Machine Learning With Naive Bayes Algorithm

Jonathan Henke

Karthik Kumar

Abstract

This paper describes the implementation of Naive Bayes Algorithm with five data-set from the UCI Machine Learning repository and introducing noise into the data. Data-sets with continuous variables are binned into discrete variables. To perform the Naive Bayes machine Learning algorithm each data-set split into training data and testing data. The algorithm is tested on the a normal data-set and a data-set with noise in the data. The accuracy level of our algorithm is then measured with 10 fold cross validation by running each type of data-set through the algorithm 5 times comparing the average of both the data-sets.

1. Problem Statement

The purpose of this assignment is to implement the Naive Bayes algorithm and test the affects of introducing noise into 5 real world Data-sets. The results are then validated using 5x2 cross validation and the average accuracy results are then compared.

1.1 Hypothesis

Our hypothesis is that introduction of noise into the data-set will have a larger effect on performance of data-sets with less data points. This is because with less data points available to classify the data it will be more inaccurate when some of the features cannot be relied upon to accurately classify the data.

2. Description Of Experimental Approach

Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training data-set. Naive Bayes Classifier is one of the simple and most effective classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naive Bayes Algorithm are spam filtration, sentimental analysis, and classifying articles.

2.1 Performance Calculation

To measure the performance of our Naive Bayes algorithm, we will be using Macro-averaging of precision and recall. To calculate this a confusion matrix is created that contains the predicted class on one axis and the actual class on the other. The counts for each occurrence are recorded.

Predicted	1	2	3	5	6	7
Actual						
1	29	3	2	0	1	0
2	9	22	3	2	2	0
3	3	5	1	0	0	0
5	0	1	0	5	0	1
6	1	0	0	0	2	2
7	0	1	0	2	1	11

Table 1: Sample confusion matrix from glass data-set

To measure recall and precision, we must identify True Positives (TP), True Negatives (TP), False Negatives (FN), and False Positives (FP). True Positives are measured as when the predicted result matches the actual result (the intersection in Table 1 of Predicted 1 and Actual 1). True Negatives are measured as the amount of times the algorithm correctly predicted it was not a certain class (all counts in Table 1 that are not Predicted 1 or Actual 1). False Negatives are each time that we did not successfully predict the class (the counts in Table 1 Predicted 1 but not Actual 1). False Positives are the amount of times a class was predicted but the actual class was different (the counts in Table 1 of Actual 1 but not Predicted 1). The Macro-averaging of recall and precision are as follows where C is all of the classes, and c is an individual class:

$$P_{macro} = \frac{1}{|C|} \sum_{c} \frac{TP_c}{TP_c + FP_c}$$

$$R_{macro} = \frac{1}{|C|} \sum_{c} \frac{TP_c}{TP_c + FN_c}$$

2.2 Program Design

The Program is split into 4 parts, the Naive Bayes algorithm, data shuffling, cross validation, and tuning.

Naive Test
Given training data as an input: the data is trained and parameterized into probabilities and then tested. The performance from the test is then measured using macro-averaging of precision and recall, these numbers are the output. This the algorithm below trains the data:

$$F(A_j = a_k, C = c_i) = \frac{\#\{(\mathbf{x}_{Aj} = a_k) \wedge (\mathbf{x} \in c_i)\} + 1}{N_{c_i} + d}$$

Where $\mathbf{x}$ is a data point, A_j is an attribute of $\mathbf{x}$, and a_k is a value for that attribute. C is all the available classifications and c_i is an individual class. N_{c_i} is the number of times a given class occurs in the training data and d is the total amount of attributes or features each data point has.

The proportion of each class is counted using the following equation, N is the number of examples in the training data-set:

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

This training portion calculates the probability that each attribute value results in a given class. It also records the proportion of each class in the training set. These results are then stored and the testing is begun.

To do the testing, the probability of a data point being in each class is calculated and the class with the highest probability is chosen. To calculate this probability, the below equation is used:

$$C(\mathbf{x}) = Q(C = c_i) \times \prod_{j=1}^{d} F(A_j = a_k, C = C_i)$$

This simply means that the testing function looks up the probabilities for each attribute value of the data point given the class as well as the probability that any given data point is that class. These probabilities are all multiplied together to get the probability the data point should be classified as that class. The class with the highest probability is chosen and set as the predicted class, this is then compared with the actual class to measure performance

Data Shuffling, here noise is introduced to the training data, at least 10% of the features in the training data are randomly shuffled to introduce noise. This occurs each time a test is done on shuffled data.

Cross Validation is the part of the program where a data point for each of the five data-sets is created from the output of 5X2 cross validation on the data-set without shuffling and one with shuffling.

Tuning, here our hyper parameter for bin size is tuned to the best performance.

2.3 Experimental Design

Our experiment is conducted in 5 steps, data preparation. bin size tuning, data discretization, introducing noise, and then running 5x2 cross validation. It should be noted that only the Glass and Iris data-set underwent the bin size tuning and data discretization as they were the only data-sets with continuous data.

Data Preparation

To prepare the data we had to figure out what to do for missing values in breast-cancer data set and the house-votes data-set. In the breast-cancer data-set there was only 16/699 data points missing values, so we deleted them. For the house-votes data-set we changed them to a third value, a, to indicate absent for the vote.

Bin Tuning

To tune the bin sizes of the data a tuning program was created that iterates through even bin sizes from 2-60 and does 5 tests for each bin size. It then plots the results in a scatter plot using the macro-averaging of precision from each bin size test. We then used the bin size that corresponded with the highest macro-averaging precision value. As seen below on Figure 1 there is little benefit to using a bin size larger than 6 for the Iris data and 20 for the Glass data.

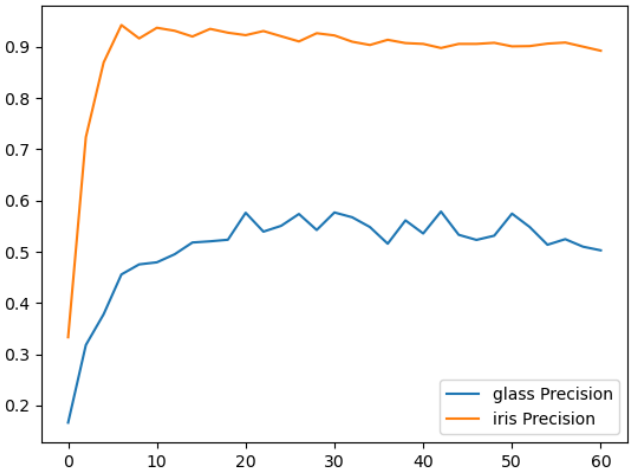

Figure 1: Bin tuning of Glass and Iris data-sets

Data Discretization and Binning

To bin the data we begin by iterating through each attribute and checking to make sure the data is a decimal or "float64" type. The data is then placed into bins using one of two pandas functions: qcut or cut. Qcut puts equal data into each bins, however data with many of the same values cannot be binned this way as the edges of the bins will overlap. When this is the case, cut is used and separates the values into equal sized bins. The difference can be see in Table 2a and Table 2b.

(1.51, 1.517]	54
(1.517, 1.518]	54
(1.519, 1.534]	54
(1.518, 1.519]	52

(a) Refractive Index binned with Qcut

(-0.00315, 0.788]	197
(1.575, 2.362]	8
(0.788, 1.575]	7
(2.362, 3.15]	2

(b) Barium binned using cut

Table 2: Two features from glass.data binned into 4 bins

Bin size 6 for Iris and bin size 20 for Glass were used. To do this we took the unbinned data and ran it through the binning function with the given bin sizes and output the data to a new file.

Noise Introduction

To conduct our experiment we need a version of our data-sets that includes noise. At least 10% of the features are shuffled in the training data to get a shuffled data point. To shuffle, we took the number of features in each data-set and divided it by 10 and rounded to the nearest whole number. We then randomly selected the features to be shuffled and used numpy random shuffle on the column of the chosen feature. This process is repeated 10 times for each data-set as it is done another time when the training and test data are switched. It should be noted that only the training data is ever shuffled.

5X2 Cross Validation

For each data-set a stratified sample of 50% of the data is taken and used as training data, the other 50% is used as test data. Once the performance has been measured, the two samples are swapped and the average from both tests are taken. This process is then repeated 5 times and the result is the average performance of the 10 tests. For comparison we then run the 5X2 cross validation again but shuffling the training data each time to find the performance when noise is introduced to the data.

3. Results

3.1 Tables and Graphs

data-set	Control Precision	Shuffled Precision	Control Recall	Shuffled Recall
glass	0.545970	0.422161	0.504830	0.407773
house-votes-84	0.897066	0.808238	0.888464	0.788838
iris	0.932000	0.840000	0.934956	0.837903
soybean-small	0.745000	0.620000	0.681319	0.661307
breast-cancer	0.974923	0.935212	0.965730	0.937193

Table 3: Performance results of each 5X2 cross validation

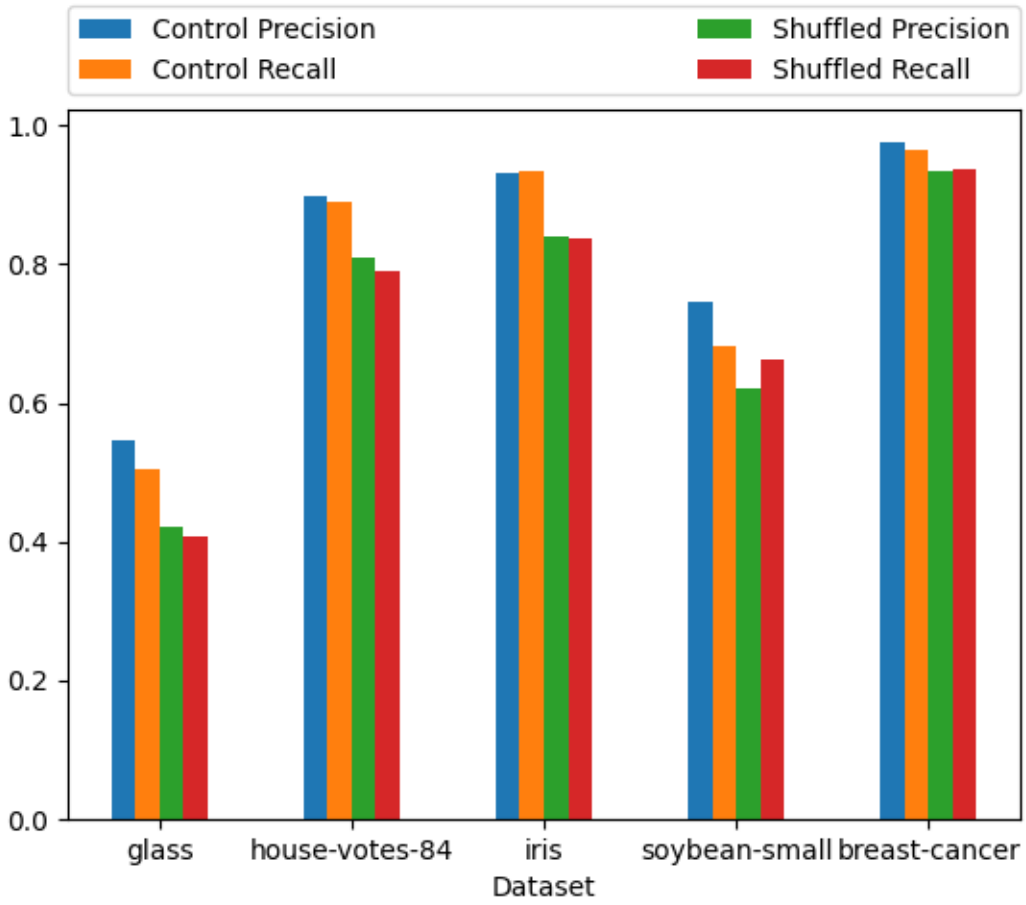

Figure 2: Performance of each data-set compared to performance of shuffling the data

data-set	Features	Data points
glass	9	214
house-votes-84	16	434
iris	4	149
soybean-small	35	46
breast-cancer	9	683

Table 4: Features and data points for each data-set

3.2 Analysis

The final performance results show the precision and recall measurements for both shuffled and control data-sets. Our hypothesis was that the effects of shuffling would be larger in the performance of data-sets with less data points.

In Table 3 and Figure 2, we can see that all data-sets performed worse with the introduction of noise in the training data. The breast cancer data-set was the least affected, it had 699 data points, the largest of the data-sets. This fits with our hypothesis as its training model is more robust due to the larger amounts of data that can be used to create the classifiers.

We can also see that from Figure 2, the performance of the shuffled iris, soybean-small, and glass data-sets are much less than that of the control data-set. These data-sets all have less than half the data points of the house-votes-84 and breast cancer data-sets. This also fits with our hypothesis.

However the house-votes-84 data-sets had a similar decline in performance the reason for this is unknown as it has a large amount of data points and a large amounts of features. It is possible that it is because each feature only has 3 values: yes, no, or absent. where as the breast-cancer data-set has many values for each attribute.

Our hypothesis should be accepted as the data-sets with less data points all had larger declines in performance with the introduction of noise into the data.

4. Summary

In this project we took 5 real world data-sets and implemented the Naive Bayes algorithm to test how introducing noise to the data-set affected performance. We saw larger declines in performance in data-sets with less data points and the data-set with largest amount of values saw the least degradation of performance. We also saw declines in the performance of the house-votes data-set that could not be explained by a lack of data points.