Journal of Machine Learning Research 24 (2024) 1-10 Submitted 9/13; Published N/A

Investigating Naive Bayes Classifier Performance With Noise Injection

Carlos Rivas CARIVAS007@GMAIL.COM
Department of Computer Engineering
Montana State University
Bozeman, MT 59715, USA

Finnegan Davidson DAVIDSONFINN@GMAIL.COM
Department of Computer Engineering
Montana State University
Bozeman, MT 59715, USA

Editor: Carlos Rivas and Finnegan Davidson

Abstract

Machine Learning (ML) algorithms require data to operate, and noise in ML training data usually has either a positive or negative effect on algorithm performance. The effect of noise on ML training data for particular algorithms requires investigation, where this experiment involves examining the affects of noise on a Naive Bayes classifier. Results will be inspected across 5 datasets, where each dataset will be tested without noise and with noise. Adding noise to the datasets involves randomly selecting 10% of the features for a given dataset and shuffling values across examples for those features. This will result in 10 separate datasets where 10-fold cross validation will be performed and loss performance will be assessed. Key results from our experiments show that the affect of noise on model performance is dataset-dependent, where some datasets benefited from the noise addition procedure, others did not, and some were indifferent.

1 Problem Under Test

Problem Statement The goal of this experiment is to analyze the effect noise has on Naive Bayes classifier performance. The performance of the algorithm should be tested on multiple metrics and a variety of datasets.

Hypothesis Noisy data will lead to better classification performance after training. The reasoning for this line of thinking is a model that has less precise information when trained might perform better when given more precise data for classification validation. This hypothesis will be tested by analyzing the performance of models trained with noisy and non-noisy input data. Performance will be assessed by examining statistics produced from loss functions.

2 Experimental Approach and Program Design

Proposed Problem Solution To analyze the effect noise has on a Naive Bayes classifier, there are three main areas that need to be considered: data processing, algorithm training,

and algorithm performance evaluation. Within the dataset class, the data will receive two levels of pre-processing (with/without noise) and be folded. This will result in two trainable datasets from each raw data file. By using 2 different pre-processing methodologies, model performance can be evaluated on training data with and without noise characteristics.

Once the data is processed, the next step is to train models to classify examples from the datasets. This will take place within the algorithm class. Training begins by selecting a test fold from the 10 folds created during processing. The other 9 folds are then used to train a model on the class probabilities and conditional feature probabilities. After the model is trained, the model will classify examples from the test set and save its responses.

This process is then repeated, selecting a different fold for testing, and training a new model on the remaining 9 folds each time. This repeats until every fold has been used as the test set, training 10 models for each dataset. Once the 10 models have been trained and the examples are classified, loss is calculated using two metrics. These two metrics can then be used to evaluate the performance of the algorithm.

Dataset The goal of the dataset class is to prepare the raw data for training. This involves different levels of processing for each dataset. To process the data, five methods were included in the class. These methods provided the functionality to save, impute, discretize, remove a feature, and add noise to the data. To impute the data, either a number from one to ten was randomly selected or yes or no was randomly chosen, depending on which data set was being worked with. To discretize the data, we utilized binning to adjust continuous feature values into a range of discrete values. The process of adding noise was accomplished by selecting 10% of the features, shuffling the feature values, and redistributing them to the examples in the dataset. Each dataset had to be processed in two ways, low level processing where imputation is performed as necessary, and high level processing where the data was imputed, discretized and cleaned (removing non-attributes from the data) when necessary, and noise is added.

- **Iris Dataset** For the low level processing of the Iris dataset, the data was discretized only. For the high level of processing, the data was discretized and noise was added.
- **Glass Dataset** For the low level processing of the glass dataset, the data was discretized and cleaned, removing the ID number from the first index. For the high level of processing, the data was discretized, cleaned, and noise was added.
- **Voter Dataset** For the low level processing of the voter dataset, the data was only imputed. For the high level of processing the data was imputed and noise was added.
- **Soybean Dataset** For the low level processing of the soybean dataset, nothing was required. For the high level of processing, noise was added to the data.
- **Cancer Dataset** For the low level processing of the cancer dataset, the data was imputed and the patient ID in the first index was removed. For the high level of processing, the data was imputed, cleaned, and noise was added.

Algorithm The algorithm that is being tested is a Naive Bayes classifier. The goal of this algorithm is to classify examples using conditional probabilities. To do this the algorithm

uses Bayes rule. Bayes rule can be described by the following equation:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{1}$$

where the posterior probability, or the probability of an example belonging to a class given its features, $P(B|A)$, is equal to the probability of the features given the class, $P(A|B)$, multiplied by the probability of the class, P(B), all divided by the probability of the features P(A).

To simplify implementation, this algorithm assumes conditional independence of the features within a given class. This results in worse performance during classification but significantly simplifies the computational complexity, hence the naive description of the algorithm.

Once the algorithm algorithm has calculated the posterior probability of each class and the conditional probabilities for a set of features, it will be given an example to classify. The algorithm will then return the class that is most probable based on the features in the example. This can be described by the following equations:

$$C(x) = P(c_j) \prod_i P(x_i|c_j) \tag{2}$$

$$\text{class}(x) = \arg\max_{c_j \in C} C(x) \tag{3}$$

Where equation (2) shows the process for determining the probability of each class and equation (3) shows the method for returning the most probable class. To calculate the probability that a set of features is in a given class, find the product of all feature probabilities given the class, $\prod_i P(x_i|c_j)$, and multiply by the probability of the class within the dataset, $P(c_j)$. Equation (3) is then used to determine which class is most probable. Once the most probable class has been predicted the algorithm is complete and the result can be analyzed. By letting the algorithm make many predictions on multiple different datasets, it is possible to evaluate the performance of the algorithm as a whole. In our experiments, we are evaluating the effect of noise in the data.

Training During training, the algorithm needs to calculate the probability of each class within the dataset and the probability of each feature given a class. Calculating the probability of each class in the dataset entails counting the number of examples belonging to a class, dividing by the number of examples in the dataset, and repeating for each class. This is shown in the following equation:

$$Q(C = c_i) = \frac{\#\{x \in c_i\}}{N} \tag{4}$$

where the numerator is the number of examples, x, that are in class c_i, and the denominator, N, is the number of examples in the dataset.

The next step is to calculate the probability of a feature, given the class it's in. This can be described by the following equation:

$$F(x_i|C_i) = \frac{n_{x_i,C_i} + 1}{n_{C_i} + d} \tag{5}$$

Where n_{x_i,C_i} represents the number of examples in class C_i with feature x_i, and n_{C_i} is the total number of examples in class C_i. In the denominator, d represents the number of features used to describe an example. When you add 1 in the numerator and d tin the denominator,the feature probabilities are smoothed. This is done so that there is no longer a chance of getting a feature probability of 0 if a specific feature never occurs within a given class in training. Without doing this, the result of the product of the individual conditional feature probabilities, calculated in equation (2), could return 0, meaning it is impossible for an example with the specified feature to belong to the class.

Evaluation Metrics The algorithm's performance was evaluated using two metrics. The first metric, 0/1 Loss, presents what proportion of examples were incorrectly classified. This metric was chosen because it provides a high level overview of the classifier's performance across the whole data set. The equations to calculate 0/1 loss are:

$$L(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y \neq \hat{y} \end{cases} \tag{6}$$

$$L_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq \hat{y}_i) \tag{7}$$

where equation (6) shows how to classify predictions as either a 0 or a 1 depending on whether the prediction was correct, and equation (7) shows how to calculate 0/1 loss for an entire dataset. This is done by summing the number of incorrect predictions and dividing by the total number of predictions.

The second metric used to evaluate algorithm performance is F_1 score. F_1 score is calculated on a per class basis using precision and recall. The equations for precision and recall can be seen below:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{8}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{9}$$

Where true positives are examples in class C_i that were correctly classified as C_i, false positives are examples not in class C_i that were classified as C_i, and false negatives are examples in class C_i that aren't classified as C_i. After precision and recall have been calculated for a class, F_1 score can be calculated using the following equation:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

This equation represents a harmonic mean of precision and recall. Once F_1 score has been calculated for each class, the scores are averaged using a micro-averaging approach to provide a dataset wide metric for performance analysis. The process for micro averaging sums the class based F_1 scores and divides that sum by the number of classes.

Using these two metrics, the performance of the algorithm can be evaluated over an entire dataset, and the effect of adding noise into the data can be analyzed. Assuming perfect performance of the classifier, a dataset would have a 0/1 loss score of 0 and an F_1 score of 1.

3 Results

3.1 Glass Dataset Classification Results

The Glass Identification Database (September, 1987) contains 214 examples of different types of glass. The examples are distributed into 7 different classes. Each example has 10 features and a class label. Of the 10 features, 9 were used within training with the ID number found in the first index of each example excluded from the data. The 9 training features are all continuously valued and fall within a range specific to each feature. There are no missing features in the dataset.

Training Data	Measure	Positive Class	Statistics			
			Mean	Min.	Max.	Std.
w/o Noise	0/1 Loss	—	0.402	0.238	0.619	0.101
	F_1	Class 1	0.780	0.615	0.923	0.099
		Class 2	0.725	0.400	0.923	0.170
		Class 3	0.208	0.000	0.500	0.220
		Class 4	—	—	—	—
		Class 5	0.000	0.000	0.000	0.000
		Class 6	0.360	0.000	1.000	0.349
		Class 7	0.579	0.000	1.000	0.336
w/ Noise	0/1 Loss	—	0.420	0.238	1.000	0.590
	F_1	Class 1	0.754	0.500	0.923	0.129
		Class 2	0.703	0.400	1.000	0.174
		Class 3	0.250	0.000	0.666	0.211
		Class 4	—	—	—	—
		Class 5	0.066	0.000	0.666	0.200
		Class 6	0.203	0.000	0.666	0.240
		Class 7	0.613	0.000	1.000	0.259

3.2 Voters Dataset Classification Results

The 1984 United States Congressional Voting Records Database contains 435 examples where each example represents one voter. Each example of a voter has 16 features as well as the political party they belong to. The 16 boolean valued features are used to train the model before it classifies examples into a political party. Of the 435 examples, 267 are democrats, and 168 are republicans. Many of the examples have missing features, including one example with every feature missing.

Training Data	Measure	Positive Class	Statistics			
			Mean	Min.	Max.	Std.
w/o Noise	0/1 Loss	—	0.087	0.045	0.139	0.032
	F_1	Democrat	0.928	0.900	.957	0.020
		Republican	0.879	0.727	.957	0.070
Noise	0/1 Loss	—	.101	0.047	0.159	0.034
	F_1	Democrat	0.913	0.844	.962	0.034
		Republican	0.874	0.800	.941	0.039

3.3 Iris Dataset Classification Results

The Iris Plants Database (July, 1988) contains 150 examples of Iris plants split evenly between 3 different classes. Each example has 4 features and a class label. Each feature is given a continuous value within a specified range that changes between features. None of the examples in the dataset are missing feature values.

Training Data	Measure	Positive Class	Statistics			
			Mean	Min.	Max.	Std.
w/o Noise	0/1 Loss	—	0.133	0.066	0.333	0.094
	F_1	Setosa	0.950	0.727	1.000	0.081
		Versicolour	0.881	0.666	1.000	0.118
		Virginica	0.770	0.285	1.000	0.183
w/ Noise	0/1 Loss	—	0.046	0.000	0.200	0.060
	F_1	Setosa	0.978	0.857	1.000	0.046
		Versicolour	0.935	0.666	1.000	0.102
		Virginica	0.957	0.842	1.000	0.056

3.4 Soybean Dataset Classification Results

The Small Soybean Database (1987) contains 47 examples of diseased soybeans. The examples are distributed across 4 classes with 10 examples each in D1, D2, and D3, and 17 examples in D4. Each example has 35 discrete valued features. There are no missing feature values within the dataset.

Training Data	Measure	Positive Class	Statistics			
			Mean	Min.	Max.	Std.
w/o Noise	0/1 Loss	—	0.325	0.000	0.600	0.197
	F_1	D1	.427	0.000	1.000	0.436
		D2	0.453	0.000	1.000	0.400
		D3	0.557	0.000	1.000	0.471
		D4	0.267	0.000	.667	0.327
w/ Noise	0/1 Loss	—	0.255	0.000	0.600	0.185
	F_1	D1	0.367	0.000	1.000	0.046
		D2	0.567	0.000	1.000	0.396
		D3	0.467	0.000	1.000	0.476
		D4	0.697	0.000	1.000	0.3854

3.5 Breast Cancer Dataset Classification Results

The Wisconsin Breast Cancer Database (January 8, 1991) contains 699 examples of breast cancer diagnoses. Each example has 10 discrete valued features. For training, the ID number in the first index was removed. There are 458 examples of diagnoses involving a benign tumor and 241 examples of diagnoses involving malignant tumors. There are 16 examples in the dataset with a missing feature value.

Training Data	Measure	Positive Class	Statistics			
			Mean	Min.	Max.	Std.
w/o Noise	0/1 Loss	—	0.026	0.014	0.057	0.014
	F_1	Benign	0.980	0.952	.990	0.012
		Malignant	0.962	0.929	.982	0.020
w/ Noise	0/1 Loss	—	0.024	0.000	0.057	0.016
	F_1	Benign	0.981	0.955	1.000	0.012
		Malignant	0.964	0.919	1.000	0.025

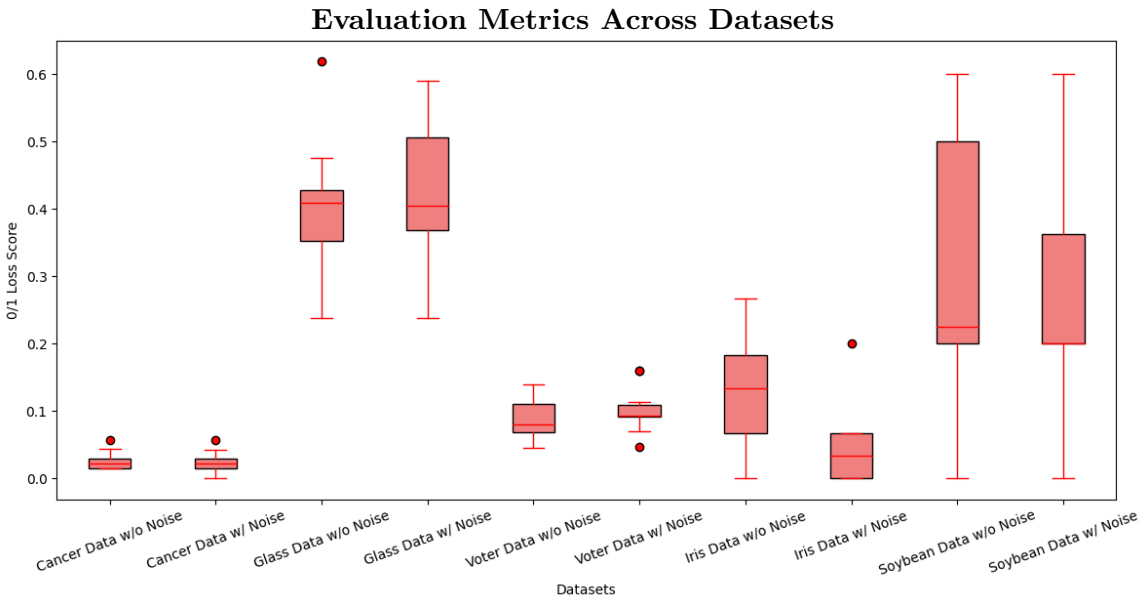

Figure 1: 0/1 Loss Scores Across Datasets

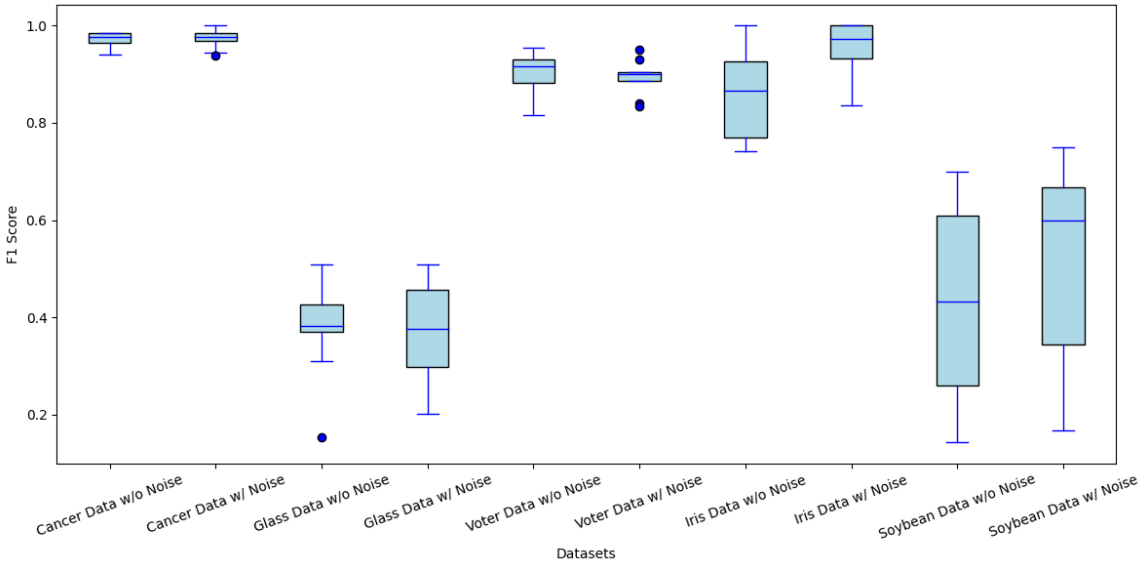

Figure 2: Average F_1 Scores Across Datasets

4 Discussion and Conclusions

4.1 Analysis of Results

When examining model performance results for each dataset, there are a few key takeaways that stand out. First, it is notable that the cancer, voter, and iris datasets performed significantly better than the glass and soybean datasets for classification. This performance contrast is best demonstrated in figure 2, where F1 scores for the glass and soybean datasets perform worse across the board when compared to all other datasets. When considering the performance difference between datasets with and without added noise, the verdict for which pre-processing method performs best is unclear. Both the soybean and iris datasets produced better average performance metrics with noise (high F1 score, low 0/1 loss), whereas the voter and glass datasets had an inverse reaction. Meanwhile, the difference between performance metrics for the cancer datasets with and without noise is statistically insignificant. The key takeaway from these tests is that more data needs to be collected with a more robust testing procedure. Limitations to our methodology are outlined below, but future work would include testing the proposed problem on more sets of data, larger sets of data, and adjusting noise injection to levels above and below 10% to determine the exact receiver operating characteristic with respect to noise.

4.2 Limitations

Key factors limit the viability of our results. One major limitation of our implementation is how we handled folding the data during processing. Ideally, the data is shuffled, duplicated, noise is added to one of the sets, and both datasets are folded in the same way so that all of the folds are identical in what examples they hold. Instead, the processing we used to prepare the data duplicated it, shuffled both datasets individually, added noise to one dataset, and folds both independently. This creates a discrepancy in what data we use to test our algorithm. With ideal processing, the models are trained on the exact same set of examples and classify the exact same set of examples. In our processing, since the data is shuffled individually, the training data and the test data don't contain the same examples across the two duplicated datasets. This adds more variability to the results of training on a single dataset with and without noise, which hinders our ability to determine how well a Naive Bayes classifier handles noise. Another limitation of this experiment is the size of the datasets. As the datasets get smaller, there is less data to train the model on. This creates a situation where the model doesn't have enough experience to determine what features correspond to what class. For example, a doctor who has diagnosed 1000 patients is likely able to understand which symptoms correspond to which illnesses better than a doctor who has diagnosed only 10 patients. The same is true for classification models.

4.3 Summary

The goal of this project was to implement a Naive Bayes classifier algorithm and evaluate its performance on datasets with and without noise. Understanding classification algorithms and what effects their performance is important for applications in a wide variety of fields. Trying to contribute to this understanding, we developed and tested a multinomial classifier on 5 different datasets. To analyze the effect of noise, we implemented 10

fold cross-validation to provide a more diverse set of models to evaluate. We then used 0/1 loss and F_1 score to assess how well the models performed. This was accomplished by developing two Python classes with one class handling the data processing and one class handling training and classification. Looking over the results in Figures 1 and 2, we found a negligible difference in performance for datasets with and without noise. This went against our hypothesis that adding noise to the training data would lead to better performance during classification. This led us to the conclusion that Naive Bayes classifiers don't benefit from being trained with noise to the extent we had originally predicted.

5 References

Materials and ideas discussed were pulled exclusively from CSCI-447 lecture content and class material.

6 Appendix

Throughout the duration of this project, both group members worked together to complete the necessary tasks, meeting daily to discuss progress and redistribute work when necessary. Both group members feel the work load was equitable and balanced, with neither member providing significantly more than the other to any task.

Task	Completed By:
Design Document	Finn/Carlos
Method 1 Pre-processing	Finn
Method 2 Pre-Processing	Carlos
Training Algorithm Part 1	Finn
Training Algorithm Part 2	Carlos
Classification	Finn
Process Verification	Carlos
Project Report	Finn/Carlos
Project Video	Finn/Carlos

Task Delegation