Naive Bayes Classification

Isaac Boyd g55b753@msu.montana.edu
Department of Electrical Engineering
Montana State University
Bozeman, MT 59752, USA

James Lucas f72f751@msu.montana.edu
Department of Computer Science
Montana State University
Bozeman, MT 59752, USA

Abstract

This paper investigates the algorithm known as Naive Bayes. Theory behind the algorithm as well as its implementation on five unique data sets are addressed. The development of the algorithm is discussed and hyper parameter optimizations are made according to loss functions through the use of a grid searching technique. In this work we hypothesize that if the data has additional noise there will be a statistically significant decrease in the scores of the calculated loss functions. Finally, we test our classification model in the presence of noise and find that there is a significant decrease in our loss function scores. The results of these experiments are compared and explained and then used to draw a conclusions.

Keywords: Naive Bayes, Hypothesis, Grid Search

1. Introduction

Naive Bayes classification is accomplished through the use of a learning function, in which we describe a probability distribution and make predictions based on the class similarities between attributes. In this project, steps are taken to implement Naive Bayes on five separate data sets, generate and compare results through loss functions, and compare the efficacy of Naive Bayes in the presence of noise. Our hypothesis is that our Naive Bayes model will have a statistically significant decreased efficacy in the presence of additional noise.

2. Methods

2.1 Data Source

Data was imported from the UCI Machine learning repository. In total 5 data sets were used. These data sets included data on, breast cancer (whether the cancer is malignant or benign), glass type, iris type, disease types in soybeans, and congressional party affiliation. The data itself was organized in such a way so that a designated column gave the classes of each datapoint, while all of the other columns contain attributes.

2.2 Prepossessing

After the data is imported we test if it is discrete. This was a two part process. First we looked at the raw data in order to see if the data was already relatively discrete. From there we defined that the continuous data were floats, and needed to be subjected to binning in

order to make the data discrete. In the end we determined that the glass type dataset and the iris dataset needed to be binned, while the other datasets were discrete enough to use their raw data. The function that we used in order to make the datasets discrete looked through every attribute column and found the maximum and minimum. From here we spit the data in each attribute column according to their distance from evenly spaced bins between the maximum and minimum. This would, in turn, bin the data into a tunable number of bins.

From here the data was shuffled and then split into ten testing sets for ten fold cross validation. In order to implement ten fold cross validation, we used our testing data for each fold and we held out the training data and removed the labels on the testing data. Then we run each piece of testing data through the model in order to generate scores.

2.3 Model

The implementation of the algorithm entailed looping over every attribute on a per class basis and counting the number of identical occurrences. Once the number of identical occurrences within a class if found it is added to "mp". This value contains two parts that are both tunable hyperparameters. M is generally represented as an integer value while p is generally a fraction that represents a probability. This process of selecting m and p values is called smoothing via the m-estimate (Liu and Rao (2001)). Its purpose is to minimize the expectation of a discrepancy measure. Finally we divide by the length of the class and add m as seen in Eq.1.

$$(A_j = a_k, C = C_i) = \frac{\#\{(X_{Aj} = a_k) \wedge (X \in c_i)\} + mp}{N_{ci} + m} \tag{Eq.1}$$

The next part of the equation loops through every class and finds the product of all of the off the calculated scores in the previous step per class. See last part of Eq.3. This value is then multiplied by a scaling which is just the size of the class divided by the size of the total data set as seen in Eq.2.

$$\mathrm{Q}(C = C_i) = \frac{\#\{x \in c_i\}}{N} \tag{Eq.2}$$

$$\mathrm{C}(x) = Q(C = C_i) \times \prod_{j=1}^{d} F(A_j = a_k, C = C_i) \tag{Eq.3}$$

Finally, the maximum of all of the scores out of the datasets is reported and the index of this in our algorithm corresponded to the class that our algorithm would predict. See Eq.4.

$$class(x) = \underset{c_i \in C}{\operatorname{argmax}}\, C(x) \tag{Eq.4}$$

2.4 Experimental Approach and Program Design

Our experimental approach and our program design consists of two major aspects. Our program had to have the ability to first, optimize each dataset to work well with Naive

Bayes through hyper-parameter tuning. Then we needed to compare the results of our optimized algorithm with the results of introducing noise into the data set.

In order to accomplish our hyperparameter tuning we manually did a grid search through a range of possible hyperparmeters. The hyper parameters that we needed to tune were m, p, and the number of bins on the continuous datasets as discussed in our model. We started with a large range of values where each value was separated by an order of magnitude. For example for m we tested the values (1,10,100,1000,10000), for p we tested (0.001,0.01,0.1,1), and for the number of bins when we had continuous data we tested (1,10,100,1000).See fig:1.

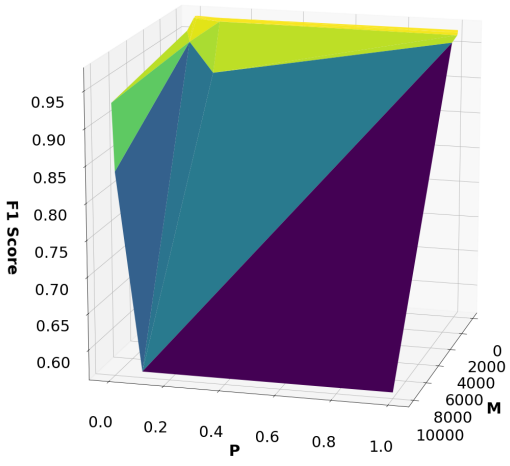

Figure 1: Macroscopic Grid search for hyperparameter tuning.

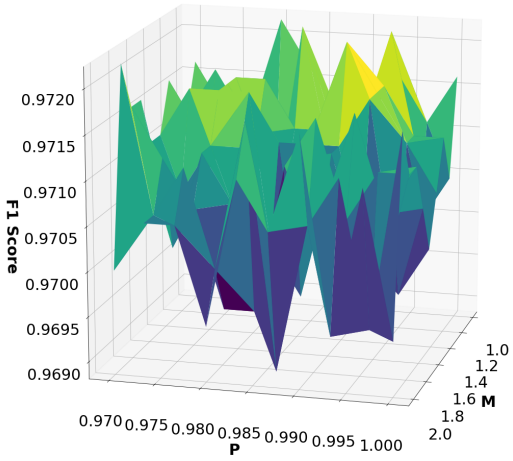

Figure 2: Local Grid search for hyperparameter tuning

After this test ran, we would take the average of thirty trials. From our results we picked the most optimal combination of these three hyperparameters and then once again did another grid search, but this time around the optimal combination from the last trial was used as a starting point. This process of finding the optimal value of the hyperparameters and then manually searching in the local region around the optimal point that we found was repeated until our loss returns diminished and we could no longer improve our model efficacy. See fig:2. For our experiment we decided to optimize according to our F1 score (see Eq.7) since we believe that it was the most representative of our model's efficacy.

$$precision = \frac{true_{positives}}{true_{positives} + false_{positives}} \tag{Eq.5}$$

$$recall = \frac{true_{positives}}{true_{positives} + false_{negatives}} \tag{Eq.6}$$

$$F1_{score} = 2 \times \frac{precision \times recall}{precision + recall} \tag{Eq.7}$$

In order to do this we construct a confusion matrix that tracked the true positives (predictions that are classified correctly), the false positives (predictions that that claim

to be in the correct class but are not), the false negatives (predictions that are in the wrong class when they were supposed to be in the correct class), and true negatives (values appropriately place into the negative class). The output yields a 2D matrix which we can do calculations on.

The second goal of our experiment was accomplished using the results of our hyperparameter tuning. We first tested our algorithm on our regular dataset, then we tested our algorithm on a dataset where we shuffle 10% of the attribute data. We did thirty trials for each dataset. Using the data that we get from these tests we ran a T-test to determine whether or not the drop in our calculated precision value mean and f1 mean were significant between the two models. We used 'r' code in RStudio to generate the t-statistic along with its associated p-value. The t-statistic was calculated using the average difference in means between the observed data and the shuffled data μ_1 - μ_2, the standard deviation s, and the number of samples n, see Eq.8.

$$t = \frac{\mu_1 - \mu_2}{\frac{s}{\sqrt{n}}} \tag{Eq.8}$$

3. Results

The results of our hyper parameter tuning can be seen in *Table 1.* For three out of five of the data sets our ideal m and p values were one. All of our ideal p values were less than or equal to one and all of our ideal m values were either one or two. On the breast cancer dataset and iris datatset the amount of bins that were required to optimize the data varied greatly with a difference of thirty six bins between the two sets. Testing a range of possible hyperparameter values we found that our model started underfitting when the bin size was reduced to two or below, and we overfit our data when we ran with over 200 bins.

Table 1: Hyper Parameter results

	BreastCancer	*Glass*	*Iris*	*Soybean*	*Voting*
m	1.0	2.0	1.0	1.0	2.0
p	1.0	0.2	1.0	1.0	0.004
number of bins	nan	45	9	nan	nan

The results of running the tuned model are shown in *Table 2. Table 2* shows our mean F1 and mean precision scores over the course of thirty trials for the unchanged data. We can determine that our non-shuffled model provided better recall than precision due to the fact that all of our f1 scores were greater than or equal to our precision scores. *Table 3* shows the mean F1 and mean precision parameters over the course of thirty trials for the shuffled data. For all of the datasets the efficacy of our model dropped according to our f1 and precision scores.

Table 2: Tuned dataset Results

	$BreastCancer$	$Glass$	$Iris$	$Soybean$	$Voting$
$\mu : f1$	0.971	0.641	0.958	1.000	0.900
$\mu : Precision$	0.966	0.608	0.958	1.000	0.895

Table 3: Shuffled dataset Results

	$BreastCancer$	$Glass$	$Iris$	$Soybean$	$Voting$
$\mu : f1$	0.918	0.622	0.908	0.953	0.856
$\mu : Precision$	0.921	0.586	0.912	0.962	0.853

Tables 3 and 4 show the results from t-tests that were ran through 'r' code in order to get statistical confirmation. For all of the datasets we see that there is a significant difference between the means when noise is introduced to the data. Our minimum t-value for all measures was 3.908 which in turn yields a probability value of $< (0.001)$ for the datasets having matched means.

Table 4: t-test F1 Results

	$BreastCancer$	$Glass$	$Iris$	$Soybean$	$Voting$
t	43.884	3.9808	22.642	10.884	26.671
p-value	$< .001$	$< .001$	$< .001$	$< .001$	$< .001$
95 percent CI	0.0502, 0.0550	0.0095, 0.0287	0.0453, 0.0541	0.0382, 0.0554	0.0402, 0.0467
Significant	Yes	Yes	Yes	Yes	Yes

Table 5: t-test Precision Results

	$BreastCancer$	$Glass$	$Iris$	$Soybean$	$Voting$
t	36.996	4.807	22.104	10.556	25.644
p-value	$< .001$	$< .001$	$< .001$	$< .001$	$< .001$
95 percent CI	0.0433, 0.0483	0.0123, 0.0299	0.0419, 0.0503	0.0311, 0.0457	0.0383, 0.0448
Significant	Yes	Yes	Yes	Yes	Yes

4. Discussion

In the end our hyperparameter tuning gave some slightly counter-intuitive results. Some of the data preformed very well when hyperparameters were set to unity while some worked better on average on values that were farther away from unity. The most glaring result from these test was that for the Breast Cancer, Iris, and Soybean dataset, the p and m values selected were identical and ideal. Given, that our efficacy was also the highest in these models, we believe that we have found optimal values, however it should be noted that these unity values break the norm that is generally seen in literature.

We also made specific hypotheses pertaining to each individual dataset and how it would perform. The Breast cancer dataset was our largest dataset. Our initial prediction was that it would be more resistance when we added noise to the dataset in our testing. However, we found out that in reality the glass dataset was the most resistant to noise. We also hypothesized that the glass dataset would be the hardest to classify. After running our tests we found this to be the case. The glass dataset had the lowest metrics across the board. The iris dataset we saw was already class balanced. We saw that this dataset was the cleanest of all of the datasets and hypothesized that it would be the easiest to classify. However, this hypothesis was misguided. The iris dataset was very clean, however, we had a hard time determining an optimal hyper parameter for this set. A very large range of m, p, and number of bins gave similar results. As a result, the space that we had to do our grid search on was vast. Narrowing down across all of these hyperparmeters was difficult and resulted in us determining a result that may not quite be ideal. The soybean dataset we hypothesized we would overfit on. The dataset did not contain very many points and we were worried that once we ran the data it would over represent the training data. We did find that it could not adjust perfectly for noise, however, it did not do as bad when the ten percent noise was introduced compared to the other models. Finally the voting data set. For this dataset we hypothesized that we would be able to classify well because there were only two classes. We did obtain good F1 and Precision scores, however, we did not have as good of results as some of the other data sets. We believe that this dataset had some unique challenges due to the three potential attribute values and the context of the problem. Unlike the other datasets the attributes in this dataset were all determined by someone's decision to vote a certain way. This processes is less predictable than many of the other observable events since it is based on human decision making and randomness.

Our tests also produced results that ultimately proved our null hypothesis. Introducing noise into the data set on our optimized models produced a reduced efficacy. This shows that our model is not adequately robust. Speaking technically, our model preforms well on data that is local, however when maximally different data is introduced it has a significant decrease in categorization efficacy. From these results we predict that our model might be slightly overfit to our data. This being said, while we may be been slightly overfit to our datasets, our model's efficacy is much larger than the maximally underfit (randomly guessing) and maximally overfit (cannot adapt to new data at all) cases for each dataset, assuming that for each of these maximal cases the average classification accuracy is one divided by the number of classes.

5. Summary

This paper investigated Naive Bayes, developed experiments where we optimized a model, and drew conclusions from our hypothesis. We found that we could make accuracy classifications with Naive Bayes despite its simplicity. We also determined that our optimal models were not ideal and we suggested how we might improve upon our existing model. In the end we found significant statistical data that supports our alternative hypothesis that the difference between the mean of our data and our shuffled data was significant.

References

Z.J. Liu and C.R. Rao. Mu-estimation and smoothing. *Journal of Multivariate Analysis*, 76 (2):277–293, 2001. ISSN 0047-259X. doi: https://doi.org/10.1006/jmva.2000.1916. URL `https://www.sciencedirect.com/science/article/pii/S0047259X0091916X`.