Journal of Machine Learning Research 20 (2019) 1-10 Submitted 9/19; Published 10/19

# Performance Effects of Data Scrambling Using Naive Bayes

**Brandon Sladek** BRANDONSLADEK@GMAIL.COM
*School of Computing*
*Montana State University*
*Bozeman, MT 59718, USA*

**Jared Thompson** J.A.THOMPSON22@GMAIL.COM
*School of Computing*
*Montana State University*
*Bozeman, MT 59718, USA*

**Kyle Hagerman** HAGERMANKYLE96@GMAIL.COM
*School of Computing*
*Montana State University*
*Bozeman, MT 59718, USA*

**Ryan Hansen** RYANHANSEN2222@GMAIL.COM
*School of Computing*
*Montana State University*
*Bozeman, MT 59718, USA*

Editor: N/A

## Abstract

In this project we ran the Naive Bayes algorithm on five different data sets from the UCI machine learning repository. We ran the algorithm on the five data sets and recorded the corresponding average prediction accuracy for each data set, and then re-ran the algorithm on scrambled versions of the five data sets in order to compare the resulting prediction accuracy measures against the scrambled data sets. We hypothesized that scrambling the data would result in significant decreases in prediction accuracy across all the scrambled data sets. However, our results showed less of a performance decrease than we expected to see, and in some cases performance actually improved. In the Analysis section below we discuss whether these results are expected, or due to some issue in the program implementation.

**Keywords:** Naive Bayes, Cross Validation, Feature Scrambling

## 1. Introduction

This project was meant to provide an introduction to the field of machine learning by having us implement the Naive Bayes algorithm on five different data sets from the UCI machine learning repository. In retrospect, the project was a great introduction to machine learning in terms of both the analytical skills needed to analyze data sets and make decisions based on the individual characteristics of each data set, as well as the programming skills needed to write code that can manage and analyze the data sets in a logical and scalable manner. For this project we chose to use Python due to its ease of use with respect to scripting and quick prototyping. We also made use of the popular Pandas library as our data structure

for storing the data sets McKinney (2010).

## 2. Problem

The problem as it was posed to us in this project was to investigate the effects of data scrambling on the performance (prediction accuracy) of a Naive Bayes algorithm implementation. Our task was to write the necessary code to read the regular data sets and run Naive Bayes against them using cross validation, and then re-run the Naive Bayes algorithm against scrambled versions of the data sets in which the values in 10% of the features were randomly shuffled in order to introduce noise, and then analyze the impact scrambling had on the resulting prediction accuracy of Naive Bayes with respect to its corresponding performance against the regular data sets.

## 3. Algorithm: Naive Bayes

### 3.1 Description

The Naive Bayes algorithm is a probabilistic classification supervised learning algorithm. It relies on Bayes' Rule for doing the classifications based on the likelihoods (probabilities) of each class value and each unique attribute value for a given class. The "Naive" part of the algorithm name comes from the fact that Naive Bayes assumes all the attributes are conditionally independent, which is often a "naive" assumption Ren et al. (2009). By making this assumption, the algorithm can strictly classify new observations with clearer margins between classes. The classification of a new observation is based solely on the maximum probability of which class the new data point belongs to. To ready an implementation of the algorithm for it to be able to classify new test cases it must be trained used a subset of the data for which we know the corresponding class values. It is common to use a training set and a testing set of known data to examine the accuracy of the model before use.

### 3.2 Implementation

The script was broken down into classes with each representing a piece of the work needed to implement a fully realized solution. In order of execution, the process followed these steps: data acquisition, data preprocessing, execution of Naive Bayes with cross validation and results processing to calculate loss function values, and a final experimental runner script to execute the code in order.

The data acquisition class, named DataApi, operates by pulling the data set information from the local csv files and creating a Pandas DataFrame object for each data set. Put simply, a Pandas DataFrame object is just a Python 2D array with some extra helper methods for accessing and analyzing the data given row and column indexes. The DataApi class also has some utility methods needed by various part of the application.

The DataFrame objects are then passed to the Preprocessor class which returns two data sets depending on the arguments passed to it: (1) a copy of the original DataFrame, possibly

modified to account for any missing data values or discretization of continuous attributes (see Approach section), or (2) a copy of the original DataFrame with 10% of the attributes (feature columns) randomly scrambled so that each row value in a scrambled column is randomly assigned to a different row in the column, thus introducing noise into the data in a controlled manner.

Then the ExperimentRunner class takes each of these preprocessed data sets (some regular, some scrambled) and runs the Naive Bayes implementation against each of them independently by passing each DataFrame to the Naive Bayes class. The Naive Bayes has a run_naive_bayes() method that takes in a training set and a test set, and returns a dictionary of predicted class values for each of the instances in the test set.

The training set and test set combinations are created using the CrossValidation class, which just has simple methods for randomly partitioning the data into ten partitions of roughly equal size. The CrossValidation class returns the appropriate training and test sets given one integer argument, which in our case is the value of the test set. Per each iteration of 10-fold cross validation, the Naive Bayes algorithm is called on the training set, and tested against the test set. The predictions are then passed to the ResultsProcessor class, which is responsible for calculating the corresponding 0/1 Loss and Mean Squared Error loss function values for a given set of predictions.

Finally, the ExperimentRunner class is used to orchestrate the entire application flow by creating instances of each class, and calling the methods on each class in the appropriate order as described above. The ExperimentRunner class has a run_experiment() method which runs the entire execution of the application for a given data set. The whole set of experiments (execution of Naive Bayes on all regular and scrambled data sets using cross validation with results processing) is ran by simply running the experiment_runner.py script.

### 3.3 Preprocessing

We made a few key decisions in terms of how to preprocess the various data sets. The Breast Cancer data set only had 16 rows with missing values, out of a total of 699 rows in the raw data set from the UCI repository. Since this is a very small ratio of rows with missing values with respect to the total number of rows in the data set, we figured it would make the most sense to just remove those rows with missing values from the breast cancer data set. On the other hand, the House Votes data set had a much higher ratio of rows with missing values compared to the total number of rows. Given that the ratio was higher, and all values in the House Votes data set are simply binary yes/no values, we figured it would make more sense to replace missing values in the House Votes data set with randomly generated yes/no values.

Finally, the Glass and Iris data sets contain features with continuously-valued data, so we had to discretize these columns before running Naive Bayes on them. We tried two different approaches to discretizing the values, putting the values into 1 of 5 fixed-width bins based on the range of values in a given column, or putting the values into 1 of 2 fixed-width bins

based on the average value of the column and whether the value to be binned is less than or greater than the average.

## 4. Experiment

### 4.1 Approach

In framing the experimental approach, two primary decisions were made regarding which loss functions to use for measuring prediction accuracy, and how to deal with the continuous data. For loss functions we chose the 0-1 Loss Function and Mean Squared Error. Both functions can be used to provide a measure of accuracy of our model. 0-1 Loss is a very basic performance measure that counts the number of classifications we got wrong for a given test set. For final results we calculate the average count of wrong classifications over all ten test sets analyzed in the 10-fold cross validation.

For our second loss function we chose to use Mean Squared Error (MSE). Given the binary nature of our classification correctness (wrong or right), we had to alter the calculation of MSE to the following: MSE $= \frac{1}{n}\sum_{t=1}^{n}(Y - Y')^2$, where $Y$ is the expected value and $Y'$ is the calculated value. We defined a correct classification to be 0 so each value of Y is equal to 0, and each incorrect classification of an observation is 1. Y' can be either 0 or 1, giving us a sum of 1's equal to the number of incorrect classifications. The number of incorrect classifications will then be divided by the number of classifications attempted for an accuracy measure. Then we calculate an average MSE value over all the folds in the 10-fold cross validation, just like with the 0-1 Loss. Therefore, if our value is close to 1 our classifications are very wrong, and if we're close to 0 our classifications are mostly right.

### 4.2 Hypothesis

We predicted that data scrambling would have a significant detrimental effect on the prediction accuracy of the Naive Bayes implementation. Our reasoning was as follows: since Naive Bayes relies on probability calculations for the likelihoods of each unique class value and each unique attribute value given a specific class value, we figured that randomly shuffling values for 10% of the features in a data set would introduce enough noise to reduce the overall prediction accuracy of Naive Bayes, since the probability calculations would be less representative after scrambling.

### 4.3 Results

The results of our experiments on the regular and scrambled data sets are displayed in the tables below. The "Avg Test Set Size" column shows the average number of test instances in the test sets used throughout the 10-fold cross validation, as an indicator of how many predictions were made on average for each test set. The "0/1 Loss (Wrong)" column shows the average number of wrong classifications for a given data set. And the "MSE" column shows the average Mean Squared Error for a given data set.

| Prediction Accuracy - Regular Data Sets | | | |
|---|---|---|---|
| Data Set | Avg Test Set Size | 0/1 Loss (Wrong) | MSE |
| breast_cancer | 82.7 | 7.6 | 0.092 |
| glass | 21.4 | 8.8 | 0.409 |
| iris | 15 | 5.9 | 0.393 |
| soybean_small | 4.7 | 4.7 | 1.0 |
| house_votes | 43.5 | 4.5 | 0.104 |

| Prediction Accuracy - Scrambled Data Sets | | | |
|---|---|---|---|
| Data Set | Avg Test Set Size | 0/1 Loss (Wrong) | MSE |
| breast_cancer | 81.1 | 5.3 | 0.065 |
| glass | 21.4 | 9.2 | 0.430 |
| iris | 15 | 5.6 | 0.373 |
| soybean_small | 4.7 | 4.7 | 1.0 |
| house_votes | 43.5 | 4.3 | 0.100 |

## 5. Analysis

For the most part, our hypothesis was incorrect. In reviewing our results, we noticed that in some cases the prediction accuracy of Naive Bayes actually improved with the data scrambling. Unfortunately, we think this is most likely because of a bug in the Python code. Intuitively, it makes sense that scrambling the data should result in decreased prediction accuracy for Naive Bayes, which is why we think there must be something wrong somewhere in the code. Besides the unexpected occasional increase in prediction accuracy after data scrambling, we also noticed that we are getting fairly poor results on the Glass and Iris data sets, each of which averaged around a 60 percent prediction accuracy, along with terrible results on the Soybean data set, which averaged 0 percent prediction accuracy.

### 5.1 Parameter Tuning

We initially started with discretizing the data sets with continuous data using a five fixed-width binning approach. To do this, we simply divided the range of a column by five, and then placed each continuous value into bins 1-5 depending on where that value fell in the range. We found that this did not provide good results for the data sets with continuous data (Glass and Iris). As an alternative approach, we tried putting all of the continuous values in a column into just one of two bins, where all values less than the average value of a column went into the first bin, and all values greater than the average value of a column went into the second bin. This did show a significant improvement in the prediction accuracy for the Glass data set, which went down from an average count of wrong classifications of 19 of 21, to around 9 of 21. In other words, we saw almost a 50 percent increase in prediction accuracy by just changing our discretization strategy.

### 5.2 Possible Improvements

Assuming there is not an issue with the code itself, we think there is room for improvement by doing further parameter tuning. More specifically, we think the prediction accuracy results could potentially be improved by using a different discretization strategy. We tried modifying the two-bin averaging approach described above to use the median value in a column instead of the mean, but that did not show any improvement in prediction accuracy. Unfortunately we did not have enough time to try other approaches to discretizing the data, but given that we saw good results on the data sets that had already been discretized (Breast Cancer and House Votes), we think there's reason to believe our results could be improved further by tuning the discretization parameters to better represent the continuous data. Along the same lines, if we had more time on this project we would have investigated dimensionality reduction techniques that could have been used to only analyze the most useful columns, and discard the other columns that didn't give us as much information.

## 6. Conclusion

This project involved a significant amount of programming, along with a fair amount of reasoning about the data sets and how best to approach and analyze each one. We saw good results in terms of prediction accuracy on the Breast Cancer and House Votes data sets, relatively poor results on the Glass and Iris data sets, and bad results on the Soybean Small data set. If we were to continue working on this project we would investigate the code further to determine if there is a bug causing the poor results, and look into other discretization strategies and data preprocessing techniques (such as dimensionality reduction) to better represent and analyze the data.

## Acknowledgments

We acknowledge the following data sets used for this assignment. All of the cited data sets below come from the UCI machine learning repository (urls in references below).

Breast cancer data: Wolberg (1992)
Glass data: German (1987)
Iris data: Fisher (1936)
Soybean small data: Michalski (1980)
House votes data: Almanac (1985)

## References

Congressional Quarterly Almanac. UCI machine learning repository, 1985. URL https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records.

R.A. Fisher. UCI machine learning repository, 1936. URL https://archive.ics.uci.edu/ml/datasets/Iris.

B. German. UCI machine learning repository, 1987. URL https://archive.ics.uci.edu/ml/datasets/Glass+Identification.

Wes McKinney. Data structures for statistical computing in python, 2010. URL https://pandas.pydata.org/.

R.S. Michalski. UCI machine learning repository, 1980. URL https://archive.ics.uci.edu/ml/datasets/soybean+(small).

J. Ren, S. D. Lee, X. Chen, B. Kao, R. Cheng, and D. Cheung. Naive bayes classification of uncertain data. In *2009 Ninth IEEE International Conference on Data Mining*, pages 944–949, Dec 2009. doi: 10.1109/ICDM.2009.90.

Dr. William H. Wolberg. UCI machine learning repository, 1992. URL https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28original%29.