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

Machine Learning Quarter 2 Project Final Report

Isabella Zhu and Lilian Zhu

02/06/2022

ML 1 - Yilmaz

2022-23

02/06/2022

**Quarter 2 Final 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 three datasets: Iris[3](#q73q48wql3qm), COVID-19 Dataset[10](#kix.xskw28lpli63) and credit-g[2](#kix.razec7tgk8t2). Iris and COVID-19 Dataset have independent attributes, whereas credit-g has dependent attributes. The inputs for Iris are sepal length, sepal width, petal length, and petal width. The class variable is the species of iris, either Iris Setosa, Iris Virginica or Iris Versicolor. 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 Y if the patient dies (Date of death listed) and N 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.[6](#kix.2bfqctx6bg21),[7](#kix.8cmn72lva166),[14](#kix.4ahqak2wpqk6) 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.[14](#kix.4ahqak2wpqk6) One simple, yet effective, approach to this issue is the one utilized in Hidden Naive Bayes[6](#kix.2bfqctx6bg21): considering the correlations between pairs of attributes, but assuming independence in triplets and larger groups.

Hidden Naive Bayes[6](#kix.2bfqctx6bg21) creates an invisible “parent” node for each “child” attribute. The effect of each attribute on the parent node is then considered individually, creating a pairwise dependence between the parent node and each attribute. The effect of the parent node on the child is then calculated; then, a change in the parent node’s value is reflected as a change in the child node’s value. Since the parent node’s value depends on the value of other attributes, the child node thus indirectly incorporates the influences of all other attributes. Thus, the pairs of attributes are pairwise dependent, but larger groups of attributes are still considered independent to save on computational time.

Gaussian Naive Bayes[8](#kix.yfq3ckgf7hgx) 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, while creating Traumatized Bayes, 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 three datasets: iris[3](#q73q48wql3qm), credit-g[2](#kix.razec7tgk8t2) and COVID-19 Dataset[10](#kix.xskw28lpli63). For each dataset, we preprocessed it by creating two versions: one completely discretized version and one completely continuous version. To create the discretized datasets, we discretized the continuous attributes using three equal-size bins. To create the continuous datasets, we removed all attributes that were discrete. We then used stratified random sampling in a 2 to 1 ratio to split each dataset into training and testing datasets.

**Iris**

The iris dataset is included in the initial WEKA download. It contains 150 instances of iris flowers of three species with length and width attributes. The iris dataset from WEKA comes as a continuous dataset. We discretized each attribute using three equal-size bins (small, medium, and large).

**Credit-g**

The credit-g dataset is included in the initial WEKA download. It contains 1000 instances of Germans with credit cards and determines whether or not they have good credit (in other words, whether they are likely to default on a loan or not). We discretized each continuous attribute using three equal-size bins in WEKA.

**COVID-19**

The COVID-19 dataset is 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 “Y” if the patient died and “N” if the patient survived. This was also our class attribute. Since the dataset was too large to be processed on our computers, we took a random sample of 100,000 instances from the original dataset. We split into training and testing after making this change. We chose this dataset as an example of an unbalanced dataset we wanted to test accuracy on.

**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

**Experiments, Results, and Discussion**

**Experiment**

Our metrics are accuracy, macro-precision, and macro-recall. We also measured the run time (build model and testing) to test speed. We tested the four algorithms mentioned above in the Methods section on three different datasets mentioned in the Datasets section. Here are our results:

**Iris Dataset**

|  | Naive | Trauma | Hidden | Gaussian |
| --- | --- | --- | --- | --- |
| Accuracy | 94.1% | 94.1% | 94.1% | 94.1% |
| Confusion[[1]](#footnote-0) | |  | VE | VI | SE | | --- | --- | --- | --- | | VE | 15 | 1 | 0 | | VI | 2 | 16 | 0 | | SE | 0 | 0 | 17 | | |  | VE | VI | SE | | --- | --- | --- | --- | | VE | 15 | 1 | 0 | | VI | 2 | 16 | 0 | | SE | 0 | 0 | 17 | | |  | VE | VI | SE | | --- | --- | --- | --- | | VE | 15 | 1 | 0 | | VI | 2 | 16 | 0 | | SE | 0 | 0 | 17 | | |  | VE | VI | SE | | --- | --- | --- | --- | | VE | 16 | 0 | 0 | | VI | 0 | 17 | 3 | | SE | 0 | 0 | 14 | |
| Precision | 0.94 | 0.94 | 0.94 | 0.95 |
| Recall | 0.94 | 0.94 | 0.94 | 0.94 |
| Run Time (s) | 0.001 | 0.009 | 0.018 | 0.031 |

**Credit-g Dataset**

|  | Naive | Trauma | Hidden | Gaussian |
| --- | --- | --- | --- | --- |
| Accuracy | 72.2% | 72.5% | 72.5% | 68.1% |
| Confusion | |  | B | G | | --- | --- | --- | | B | 44 | 37 | | G | 56 | 197 | | |  | B | G | | --- | --- | --- | | B | 46 | 38 | | G | 54 | 196 | | |  | B | G | | --- | --- | --- | | B | 31 | 23 | | G | 69 | 211 | | |  | B | G | | --- | --- | --- | | B | 55 | 61 | | G | 44 | 170 | |
| Precision | 0.661 | 0.663 | 0.664 | 0.634 |
| Recall | 0.641 | 0.649 | 0.606 | 0.646 |
| RunTime (s) | 0.015 | 0.369 | 1.683 | 0.693 |

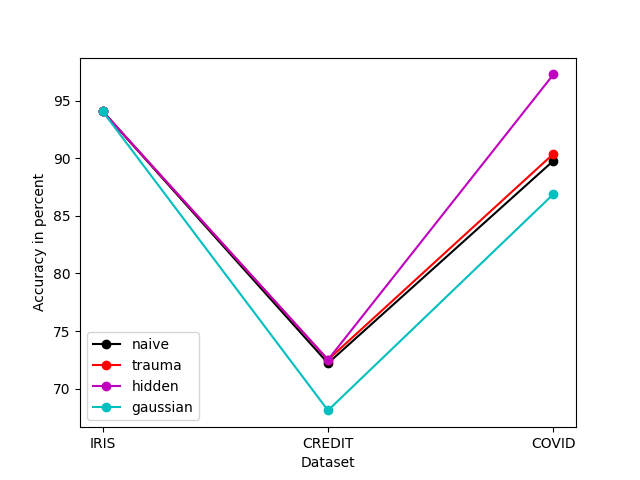
**COVID Dataset**

|  | Naive | Trauma | Hidden | Gaussian |
| --- | --- | --- | --- | --- |
| Accuracy | 89.8% | 90.4% | 97.3% | 86.9% |
| Confusion | |  | Y | N | | --- | --- | --- | | Y | 850 | 3270 | | N | 86 | 28793 | | |  | Y | N | | --- | --- | --- | | Y | 828 | 3047 | | N | 108 | 29016 | | |  | Y | N | | --- | --- | --- | | Y | 192 | 154 | | N | 744 | 31909 | | |  | Y | N | | --- | --- | --- | | Y | 858 | 4240 | | N | 78 | 27823 | |
| Precision | 0.602 | 0.605 | 0.766 | 0.583 |
| Recall | 0.903 | 0.895 | 0.600 | 0.892 |
| Run Time (s) | 2.407 | 20.681 | 369.593 | 142.876 |

**Discussion**

In terms of accuracy, Traumatized Bayes consistently performed better than Naive Bayes. The improvement in accuracy increased as datasets became larger and more complex, as seen in Figure 1 below. This makes sense because Traumatized Bayes takes into account dependencies between attributes, and thus encapsulates more information about the data compared to Naive Bayes. Compared to Hidden Naive Bayes, however, Traumatized Bayes consistently performed worse. This gap also increased as datasets became larger and more complex: for the iris dataset, these two algorithms performed the same, however, for the larger COVID dataset, Hidden Naive Bayes performed almost 7% better than Traumatized Bayes. Gaussian Naive Bayes performed significantly worse in accuracy, most likely due to the discrete attributes that we dropped from datasets.

Fig. 1. Accuracy Plot.



In terms of precision, the four algorithms followed the same trend as accuracy, with Hidden Naive Bayes performing much better than Traumatized Bayes, which performed slightly better than Naive Bayes, and Gaussian Naive Bayes performing significantly worse than the former three. The differences in precision became especially large on the COVID dataset, as it is more complex (see Fig. 2 below).

Surprisingly, in terms of recall, Naive Bayes, Traumatized Bayes, and Gaussian Naive Bayes performed about the same, but performed much better than Hidden Naive Bayes. The differences in recall became especially noticeable on the COVID dataset, as it is more complex (see Fig. 2 below).

This makes sense, since precision and recall are tradeoffs: as precision increases, recall will most likely decrease, as seen by the higher precision but lower recall for Hidden Naive Bayes. Thus, we may conclude that all four algorithms performed approximately the same degree of wellness with regards to the confusion matrix. We do not think any of the algorithms overfit to the training data, since that is an issue mostly caused by small datasets and our datasets were large.

Fig 2. Maco-precision Plot.

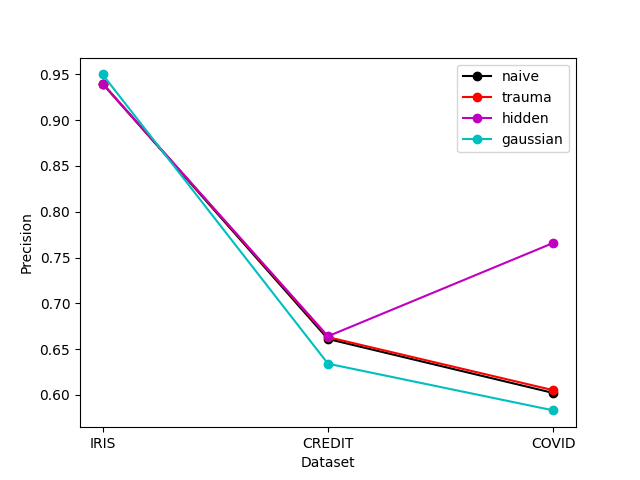
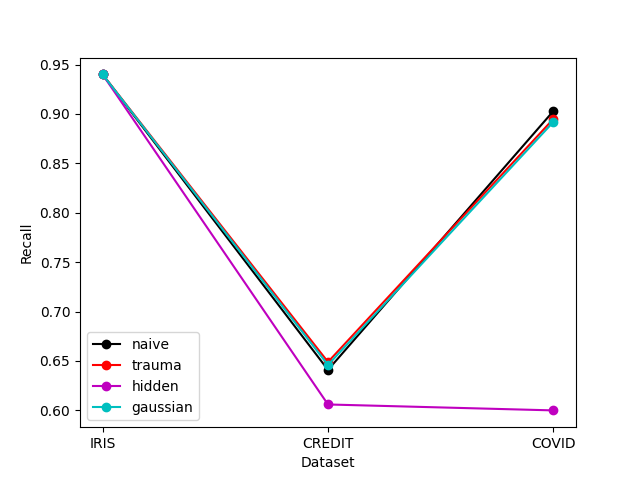
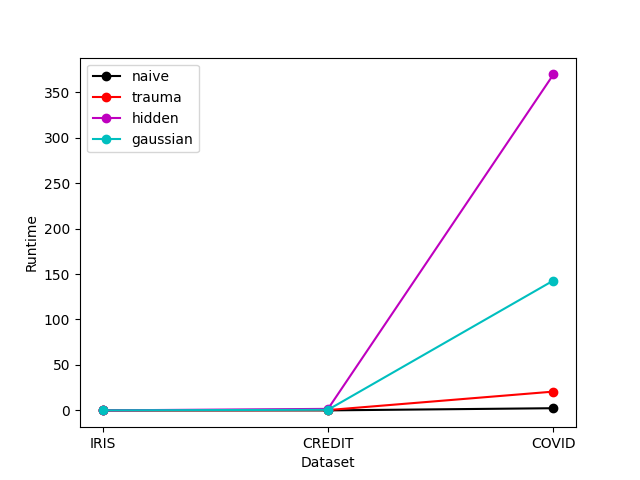


Fig 3. Macro-recall Plot.



Finally, in terms of run time, the order from shortest to longest is as follows: Naive Bayes, Traumatized Bayes, Gaussian Naive Bayes, and Hidden Naive Bayes. Furthermore, the gaps between each pair of algorithms increased as the datasets became more complex. This makes sense, since the algorithms are ordered by increasing complexity. Naive Bayes and Gaussian Naive Bayes increases linearly as more attributes are added, while Traumatized Bayes and Hidden Naive Bayes increase quadratically, but Hidden Naive Bayes has much more complex calculations and testing procedures as compared to Traumatized Bayes. However, there is a discrepancy in the runtime of Gaussian Bayes: it is expected to take shorter than Traumatized Bayes, but due to the slow process of finding p-values (which must be calculated from a library via an imported function), it ends up taking longer than Traumatized Bayes. It is still expected to be faster than Traumatized Bayes, however, once the dataset reaches a certain complexity.

Fig 4. Runtime Plot.



To summarize, our algorithm, Traumatized Bayes, is an improvement of Naive Bayes in terms of accuracy. It is a good alternative to Naive Bayes that can improve the accuracy of the model while still maintaining a reasonable speed, as compared to complex algorithms like Hidden Naive Bayes that require extremely long runtimes for large amounts of data. Traumatized Bayes is also more versatile as compared to Gaussian Naive Bayes, as it works with discrete attributes and continuous attributes (that can be discretized), while GNB only works with continuous attributes and must drop discrete ones. Thus, Traumatized Bayes is a viable alternative to Naive Bayes that can be used to solve classification problems.

**Conclusion and Future Work**

Naive Bayes assumes independence between attributes, and has a high accuracy when this assumption of independence is met. When the attributes are correlated, its accuracy suffers, so we created Traumatized Bayes, a classification algorithm that takes into consideration the correlations between pairs of attributes, but still assumes independence between triples of attributes. Traumatized Bayes had higher accuracy than traditional Naive Bayes in all tested datasets, which is logical since it incorporates all the information that Naive Bayes does (distribution of attribute values across classes) plus new information (correlation between classes). It does this at a slower speed than Naive Bayes, however, which also makes sense since it has to calculate correlation coefficients and incorporate that into its final prediction. Traumatized Bayes outperformed Gaussian Naive Bayes, however, likely because Gaussian Naive Bayes assumes that all quantitative variables follow a Gaussian (normal) distribution. Many variables, such as whether someone spent time in an ICU, have skewed distributions, which impact the performance of Gaussian Naive Bayes. When the assumption of normality is met (such as in iris dataset), Gaussian Naive Bayes shows performance comparative to the other algorithms. In the future, if we had more time, we would like to implement an adaptation of Gaussian Naive Bayes that is capable of detecting geometric and binomial distributions and calculating probabilities accordingly. Gaussian Naive Bayes was also the second slowest algorithm (only faster than Hidden Naive Bayes), which may be because of the imported library of information about the normal distribution. Traumatized Bayes has worse accuracy than Hidden Naive Bayes, but Hidden Naive Bayes is much slower (up to 18x slower). As such, we conclude that Traumatized Bayes is better for general use, but Hidden Naive Bayes may be better when the computational budget is high.

**Contributions**

**Introduction:** Lilian Zhu

**Related work:**

* Both partners read all articles cited under the bibliography
* Writeup written by Lilian Zhu

**Dataset preprocessing:** Lilian Zhu

**Methods:**

* **Naive Bayes:** Writeup by Isabella Zhu and Lilian Zhu, implementation by Isabella Zhu
* **Traumatized Bayes:** Isabella Zhu
* **Hidden Naive Bayes:** Isabella Zhu
* **Gaussian Naive Bayes:** Lilian Zhu

**Experiment, Results, and Discussion:**

* Experiments and Results conducted by the same person who implemented them (listed above)
* **Discussion:** Isabella Zhu

**Conclusion/Future Work:** Lilian Zhu

**This section:** Isabella Zhu and Lilian Zhu

**References:** Isabella Zhu and Lilian Zhu

**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>.

1. Fisher, R A. “ Iris Data Set.” *UCI Machine Learning Repository: Iris Data Set*, 1936, <https://archive.ics.uci.edu/ml/datasets/iris>.
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.

1. For confusion matrices, predicted values are along the vertical, while actual values are along the horizontal. [↑](#footnote-ref-0)