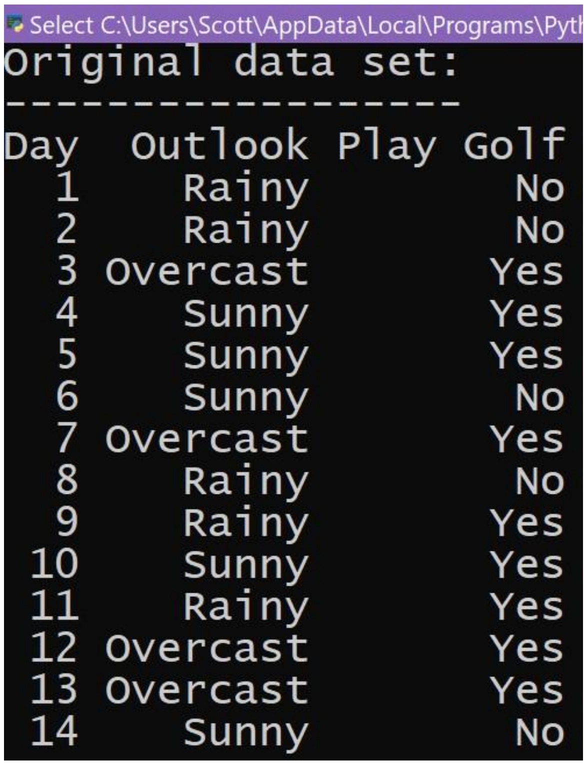
Option #1: Naive Bayes Classifier

Scott Miner

Colorado State University – Global Campus

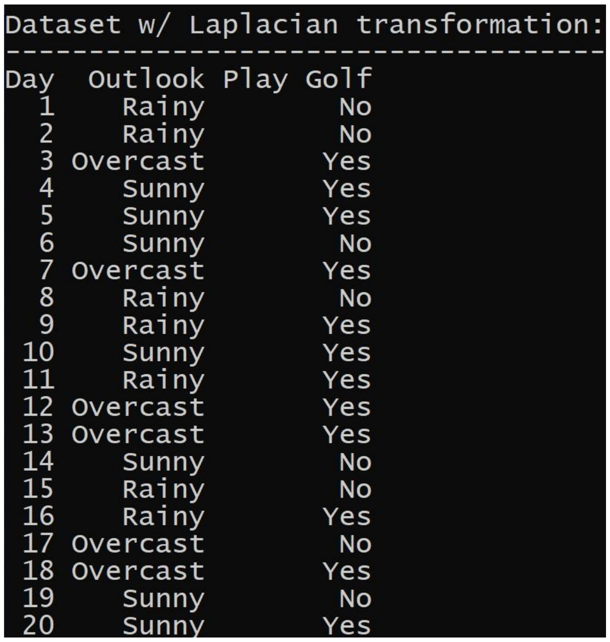
Abstract

This paper presents a simple Naïve Bayes classifier implemented in Python using the scikit-learn package, designed to predict the likelihood of playing golf on a given day based on the weather outlook. The classifier addresses the zero-probability problem by employing the Laplacian correction technique. It then converts the dataset into a frequency table and creates a "likelihood" table by computing the relevant probabilities. Subsequently, the classifier calculates the posterior probabilities for each class. To ensure accuracy, the paper verifies the output of the classifier using manual calculations in Excel.



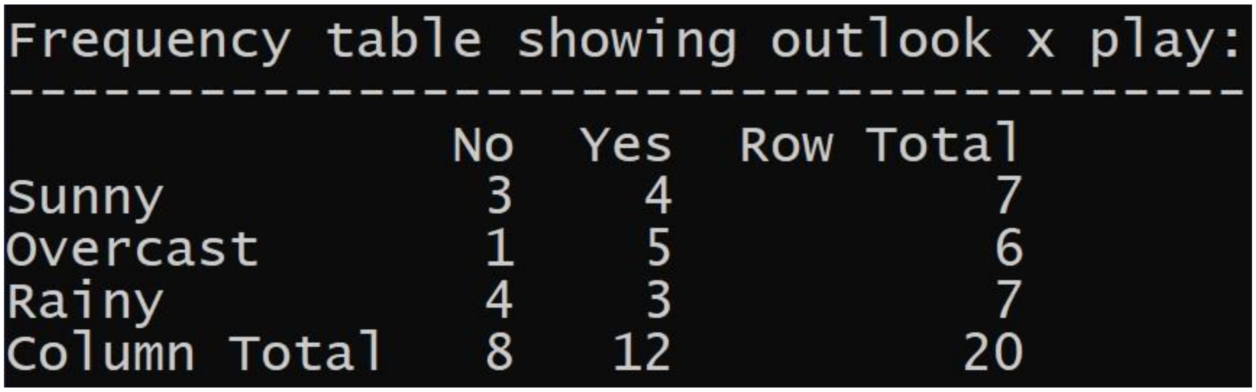
```
Select C:\Users\Scott\AppData\Local\Programs\Pyth
Original data set:
------------------
Day  Outlook Play Golf
  1    Rainy        No
  2    Rainy        No
  3 Overcast       Yes
  4    Sunny       Yes
  5    Sunny       Yes
  6    Sunny        No
  7 Overcast       Yes
  8    Rainy        No
  9    Rainy       Yes
 10    Sunny       Yes
 11    Rainy       Yes
 12 Overcast       Yes
 13 Overcast       Yes
 14    Sunny        No
```

Figure 1. The dataset before Laplace correction



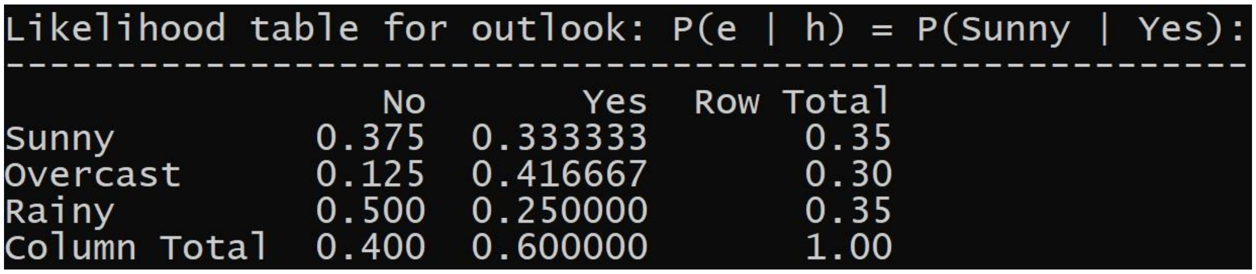
```
Dataset w/ Laplacian transformation:
------------------------------------
Day  Outlook Play Golf
  1    Rainy        No
  2    Rainy        No
  3 Overcast       Yes
  4    Sunny       Yes
  5    Sunny       Yes
  6    Sunny        No
  7 Overcast       Yes
  8    Rainy        No
  9    Rainy       Yes
 10    Sunny       Yes
 11    Rainy       Yes
 12 Overcast       Yes
 13 Overcast       Yes
 14    Sunny        No
 15    Rainy        No
 16    Rainy       Yes
 17 Overcast        No
 18 Overcast       Yes
 19    Sunny        No
 20    Sunny       Yes
```

Figure 2. The dataset after the Laplace transformation



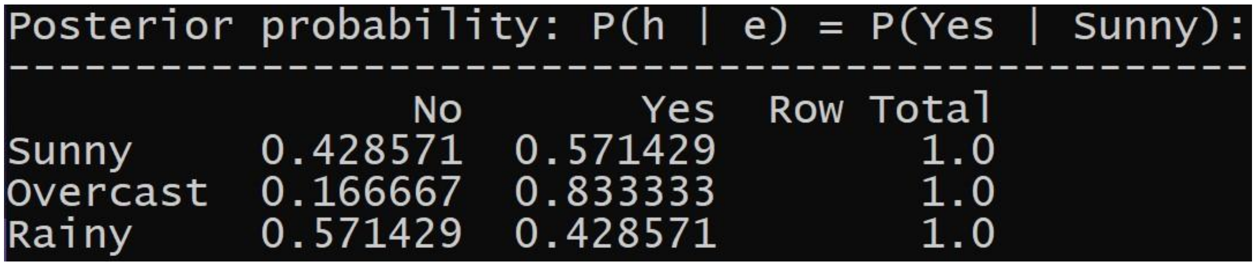
```
Frequency table showing outlook x play:
---------------------------------------
                No  Yes  Row Total
Sunny            3    4          7
Overcast         1    5          6
Rainy            4    3          7
Column Total     8   12         20
```

Figure 3. Frequency table showing the crosstab of the feature and target variables



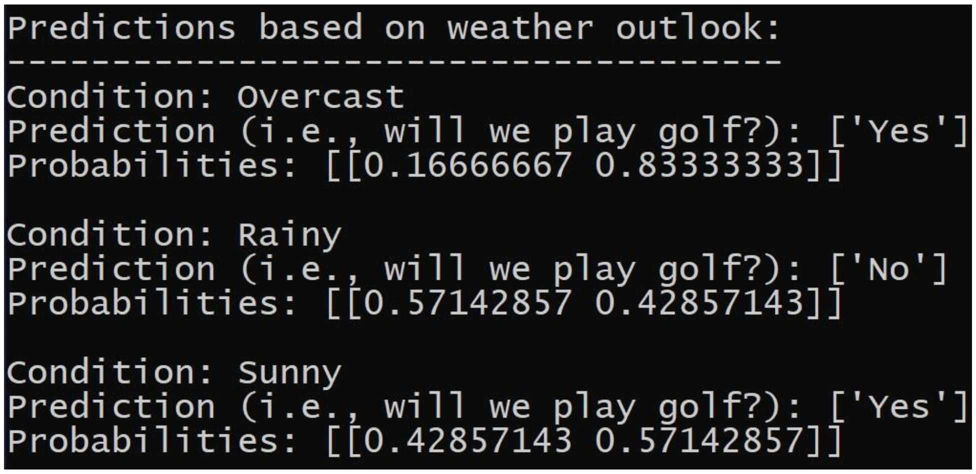
```
Likelihood table for outlook: P(e | h) = P(Sunny | Yes):
--------------------------------------------------------
                 No       Yes  Row Total
Sunny         0.375  0.333333       0.35
Overcast      0.125  0.416667       0.30
Rainy         0.500  0.250000       0.35
Column Total  0.400  0.600000       1.00
```

Figure 4. Likelihood table of the feature and target variables



```
Posterior probability: P(h | e) = P(Yes | Sunny):
-------------------------------------------------
                No       Yes  Row Total
Sunny     0.428571  0.571429        1.0
Overcast  0.166667  0.833333        1.0
Rainy     0.571429  0.428571        1.0
```

Figure 5. Posterior probability calculations

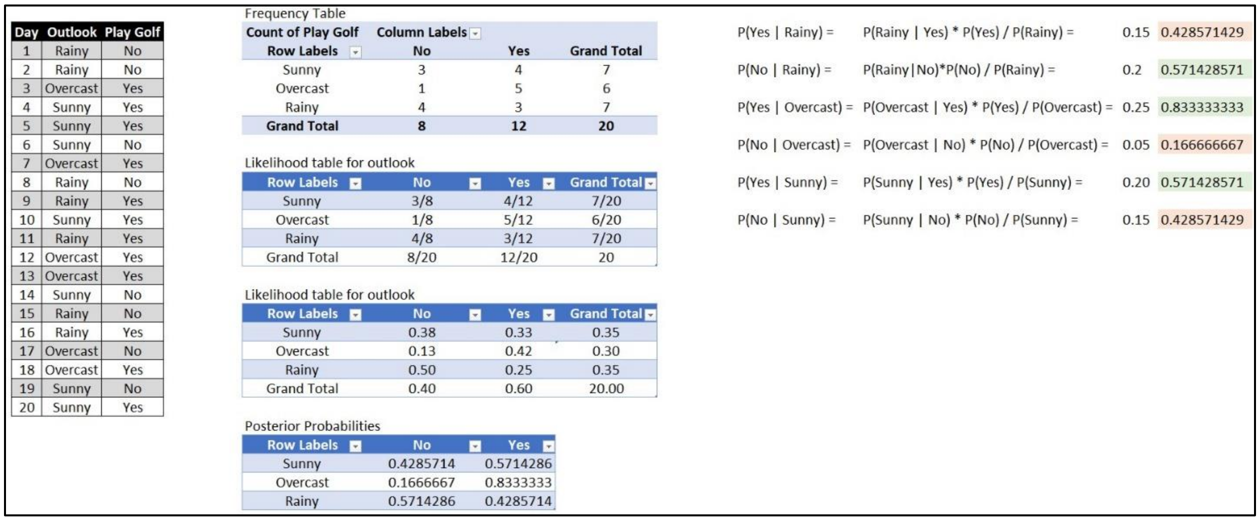


```
Predictions based on weather outlook:
-------------------------------------
Condition: Overcast
Prediction (i.e., will we play golf?): ['Yes']
Probabilities: [[0.16666667 0.83333333]]

Condition: Rainy
Prediction (i.e., will we play golf?): ['No']
Probabilities: [[0.57142857 0.42857143]]

Condition: Sunny
Prediction (i.e., will we play golf?): ['Yes']
Probabilities: [[0.42857143 0.57142857]]
```

Figure 6. Predictions produced by the Naïve Bayes classifier based on its input data



Day	Outlook	Play Golf
1	Rainy	No
2	Rainy	No
3	Overcast	Yes
4	Sunny	Yes
5	Sunny	Yes
6	Sunny	No
7	Overcast	Yes
8	Rainy	No
9	Rainy	Yes
10	Sunny	Yes
11	Rainy	Yes
12	Overcast	Yes
13	Overcast	Yes
14	Sunny	No
15	Rainy	No
16	Rainy	Yes
17	Overcast	No
18	Overcast	Yes
19	Sunny	No
20	Sunny	Yes

Frequency Table

Count of Play Golf	Column Labels		
Row Labels	No	Yes	Grand Total
Sunny	3	4	7
Overcast	1	5	6
Rainy	4	3	7
Grand Total	**8**	**12**	**20**

Likelihood table for outlook

Row Labels	No	Yes	Grand Total
Sunny	3/8	4/12	7/20
Overcast	1/8	5/12	6/20
Rainy	4/8	3/12	7/20
Grand Total	8/20	12/20	20

Likelihood table for outlook

Row Labels	No	Yes	Grand Total
Sunny	0.38	0.33	0.35
Overcast	0.13	0.42	0.30
Rainy	0.50	0.25	0.35
Grand Total	0.40	0.60	20.00

Posterior Probabilities

Row Labels	No	Yes
Sunny	0.4285714	0.5714286
Overcast	0.1666667	0.8333333
Rainy	0.5714286	0.4285714

P(Yes | Rainy) =	P(Rainy | Yes) * P(Yes) / P(Rainy) =	0.15	0.428571429
P(No | Rainy) =	P(Rainy|No)*P(No) / P(Rainy) =	0.2	0.571428571
P(Yes | Overcast) =	P(Overcast | Yes) * P(Yes) / P(Overcast) =	0.25	0.833333333
P(No | Overcast) =	P(Overcast | No) * P(No) / P(Overcast) =	0.05	0.166666667
P(Yes | Sunny) =	P(Sunny | Yes) * P(Yes) / P(Sunny) =	0.20	0.571428571
P(No | Sunny) =	P(Sunny | No) * P(No) / P(Sunny) =	0.15	0.428571429

Figure 7. The output of the Python script verified in Excel

Table of Contents

Option #1: Naive Bayes Classifier

Sharda *et al.* (2020) define Naïve Bayes as a machine-learning technique derived from Bayes' theorem. Bayes' theorem is defined as follows: $Posterior = \frac{Likelihood \times Prior}{Evidence}$. We define the posterior as $P(H \mid E)$, the probability of a hypothesis, H, given some evidence, E. Likewise, we define the likelihood as $P(E \mid H)$, the probability of the evidence given some hypothesis. We denote the prior as $P(H)$, the probability of the hypothesis. Finally, we denote the probability of the evidence as $P(E)$.

Problem Definition

In this paper, we examine a simple problem using a sample dataset containing 14 observations and three features: (a) a sequential number representing the day each observation was recorded, (b) the categorical variable containing the weather outlook for that day (i.e., raining, overcast, or sunny), and (c) the target variable, indicating whether we played golf on that day (i.e., yes or no). The problem we are trying to solve is predicting whether we will play golf on any given day, given that day's weather outlook, which is the evidence in this scenario. Predicting whether we will play golf represents the hypothesis.

Addressing the Zero-Probability Problem

However, given this dataset and the nature of Naïve Bayes classifiers, the possibility of something known as the zero-probability problem arises. The zero-probability problem occurs when there is no condition for a given scenario in our training data, which causes the probability of that scenario to be reduced to zero if it occurs in our test data. One technique to overcome this problem is Laplacian correction, a smoothing technique (Navlani, 2018). Cherian (2017) describes smoothing techniques as those that attempt to capture important patterns in data while avoiding “fine-scale structures/rapid phenomena” (p. 71). Additionally, Spiegel (1965) defines the Laplace transform as an essential component of scientists’ mathematical background and an effective means to solve many problems arising in various fields of science and engineering.

Laplacian Correction

To perform a Laplace transformation of a dataset, we add records for all the various scenarios that could occur within our training data (Navlani, 2018). For example, in Figure 1, the training dataset does not contain any record where we do not play golf when it is overcast. Therefore, we add a record for each possible scenario that could exist within our training data (Pieter Abbeel, 2012). Figure 2 shows the dataset after applying the Laplacian correction. We have added a record for each possible combination of scenarios that may exist in the training data, adding a total of six records, which increases the record count from 14 to 20 and eliminates the zero-probability problem.

Creating the Frequency and Likelihood Tables

The next step is to create a frequency table using these updated counts. Figure 3 shows the frequency table that the Python script creates. The frequencies are simply the number of times a condition and outcome occur together. To create a likelihood table, we divide each frequency by the total number of frequencies in each category of the target variable (i.e., yes or no). Figure 4 shows the likelihood of the weather outlook, given that we played golf on that day (i.e., $P(E \mid H)$). Additionally, the column and row totals in this table allow us to calculate the priors: (a) $P(H)$ (i.e., the likelihood that we played golf) and (b) $P(E)$ (i.e., the likelihood of the weather outlook).

Calculating the Posterior Probabilities

With the likelihood table and priors determined, we can plug these values into Bayes' theorem to compute the posterior probabilities (i.e., "What is the probability that we played golf given a certain weather outlook?"). Figure 5 displays the posterior probabilities for each category of the predictor and target variables (i.e., *P(H | E) = P(Yes | Rainy)*). The model's final prediction is determined by whether the probability is higher for the "Yes" or "No" class. For instance, the model predicts that we do not play golf when the weather is rainy because *P(No | Rainy)* > *P(Yes | Rainy)* (i.e., 0.57 > 0.43). Figure 6 presents the predictions output by the model for each category of the outlook variable. When the weather is overcast and sunny, the model predicts we will play golf. However, when the weather is rainy, the model, hopefully correctly, predicts that we will not play golf.

Notice that the probabilities that the model outputs match those in the posterior probability table. Also, note that the call to the *categoricalNB()* constructor in line 96 of the accompanying NaiveBayes.py file instantiates the classifier and passes the *alpha=0* argument to bypass the Laplacian correction that is implemented by default since we handled the Laplacian transformation manually as described above (*Sklearn.Naive_bayes.CategoricalNB — Scikit-Learn 0.24.2 Documentation, n.d.*). The *categoricalNB()* classifier is the correct Naïve Bayes classifier for categorical features, as is the case with our weather outlook variable (*Sklearn.Naive_bayes.CategoricalNB — Scikit-Learn 0.24.2 Documentation, n.d.*).

Verification of Results

Figure 7 demonstrates the output verification using Excel to manually calculate the frequency tables, likelihood tables, and posterior probabilities. The upper right-hand corner of the image displays the probabilities output in Excel, which match those output by the Naïve Bayes classifier created using the scikit-learn package in Python. The higher of each condition's probabilities are highlighted in green, while the lower probability is highlighted in orange. The output corresponds to that created by the Naïve Bayes classifier in the Python script. To run the accompanying Python script, NaiveBayes.py, one only needs to call the script from the Python interpreter since the Python script includes the sample dataset.

Handling Multiple Features

In cases with multiple feature variables, once the posterior probability tables are created for each feature, we can calculate the probabilities of multiple features by multiplying them using the following formula: $P(Yes|E) = P(E_1|Yes) \times P(E_2|Yes) \ldots \times P(E_n|Yes) \times P(Yes),$ where *n* represents the number of features in the dataset and *E* represents the evidence. We calculate the same probability for the "No" class and standardize the variables by dividing each over the sum of the two, allowing us to compute the posterior probabilities when multiple features exist (i.e., temperature and humidity). The simple example this paper describes only uses a single feature variable (i.e., weather outlook) for demonstration purposes.

Conclusion

In conclusion, this paper has presented a clear and concise explanation of the Naïve Bayes classifier, a machine learning technique derived from Bayes' theorem. By utilizing the scikit-learn package in Python, we demonstrated the application of this classifier to a simple problem: predicting whether golf would be played given specific weather conditions. The step-by-step process, including the Laplacian correction to address the zero-probability problem, creation of frequency and likelihood tables, and calculation of posterior probabilities, offered a comprehensive understanding of the method and its applications.

The paper also highlighted the importance of the Laplace transform in overcoming the zero-probability problem, allowing for more accurate and reliable predictions. The classifier's effectiveness and efficiency were demonstrated through the Python implementation and the manual verification of results using Excel, showcasing its practicality for real-world applications.

In summary, the Naïve Bayes classifier has proven to be a powerful and straightforward machine learning technique for solving classification problems. The detailed explanation, Python implementation, and Excel verification provided in this paper serve as a solid foundation for further exploration and utilization of the Naïve Bayes classifier in more complex problems involving multiple features and larger datasets. This study contributes to the understanding of this versatile technique and its potential applications in various fields.

References

Cherian, V. (2017). *Heart Disease Prediction Using Naïve Bayes Algorithm and Laplace Smoothing Technique*. *5*(2), 6.

Naive Bayesian. (n.d.). Retrieved July 18, 2021, from https://www.saedsayad.com/naive_bayesian.htm

Navlani, A. (2018, December 4). *Sklearn Naive Bayes Classifier Python: Gaussian Naive Bayes Scikit-Learn Tutorial*. DataCamp Community. https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn

Pieter Abbeel. (2012, May 11). *Laplace smoothing*. https://www.youtube.com/watch?v=gCI-ZC7irbY&t=3s

Sharda, R., Delen, D., & Turban, E. (2020). *Analytics, data science, & artificial intelligence* (Eleventh edition). Pearson.

Sklearn.naive_bayes.CategoricalNB — scikit-learn 0.24.2 documentation. (n.d.). Retrieved July 18, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.CategoricalNB.html

Spiegel, M. R. (1965). *Laplace transforms*. McGraw-Hill New York.