**Traumatized Bayes: Naive Bayes with a Correlation Coefficient**

Machine Learning Quarter 2 Project Intermediate Report

An update on our current progress in improving the Naive Bayes classification model

Isabella Zhu and Lilian Zhu

1/23/2022

ML 1 - Yilmaz

2022-23

1/30/2022

**Quarter 2 Intermediate Report**

**Abstract**

This paper contains a generalization of Naive Bayes towards datasets with dependent attributes. In particular, our algorithm, Traumatized Bayes, calculates the CramerV correlation coefficient between each pair of attribute values and incorporates this correlation coefficient in calculating the conditional probability of an instance belonging to a specific class. Traumatized Bayes turns out to be more accurate than traditional Naive Bayesian methods on datasets with dependent attributes, without sacrificing accuracy on datasets with independent attributes. Furthermore, Traumatized Bayes does not sacrifice computational speed, as compared to other advanced Bayes algorithms such as Hidden Naive Bayes.

**Introduction**

The problem of classifying unknown instances is a common one; common uses include predicting plant species, susceptibility to diseases, and ability to repay loans. Naive Bayes is a probabilistic classifier algorithm that achieves fairly high accuracy levels. However, Naive Bayes makes the assumption that all attributes are independent of each other, which is not true of most real life data. When this assumption doesn’t hold, Naive Bayes decreases in accuracy. Our goal is to build upon the Naive Bayes algorithm, increasing the accuracy when the attributes are correlated without sacrificing accuracy when the attributes aren’t correlated. We tested the algorithms on two datasets: COVID-19 Dataset[9](#kkz4j598fbjd) and credit-g[2](#7zv6w4eio453). COVID-19 Dataset has independent attributes and credit-g has dependent attributes. The inputs for COVID-19 Dataset are patient level, medical unit, sex, patient type, intubed, pneumonia, age, pregnancy, and whether the patient had diabetes. The original output (class variable) was the date of patient death; since this would cause data to be sparse, we replaced it with “Did the patient die?”, which outputs 1 if the patient dies (Date of death listed) and 0 if the patient did not die (Date of death listed as 9999-99-99). The inputs for our second dataset, credit-g, are checking status, duration, credit history, purpose, credit amount, savings status, employment, installment commitment, personal status, other parties, residence since, property management, age, whether they have other payment plans, housing, existing credits, job, number of dependents, whether they own a telephone, and whether they’re a foreign worker. The output (class variable) is whether their credit rating is good or bad.

**Related Work**

Most attempts to improve Naive Bayes have centered on decreasing the assumption of independence between attributes.[5](#ggkm9rul6wjw),[6](#bp17vqtl59hj35CuO5I6gtP2dMh2SLc/edit),[13](#72bs0l5zcbpu) Our algorithm attempts to do the same. The chief issue with not assuming independence, however, is that it vastly increases the computational complexity and the volume of data required.[13](#72bs0l5zcbpu) One simple, yet effective, approach to this issue is the one utilized in Averaged One-Dependence Estimators[13](#72bs0l5zcbpu) and Hidden Naive Bayes[5](#ggkm9rul6wjw): considering the correlations between pairs of attributes, but assuming independence in triplets and larger groups.

Naives Bayes with Weight Function[13](#72bs0l5zcbpu) aimed to decrease computational complexity for binary classification models. By restricting the classification to binary classification, an approach we found innovative yet overly restrictive, the author reduced the number of instances required to achieve passable accuracy. The author also included a weight function, which places more emphasis on non-outliers, to reduce the impact of outliers on the algorithm. The other attempts to improve Naive Bayes did not include protection against outliers, which retains Naive Bayes’ weakness to outliers. This is in line with what we did; calculating a weight function increases the complexity of the algorithm, sacrificing a lot of speed for comparatively little improvement in performance.

Gaussian Naive Bayes[7](#i97g5bx09qt6) generalized Naive Bayes towards quantitative data by assuming a normal distribution. Since binomial and geometric distributions are also quite common, we considered this assumption to be too restrictive; instead, we dealt with quantitative data via discretization

All of the sources we examined evaluated algorithms based on two parameters: computational speed and accuracy. After finding the speed and accuracy of all algorithms, the results were examined by the researchers. There is potential for human bias in the “results” page of the studies; if one algorithm is faster and the other algorithm is more accurate, then it’s fairly arbitrary which one the researchers labeled “better”.

**Dataset and Features**

We evaluated algorithm performance based on two datasets: data-g[2](#7zv6w4eio453) and COVID-19[9](#kkz4j598fbjd). The dataset credit-g was included in the initial WEKA download, and predicts whether a given German has a good or bad credit (in other words, whether they are likely to default on a loan or not). There are 333 test cases and 667 training cases. The only preprocessing done was splitting the original dataset into the test and training datasets.

The second dataset was COVID-19, sourced from the official website of the Mexican government, as posted by Hector Parades. The original dataset contained an attribute, DATE-DIED, which contains the date of patient death if the patient died and 9999-99-99 if they survived COVID. We discretized this attribute, replacing it with “1” if the patient died and “0” if the patient survived. Since the dataset was large and unwieldy, we took a random sample of 100,000 instances from the original dataset and used that instead. After making this change, we split the original COVID-19 dataset into a test and a training dataset. The test dataset contains 33,333 instances and the training dataset contains 66,667 instances; this is the same 33/67 test/training split used in credit-g. The original dataset had no clearly labeled class variable; we chose the class variable to be whether a given patient died or not.

We also discretized the quantitative attributes; we used the discretized version to test every implemented algorithm except Gaussian Naive Bayes, which requires quantitative variables. For Gaussian Naive Bayes, we used the original dataset.

**Methods**

We implemented four algorithms total: Naive Bayes, Traumatized Bayes, Hidden Naive Bayes, and Gaussian Naive Bayes. Naive Bayes is the benchmark for comparison, Traumatized Bayes is our algorithm, and Hidden Naive Bayes and Gaussian Naive Bayes are other advanced Bayes algorithms for comparison. Below, we include the theory and our implementation for each algorithm.

***Naive Bayes***

*Theory*

Given instance X = (x1, x2, …, xn) in a dataset with classes Y1, Y2, …Ym, Naive Bayes approximates P(Y1 | X), P(Y2 | X), …P(Ym | X) and predicts class a such that P(Ya | X) is maximized. To do this, it turns to Bayes’ theorem, after which the algorithm is named. According to Bayes’ theorem, . Assuming conditional independence between the attributes, . Since X is constant throughout this process, P(Ya|X) is thus maximized when P(X1|Ya)P(X2|Ya)...P(Xn|Ya)P(Ya) is maximized. Iterating over each X, Naive Bayes thus predicts Ya such that this value is maximized.

*Implementation*

Our model consists of two components: class\_probs and cond\_probs. class\_probs is a list consisting of the values, while cond\_probs is a dictionary consisting of values, iterating over every attribute, every value of that attribute, and every class.

Pseudocode:

# model

for every class c:

class\_probs.append(number of instances of class c / total)

for every attribute X:

for every value v of attribute X:

for every class c:

cond\_probs[(X, v, c)] = number of instances with value v and class c / number of instances class c

To calculate the probability of a class given a certain instance, we use the formula mentioned in the theory section above. We iterate over each class and compute the probability and classify the instance as the class corresponding to the highest probability.

***Traumatized Bayes***

*Theory*

Our algorithm, Traumatized Bayes, is a variation on Naive Bayes that weakens the assumption of independence between attributes. Given instance X = (x1, x2, …, xn) in a dataset with classes Y1, Y2, …Ym, we approximate , where

and is the cramerV correlation coefficient between attributes and . This coefficient is calculated using the chi-square test.

*Implementation*

Our model consists of three components: class\_probs, a list of , cond\_probs, a dictionary consisting of values, and corr\_coeffs, a dictionary consisting of values. class\_probs and cond\_probs are calculated in the same way as in Naive Bayes. is calculated using a chi-square value, which we did through the package scipy.

Pseudocode for calculating correlation coefficients:

for every class c:

subdata = [instances that are class c]

for Ai, Aj attributes:

use chi-square test to calculate correlation between Ai and Aj in subdata

(specifically: p\_ij = sqrt(chi square / len(subdata) / min\_dimension))

***Hidden Naive Bayes***

*Theory*

Hidden Naive Bayes is a variation of Naive Bayes that weakens the assumption of independence between attributes (which is also the goal of our project). Given instance X = (x1, x2, …, xn) in a dataset with classes Y1, Y2, …Ym, Hidden Naive Bayes approximates P(Y1 | X), P(Y2 | X), …P(Ym | X) and predicts class a such that P(Ya | X) is maximized. It does this differently from Naive Bayes, however. Hidden Naive Bayes creates a hidden “parent” node for each attribute. Here, we will refer to the hidden parent of X1 as Z1, the hidden parent of X2 as Z2, and so on, with the hidden parent of Xn being Zn.



It then calculates for all Ya, with

(weights encapsulating influence of other attributes)

(conditional mutual information)

The Hidden Naive Bayes algorithm then predicts class such that , an approximation of , is maximized.

*Implementation*

Our model consists of three structures: w, a dictionary containing weights , parent\_probs, a dictionary containing , and class\_probs, a list containing .

Pseudocode:

for each class c:

compute P(c)

for each pair of attributes Ai, Aj

for each value ai, aj of Ai, Aj and for each class c:

compute P(ai, aj | c)

for each pair of attributes Ai, Aj:

compute Ip(Ai, Aj | C)

for each attribute Ai:

compute Wi = sum(Ip(Ai, Aj | C) for j not equal to i)

for each attribute Aj and j not equal to i:

compute Wij = Ip(Ai, Aj | C) / Wi

One important note is that when computing probabilities, we used a Laplace estimator to ensure that the denominator would be nonzero. For example,

.

Notice that we added the value “number of classes” in the denominator to ensure that it is nonzero.

***Gaussian Naive Bayes***

*Theory*

We also implemented Gaussian Naive Bayes, since it has a high accuracy when working with quantitative data. Much like Naive Bayes, to classify an instance X = (x1, x2, …, xn) in a dataset with classes Y1, Y2, …Ym, Gaussian Naive Bayes also finds class Ya such that P(X1|Ya)P(X2|Ya)...P(Xn|Ya)P(Ya) is maximized. Gaussian Naive Bayes approximates P(Ya) the same way as Naive Bayes, by calculating . It calculates P(X1|Ya), P(X2|Ya), …, and P(Xn|Ya) differently from Naive Bayes, however. To approximate P(Xm|Ya), Gaussian Naive Bayes calculates the mean and standard deviation of Xm’s corresponding attribute for class Ya. It then calculates the z-score of Xm using the formula z-score = . Then, it calculates the p-value of that z-score, and approximates P(Xm|Ya) using the z-score. Note that P(Xm|Ya) tends to be very small; thus, to prevent the computer from storing P(Xm|Ya) improperly (e.g. by truncating decimals or rounding), Gaussian Naive Bayes is often implemented as maximizing ln(P(X1|Ya)P(X2|Ya)...P(Xn|Ya)P(Ya)) = ln(P(X1|Ya))+ln(P(X2|Ya))+...+ln(P(Xn|Ya))+ln(P(Ya)).

*Implementation*

Pseudocode:

For each class:

Calculate the prior probability =

For each attribute:

Calculate the mean and standard deviation of the class-attribute pair

For each testing instance A:

Instantiate list probabilities, which currently contains the prior probabilities for each class

For each class C:

For each attribute B:

Calculate the z-score =

Calculate the p-value of the z-score using a normal distribution

Multiply probabilities[class] by the p-value obtained

Find the class with the highest predicted probability (as stored in list probabilities) and predict that class

**Results**

Our primary metrics are accuracy and confusion matrices (TP, FP rate). We chose these metrics because accuracy is a good holistic measure to test the correctness of a model, while confusion matrices give more specific information on the distribution of results, while also providing a better analysis for data with skewed class distributions.

We tested Naive Bayes and Traumatized Bayes on the iris dataset. We discretized the iris dataset by dividing each feature into 3 equal-width bins.

Both Naive Bayes and Traumatized Bayes achieved a 96% accuracy rate. Furthermore, they also achieved the same confusion matrix. Although Traumatized Bayes did not outperform Naive Bayes on the iris dataset, it at least achieved the same good results.

We still need to test all four algorithms on the credit-g and the COVID dataset. Naive Bayes and Traumatized Bayes are bug-free, but we will probably need to debug Hidden Naive Bayes and Gaussian Naive Bayes before we run tests.

**References**

1. Bhuwanesh. “How to Calculate Cramer’s V in Python?” *Geeks for Geeks*, 28 Feb. 2022, <https://www.geeksforgeeks.org/how-to-calculate-cramers-v-in-python/>. Accessed 20 Jan. 2023.

1. Datopian. “Credit G.” *DataHub*, 2018, https://datahub.io/machine-learning/credit-g.
2. Harris, Charles R., et al. ‘Array Programming with NumPy’. *Nature*, vol. 585, no. 7825, Springer Science and Business Media LLC, Sept. 2020, pp. 357–362, https://doi.org10.1038/s41586-020-2649-2.
3. HughHugh 11111 gold badge11 silver badge33 bronze badges, et al. “Probability of Three Events Occurring given Correlation?” *Mathematics Stack Exchange*, 1 Mar. 1962, <https://math.stackexchange.com/questions/1205928/probability-of-three-events-occurring-given-correlation>.

1. Jiang, Liangxiao, et al. “A Novel Bayes Model: Hidden Naive Bayes.” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 10, 2009, pp. 1361–1371., <https://www.aaai.org/Papers/AAAI/2005/AAAI05-145.pdf>.

1. Larsen, Kim. “Generalized Naive Bayes Classifiers.” *ACM SIGKDD Explorations Newsletter*, vol. 7, no. 1, 2005, pp. 76–81., <https://doi.org/10.1145/1089815.1089826>.

1. Majumder, Prateek. “Gaussian Naive Bayes.” *OpenGenus IQ*, 2023, <https://iq.opengenus.org/gaussian-naive-bayes/>. Accessed 20 Jan. 2023.
2. McKinney, Wes. ‘Data Structures for Statistical Computing in Python’. *Proceedings of the 9th Python in Science Conference*, SciPy, 2010, [https://doi.org10.25080/majora-92bf1922-00a](about:blank).

1. Parades, Hector. “Covid-19 Dataset.” Translated by Meir Nizri, *Kaggle*, 13 Nov. 2022, <https://www.kaggle.com/datasets/meirnizri/covid19-dataset>.
2. Seabold, Skipper, and Josef Perktold. ‘Statsmodels: Econometric and Statistical Modeling with Python’. *Proceedings of the 9th Python in Science Conference*, SciPy, 2010, [https://doi.org10.25080/majora-92bf1922-011](about:blank).
3. Van Rossum, G. *The Python Library Reference, Release 3.8.2. Python Software Foundation*. 2020.
4. Virtanen, Pauli, et al. ‘SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python’. *Nature Methods*, vol. 17, no. 3, Springer Science and Business Media LLC, Mar. 2020, pp. 261–272, [https://doi.org10.1038/s41592-019-0686-2](about:blank).

1. Webb, Geoffrey I., et al. “Not so Naive Bayes: Aggregating One-Dependence Estimators.” *Machine Learning*, vol. 58, no. 1, 2005, pp. 5–24., <https://link.springer.com/content/pdf/10.1007/s10994-005-4258-6.pdf?pdf=button>.
2. Zach. “How to Calculate Correlation Between Categorical Variables.” *Statology*, 27 Sept. 2021, <https://www.statology.org/correlation-between-categorical-variables/>. Accessed 20 Jan. 2023.