Learning with Naive Bayes

Andrew Kirby ANDY-KIRBY@LIVE.COM
Kevin Browder BROWDERKEVIN54@GMAIL.COM
Nathan Stouffer NATHANSTOUFFER1999@GMAIL.COM
Eric Kempf ERICKEMPF123@GMAIL.COM

Abstract

The Naive Bayes classifier is a probabilistic model used for classification problems in machine learning. The ease of implementation of the algorithm and basis in Bayesian Decision Theory facilitates its popularity. This experiment shows that introducing noise by scrambling attributes does not significantly change the performance of the algorithm, demonstrating the surprising leniency of the algorithm.

1. Problem Statement

Given an input file that contains examples (each example consists of a list of attributes and an associated classification), the task is to implement a learning algorithm, Naive Bayes, that is trained to classify examples. Of particular interest are the key factors contributing to the performance of the Naive Bayes classifier.

1.1 Variables

The independent variable is whether the data is scrambled or not. Scrambling is described as follows: First, 10% of the attributes in a given data set are randomly selected. Then, within each attribute, the values are randomly swapped between examples. Now the data set is scrambled. The dependent variable is how well the Naive Bayes algorithm performs.

1.2 Hypothesis

The hypothesis is that scrambling a given data set will not significantly change the performance of the Naive Bayes Algorithm. In essence, scrambling renders 10% of attributes useless. Any pattern that existed before scrambling is no longer discernible within those attributes. However, there remains 90% of the data that persists with the original pattern. Naive Bayes chooses a classification based on the relative probability that a given example is in a class. The order of the relative probabilities should vary little, even when the original pattern is lost in 10% of the attributes, yielding similar performance.

2. Naive Bayes

Naive Bayes is a low cost classifying algorithm that utilizes a probabilistic model based in Bayesian Decision Theory. Given an example $x \in X$ with attributes $a_1, a_2, ..., a_d$ and

belonging to class $c \in C$, the algorithm will classify x by choosing the maximum probability $P(c|a_1, a_2, ..., a_d)$ from all $c \in C$ (Langley et al., 1992).

Calculating this probability becomes more apparent when rewritten using Bayes Theorem:

$$P(c|a_1, a_2, ..., a_d) = \frac{P(a_1, a_2, ..., a_d|c)P(c)}{P(a_1, a_2, ..., a_d)}$$

Because the algorithm chooses a classification $c = argmax_{c \in C} P(c|a_1, a_2, ..., a_d)$, the classification is equivalent to

$$c = argmax_{c \in C} P(a_1, a_2, ..., a_d|c)P(c)$$

Naive Bayes assumes all attributes $a \in A$ are conditionally independent given the class, simplifying the classification decision to

$$c = argmax_{c \in C} P(c) \prod_{i=1}^{d} P(a_i|c)$$

These probabilities are calculated when the algorithm is trained with a given set of examples $X_{test} \subset X$. During training, $P(c)$, is calculated for all classes by

$$P(c) = \frac{|X_c|}{|X_{test}|}$$

where X_c is the set of all examples that belong to the class c. $P(a_i|c)$ is calculated with

$$P(a_i|c) = \frac{|X_{c_{a_i}}| + 1}{|X_c| + d}$$

where $X_{c_{a_i}}$ is the set of all examples in class c which have attribute a_i and d is the number of attributes. This expression for $P(a_i|c)$ includes smoothing techniques (characterized by the "+1" and the "+d") (Langley et al., 1992).

Once the algorithm has been trained, all probabilities $P(c)$ and $P(a_i|c)$ have been calculated and stored for future reference. Classification of a new example will be selected quickly with little further computation.

3. Experimental Design

The implementation of Naive Bayes will be trained and tested using ten-fold cross validation. First, the original data sets will be used for the cross validation, then 10% of their attributes are shuffled (to introduce noise) and used for another cross validation. The performance of the learning algorithm will be evaluated by two loss function metrics, accuracy and mean squared error.

3.1 Pre-processing

All pre-processing is done using the pandas library in Python. The class column is moved before all of the attributes for a consistent output between datasets. Next the examples

are randomly shuffled. A new column is added after the class column and each example is assigned to one of 10 sets that are used in the 10 fold cross validation. If the data is continuous, it is discretized into a chosen number of bins with each bin containing an equal number of examples. Non numerical data is changed to integers for ease of use with the algorithm. The data sets used do not have any missing values, so the preprocessing did not need to handle this (Dua and Graff, 2017).

Set information is included in the first three lines of the processed data. The first line includes the number of classes, number of attributes and the number of examples. The second line includes the number of bins for each attribute. The third line includes the class names so the algorithm can convert back from numerical class names to the original string names. After these lines are generated, they are outputted to the first three lines of the .csv file and the pre-processed data is outputted starting on the fourth line. After this data is outputted, ten percent of the attributes are randomly chosen and the data in each of these attributes is randomly shuffled. This shuffled data is then outputted to a new .csv with the same header as the first file.

3.2 Tuning

The most pertinent parameter to tune is the number of bins used when discretizing an attribute with continuous output. Tuning is done by changing the number of bins and then evaluating the loss functions for the different values.

As seen in Table 1, for 2 bins, the accuracy ranges from 69.11% to 91.92% and the mean squared error ranges from 0.65 to 7.8. This begins the process of testing different numbers of bins to produce the best loss function.

Table 1: Loss Function Metrics for num_bins = 2

Dataset	Accuracy	MSE
glass	69.11%	5
iris	70.67%	7.8
house-votes-84	89.66%	4.3
soybean-small	79%	0.65
wdbc	91.92%	7.4

In Table 2, where there are 5 bins, the accuracy ranges from 79.50% to 94.02% and the mean squared error ranges from 0.73 to 6.1.

Table 2: Loss Function Metrics for num_bins = 5

Dataset	Accuracy	MSE
glass	80.78%	2.9
iris	92.67%	0.73
house-votes-84	90.13%	6.1
soybean-small	79.50%	0.9
wdbc	94.02%	4

With 5 bins, the minimum bound for accuracy increases by 10.49% and the range of the mean squared error slightly decreases. This means that 5 bins performs better than 2 bins.

More testing shows that as the number of bins increases past 5, the performance of Naive Bayes stays consistent. The number of bins where the performance begins to drop is not known.

However, viewing Table 3, it is clear that 1000 bins does not produce a good model for all data sets. The accuracy ranges from 5.13% to 92.67% and the mean squared error ranges from 0.6 to 1191.4.

Table 3: Loss Function Metrics for num_bins = 1000

Dataset	Accuracy	MSE
glass	5.13%	80.67
iris	92.67%	0.87
house-votes-84	89.66%	7.9
soybean-small	83.50%	0.6
wdbc	39.54%	1191.4

Thus, the minimum number of bins required to produce a satisfactory model is 5, which was implemented in the final model.

4. Results

The results support the hypothesis that scrambling a given data set will not significantly change the performance of the Naive Bayes Algorithm. Table 4 displays the results of running ten-fold cross validation on the Naive Bayes algorithm trained on various data sets. The accuracies and mean square errors are the average loss function values across the ten different test cycles during cross validation.

Table 4: Loss Function Metrics

Dataset	Accuracy	MSE
glass	80.78%	2.9
glass-scrambled	78.03%	3.17
iris	92.67%	0.73
iris-scrambled	90.67%	1
house-votes-84	90.13%	6.1
house-votes-84-scrambled	89.90%	4.2
soybean-small	79.50%	0.9
soybean-small-scrambled	79.50%	0.9
wdbc	94.02%	4
wdbc-scrambled	93.85%	3.3

Interestingly, the performance of Naive Bayes on the soybean-small data set did not change with scrambled data. This could mean that the scrambled attributes do not define each classification in the soybean-small data set.

Naive Bayes tends to perform better on datasets that allowed them to have a larger training set. As a probabilistic model, this is expected because the probabilities will become more representative (statistically relevant) as more data points are fed to the model. The *soybean-small* dataset is hard to build a sufficient model for because it is very small. Also note that as the number of attributes increase, the amount of data needed to train the algorithm effectively increases.

Of particular note is the effect of scrambling 10% of the attributes in the datasets. Overall, the loss functions (accuracy and mean squared error) were not affected significantly by the scrambling. The attribute scrambling may be likened to noise in the dataset, whose impact is buffered in the "probabilistic nature" of the algorithm, which allows the asymptotic accuracy of the model to remain unchanged (Langley et al., 1992). In all scrambling cases but one, accuracy decreased a small amount. The mean square error, however, decreased only in the *house-votes-84* and *wdbc* scrambled data sets. In these cases, the model slightly underperformed in terms of accuracy, but its class distribution was closer to the actual despite noise in the dataset, yielding a better mean squared error. This hints that certain attributes may have been misrepresenting specific classes within the data.

5. Summary

For its relative cost and simplicity, Naive Bayes is a powerful model for classification. A surprising level of performance was noted given the small size of the data sets used. Although considered a more primitive algorithm, Naive Bayes may be further enhanced with appropriate smoothing techniques and tuning parameters. Knowing the characteristics of the data sets, including the level of attribute conditional independence and relevance, allows for further optimization.

Drawbacks of Naive Bayes include the costliness of optimizing tuning parameters such as discrete bin size and attribute weights (if implemented). The model must be trained and tested in order to compare these parameters. Additionally, the algorithm is only designed to represent examples with conditionally independent attributes, which requires careful planning when deciding included attributes.

References

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Pat Langley, Wayne Iba, Kevin Thompson, et al. An analysis of bayesian classifiers. In *Association for the Advancment of Artificial Intelligence*, volume 90, pages 223–228, 1992.