, Naive Bayes can be expanded to datasets with continuous data by transforming features into discrete intervals. Naive Bayes performs exceptionally well with categorical data; some common applications include email spam filtering, text classification and medical diagnosis.

One common issue with Naive Bayes is applying the classifier to continuous data without violating the feature distribution assumptions for binary data. Addressing this challenge consists of partitioning the continuous features into bins of equal-width size, through a process called equal-width discretization. Though effective when working with continuous data, there remains the possibility of information loss. Paired with careful bin-width selection and exploratory data analysis, Naive Bayes can be effective when working with both categorical and continuous datasets.

Because several datasets provided for this experiment contain continuous data, we propose the use of equal-width discretization as a pre-processing step. Additionally, k-fold cross-validation and randomly adding noise to 10 percent of the data further tests our model's

robustness. Our aim is to develop an initial model that implements Naive Bayes on the original datasets, while the addition of k-fold cross-validation and random noise provides insight into our model's ability to generalize. This study details our design and implementation of a Naive Bayesian Classifier, and we hypothesize that our model will perform better than random classification.

2 Design

We approached this problem by testing the general Naive Bayesian process on a single, categorical dataset ('house-votes-84'). More specifically, each major process was first tested in a Jupyter notebook before being expanded to an object-oriented approach. We identified the six major requirements for a simple Naive Bayesian Classifier: data loading, data processing, prior probability computation, conditional probability computation, model prediction and evaluation.

2.1 Data Loading and Processing

Datasets were provided as .data and .names files, where .data files contained features for each class and .names files contained information about each class. Loading .data files consisted of simply reading the 2D arrays into Pandas Dataframes, though loading in the .names files was somewhat trivial. We found that there was no replicable method to load in class header information from the .names files, so those values were hard-coded. Once all data was confirmed to be loaded in the dataframe, it was crucial to shuffle it. We found that some datasets had extreme biases present, especially toward the end. After randomly shuffling rows with a set seed to ensure reproducibility, we decided on a train/test split of 80/20 to ensure that Naive Bayes had plenty of training and testing data, dropping any missing values in the process.

2.2 Prior Probabilities

At this point, we had a dataframe containing categorical information for all instances present in our data. To compute prior probabilities, we followed this formula:

$$Q(C = c_i) = \frac{|\{x \in c_i\}|}{N}$$

Where $Q(C = c_i)$ represents the prior probability for a given class, $|\{x \in c_i\}|$ the count of instances for a given class and N the total number of instances in the dataset. Effectively we computed the length of all instances in the dataset, and the length of each individual class. We obtained prior probabilities by dividing the length of all instances by the length of the given class. This computation represents the proportion of each class in the entire dataset, and is stored in a dictionary for later reference.

2.3 Conditional Probabilities

Conditional probabilities were computed based on the following formula:

$$F(A_j = a_k, C = c_i) = \frac{|\{x_{A_j} = a_k \wedge x \in c_i\}| + 1}{N_{c_i} + d}$$

Where $F(A_j = a_k, C = c_i)$ represents the conditional probability for a specific feature given its attribute on the selected class, $|\{x_{A_j} = a_k \wedge x \in c_i\}| + 1$ represents the count of the number of instances where both the feature A_j has the same value as a_k and the instance belongs in class c_i with the value + 1 added on as Laplacian Smoothing, and $N_{c_i} + d$ is the total number of instances in class c_i added to d, which is the total number of unique feature values A_j can have. Effectively, we extracted features from the training dataset and looped through each feature to find unique values. Next, we looped through each class and unique value and found the class count by computing the length of the count of how often the feature takes the value a_k for the class c_i divided by the total number of instances in class c_i plus the count of the possible values for the feature. This computation represents the likelihood of observing a specific feature value given that the instance belongs to a particular class, also stored in a dictionary for later reference.

2.4 Predictions

To predict whether a given attribute belongs to a given class, we expanded on this formula:

$$C(x) = P(C = c_i) \times \prod_{j=1}^{d} P(A_j = a_k \mid C = c_i)$$

Where $C(x)$ is the classification function for instance x, $P(C = c_i)$ is the prior probability of class c_i, $P(A_j = a_k \mid C = c_i)$ is the conditional probability and $\prod_{j=1}^{d}$ is the product of these values. However, there existed the possibility that the prior and conditional probabilities contained extremely small values, skewing computations. To counter this, we applied log transformations to these probabilities which both avoids numerical underflow and allows us to take the sum of the probabilities rather than the product. This computation is handled recursively for each class, and the final prediction is made by returning the class with the highest value for $C(x)$.

2.5 Evaluation

Once Naive Bayes finished training and made its predictions on unseen data, confusion matrices were created to better understand the model's performance. Specifically, predictions were sorted based on their predicted values and actual values in the testing dataset. For binary data, given a class, if the actual class value matches the predicted class value it is considered a true positive. If the actual class value does not match the predicted class value, it is a false negative. If the actual class value matches the predicted class value for the other given class, it is a true negative and if there is no match, it is a false positive. Using these values, we computed precision, recall and F1 scores to understand our model's performance as follows:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \qquad F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Precision is a measure of accuracy for the positive predictions, providing direct insight on how our model predicts on the unseen testing data. Recall measures the sensitivity of the model's correct predictions. F1 score supplies insight on the whether the model is better at

making accurate positive predictions versus finding all positive instances. Combining these metrics gave us valuable insight on our model's predictive ability across the five datasets we made predictions on.

2.6 Expansion on Continuous Data

In order to work with continuous data, we discretized it using equal-width binning. The formula used for calculating the width of each bin is:

$$BW(a_k) = \frac{max(a_k) - min(a_k)}{n}$$

Where $BW(a_k)$ is the bin width for attribute k, $max(a_k)$ and $min(a_k)$ are the largest and smallest values for attribute k, and n is the number of bins. 10 was picked as the number of bins for discretization, as it reduces most of the variability in the dataset but retains any key differences between substantially different observations. This method works well for data that does not have any outliers, which were dropped during pre-processing.

2.7 K-Fold Cross-Validation

K-fold cross-validation was used to evaluate the results of the data, with K being set to 10. The data is split into K folds by dividing the length of the dataset by K. This will usually result in a float, but the output of this is plugged into a round() command to remove floats. This means some folds will be 1 observation larger than others, but this is inevitable with any data that does not have an observation count divisible by K. Data gets shuffled before being assigned into folds to ensure folds don't always contain the same data. Each fold is used as the testing dataset with the other K - 1 folds used as the training dataset. All accuracy metrics from each fold are averaged to produce one set of accuracy metrics for the entire procedure.

3 Results

Aforementioned, we evaluated our results using precision, recall, and F1 scores computed from confusion matrices. For all datasets with more than 2 outcomes for classification, the average of each of these was taken from each classification outcome. For example, there are 6 different classifications for glass, corresponding to 6 different measures of precision, recall, and F1 scores that were averaged. Cross-validation takes this a step further by retrieving the average of each score from each fold combination.

Listed below is a confusion matrix from a single fold from 10-fold cross-validation, as well as summary statistics from the entire 10-fold set:

Wisconsin breast cancer data without noise

Original	Predicted Benign	Predicted Malignant
Actual Benign	43	3
Actual Malignant	0	24

Wisconsin breast cancer data with noise

Noisy	Predicted Benign	Predicted Malignant
Actual Benign	44	0
Actual Malignant	5	21

The original data had an average precision of 0.99, average recall of 0.97, and average F1 score of 0.98. The noisy data had an average precision of 0.95, average recall of 0.93, and average F1 score of 0.94. The results from this dataset are fairly strong and are much better than random chance suggests (0.50). This dataset features the largest number of observations in the 5 datasets we are using as well as no continuous variables, providing data that is ideal for Naive Bayes.

1984 House of Representatives votes data without noise

Original	Predicted Republican	Predicted Democrat
Actual Republican	18	4
Actual Democrat	1	20

1984 House of Representatives votes data with noise

Noisy	Predicted Republican	Predicted Democrat
Actual Republican	15	1
Actual Democrat	6	21

The original data had an average precision of 0.94, average recall of 0.89, and average F1 score of 0.91. The noisy data had an average precision of 0.78, average recall of 0.85, and average F1 score of 0.81. While no continuous variables are present, this data features less total observations overall. Our model's performance was impacted the most by noise here, with each accuracy measure decreasing by more than 0.1. Even with this increased impact from noise, the model performs better than random chance would suggest (0.5).

Iris data without noise

Original	Predicted Versicolor	Predicted Setosa	Predicted Virginica
Actual Versicolor	6	0	0
Actual Setosa	0	2	0
Actual Virginica	2	0	5

Iris data with noise

Noisy	Predicted Versicolor	Predicted Setosa	Predicted Virginica
Actual Versicolor	3	0	0
Actual Setosa	0	3	1
Actual Virginica	1	0	7

The original data had an average precision of 0.95, average recall of 0.94, and average F1 score of 0.94. The noisy data had an average precision of 0.88, average recall of 0.86, and average F1 score of 0.86. On the Iris dataset, equal-width discretization worked well with handling the continuous data present. The model performed much better here than random chance would suggest (0.33).

Soybean data without noise

Original	Predicted D1	Predicted D2	Predicted D3	Predicted D4
Actual D1	1	0	0	0
Actual D2	0	2	0	0
Actual D3	0	0	1	0
Actual D4	0	0	0	1

Soybean data with noise

Noisy	Predicted D1	Predicted D2	Predicted D3	Predicted D4
Actual D1	1	0	0	0
Actual D2	0	2	0	0
Actual D3	0	0	1	0
Actual D4	0	0	0	1

The original data had an average precision of 1.00, average recall of 1.00, and average F1 score of 1.00. The noisy data had an average precision of 0.93, average recall of 0.94, and average F1 score of 0.93. The massive amount of attributes present gives more information for the algorithm to work with, making the algorithm perfect at predicting the original data. Due to the small amount of observations, each fold consisted of 4 or 5 observations. This lowers the individual loss function measurements whenever an incorrect prediction occurs, which is seen in the noisy data. Our algorithm still performed drastically better than what random chance suggests (0.25).

Glass data without noise

Original	Predicted 1	Predicted 7	Predicted 2	Predicted 6	Predicted 5	Predicted 3
Actual 1	2	0	2	0	0	0
Actual 7	0	2	0	0	1	0
Actual 2	3	1	5	0	1	1
Actual 6	0	0	0	1	0	0
Actual 5	0	0	0	0	0	0
Actual 3	2	0	0	0	0	0

Glass data with noise

Original	Predicted 1	Predicted 7	Predicted 2	Predicted 6	Predicted 5	Predicted 3
Actual 1	2	1	3	0	0	0
Actual 7	0	0	1	0	0	0
Actual 2	1	0	6	0	0	0
Actual 6	0	0	0	0	0	0
Actual 5	0	0	0	0	5	0
Actual 3	1	0	0	0	0	1

The original data had an average precision of 0.57, average recall of 0.46, and average F1 score of 0.65. The noisy data had an average precision of 0.50, average recall of 0.40, and average F1 score of 0.59. The data with the most possible options for classification (6 different kinds of glasses) and completely continuous data is the data that our algorithm performs the worst with. It still performs better than what random chance suggests (0.16), but other models would be preferred for accurately classifying glass.

4 Discussion

Our version of Naive Bayes produced the expected results, as the predictions it made with each dataset were accurate. Naive Bayes seems to perform the best on classification problems with fewer distinct categories, as demonstrated by the much lower performance on the glass dataset but strong performance on the breast cancer dataset. The soybean dataset seems to be an exception due to its very high number of parameters present, providing more information to the algorithm. It seems to be less affected by continuous variables when using equal-width discretization, but this would likely change with datasets that include outliers. Additionally, because of the model's reliance on the independence assumption, it would likely perform poorly on other classification problems, such as image recognition or anything time-series related.

Exploring Naive Bayes provided us with a solid foundation toward understanding machine learning concepts and models that use probability as inputs. Our largest challenge was properly implementing k-fold cross-validation, as there are many different methods that can be used. In conclusion, we are happy with the results of developing our algorithm and excited to learn about models that handle more complex classification tasks.

5 Appendix

Jackson wrote the following:

- The model trainer
- The classifier
- The evaluator
- Data handling and pre-processing
- Train-test split
- Code documentation
- Abstract, intro, and design of this paper
- Rough outline of the video script

Turner wrote the following:

- Making the model usable with all datasets
- Adapting model to be object-oriented
- Cross-validation
- Noise introduction
- Discretization
- Results, discussion, and appendix of this paper
- Detailed video script and recording of the video