Naïve Bayes Algorithm Implementation from scratch

Meritxell Arbiol

23-06-2022

Abstract

In this project we are going to go into detail on how the Naive Bayes algorithm works and we are going to implement it from scratch in Python. In addition we are going to compare the result obtained with the code implemented by us with the existing code of the sklearn library to see if the result is similar in both.

1 Introduction to Naive Bayes Algorithm

Naive Bayes is one of the simplest classification algorithms, not only the simplest but also one of the fastest computationally speaking and most accurate compared to others.

Naive Bayes is a machine learning algorithm that is based on the Bayes theorem, and is used in a wide variety of classification tasks. It is important to first understand Bayes theorem to understand all the logic behind the classifier and then to be able to implement it in python from scratch [1].

2 Bayes Theorem

The Bayes theorem is simply a mathematical formula that calculates the conditional probability, which is the probability that an event will happen given another event that has already occurred [3].

The formula of the Bayes theorem is as follows:

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

What the formula (1) tells us is each time A happens given that B happens, it is written P(A|B). So, when we know every when B happens given A (P(B|A)) and every when A happens regardless of B (P(A)), and every when B happens regardless of A (P(B)), then we have all the information needed to calculate P(A|B). So it is a simple and easy formula to calculate the Bayes theorem as long as we have information about the other probabilities [4].

Assumptions we make:

We assume that each feature makes an independent and equal contribution to the outcome, in other words, there are no features that have a greater weight.

Even if we assume that, we have to keep in mind that normally in the real world there is always a relationship between the features, so they are not 100% independent, so the assumption made of independence is not correct but still works well in practice.

To understand how this theorem works, let's use an example from a sample of 5 lines of the famous Iris dataset, where we have a series of characteristics (*SepalLengthCm*, *SepalWidthCm*, *PetalLengthCm*, *PetalWidthCm*), with which we are going to predict our target *Species*:

```
SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
5.1,3.5,1.4,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
5.7,2.8,4.1,1.3,Iris-versicolor
6.9,3.1,5.4,2.1,Iris-virginica
```

From the above dataset we will classify which species it is given the four characteristics (each column represents one characteristic, and each line one more sample for).

Let's rewrite Bayes' formula as follows:

$$P(y|X) = P(X|y)P(y)/P(X) \tag{2}$$

Where the variable y is our target (Species), and the variable X the characteristics y la variable X = (x_1, x_2, x_3, ..., x_n) and they represent each of the features, in this case *SepalLengthCm*, *SepalWidthCm*, *PetalLengthCm*, *PetalWidthCm*.

So taking this into account, the formula would be as follows:

$$P(y|x_1, ..., x_n) = P(x_1|y)P(x_2|y)...P(x_n|y)P(y)/(P(x_1)P(x_2)...P(x_n)) \tag{3}$$

The denominator is the same for all incoming data values, so that the denominator can be removed and proportionality can be injected.

$$P(y|x_1, ..., x_n) \propto P(y)\prod_i P(x_i|y) \tag{4}$$

Thus, to calculate P(y|X) we do it following these steps: Create a Frequency table per attribute Mold the frequency tables to Likelihood Tables Use the Naive Bayesian equation to calculate the posterior probability (P(y|X)) per each class.

- Create a Frequency table per attribute.
- Mold the frequency tables to Likelihood Tables.
- Use the Naive Bayesian equation to calculate the posterior probability (P(y—X)) per each class.

3 Zero-Frequency Problem

- **Problem:** Naive Bayes has a major weakness, called The Zero-Frequency Problem [1]. This problem occurs when there are no occurrences of one of the classes in our data. When this happens, the estimated frequency probability is 0, so when all probabilities are multiplied we will get 0.
- **Solution:** Add one to the count for each value-class combination when that attribute does not occur for all classes.

4 Types of Naive Bayes Classifiers

4.1 Multinomial Naïve Bayes Classifier

This is a model that is widely used for document classification, where the characteristics of our data represent the number of times an event has been generated by the multinomial distribution.

4.2 Bernoulli Naïve Bayes Classifier

In this type of classifier, the features are independent and Boolean. As with the multinomial classifier, this model is widely used for document classification tasks, where the output is binary, indicating for example whether a certain word appears or not.

4.3 Gaussian Naïve Bayes Classifier

We assume a normal (or Gaussian) probability distribution. The likelihood of the features is assumed to be:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{5}$$

5 Advantages and Disadvantages

Advantages:

- It is an easy and fast algorithm to make predictions for both binary and multiclass classes.
- Assuming independence between variables, the classifier achieves better results compared to other models such as logistic regression and also requires less data to train the algorithm.
- It works well for both categorical and numerical features.

Disadvantages:

- This problem is the one we have already discussed in the "Zero Frequency Problem" section. If a variable has a category that has not been observed in the data used to train the algorithm, the model will assign a probability of 0, and it will be impossible to make predictions.
- To use Naive Bayes we assume that the features are independent of each other, but as we mentioned at the beginning, in the real world this does not happen, since always there is dependence between them.

6 Applications of Naive Bayes

- Used for sentiment analysis in the system/filter spam.
- Widely used for real-time forecasting because it is very fast.
- It is very much used to make multiclass predictions.
- It is also widely used to create system recommendations by predicting what the user might be interested in and showing it to them.

7 Our implementation

For our implementation, we have structured the code mainly in two parts: Fit and Predict [2].

To do this we have created a class called NaiveBayes where we have defined several functions inside: fit, predict, class_prob and density.

7.1 Fit function

We have to calculate the statistics of each class according to the characteristics of belonging to one class or another. So we explore a bit the data we have by calculating the variance, mean and prior.

How is the Prior calculated?
It is simply counting the number of samples we have of a specific class and dividing it by the total number of samples of our data.

7.2 Predict function

Prediction consists of calculating the probability of data belonging to a certain class. In our case we assume that the data have Gaussian distribution. In this way, iteratively from the mean, density and prior using the Gaussian density formula seen above (equation 5), we calculate the posterior for each of the samples separately, which is the class for which the sample has a higher probability.

7.3 Accuracy function

Now only the last part is missing, the evaluation of the model using the test data. For this we compare the predictions made with the test data, and compare them with the actual test values. We have also looked at the confusion matrix to find out exactly how many misclassified values there are and of what kind.

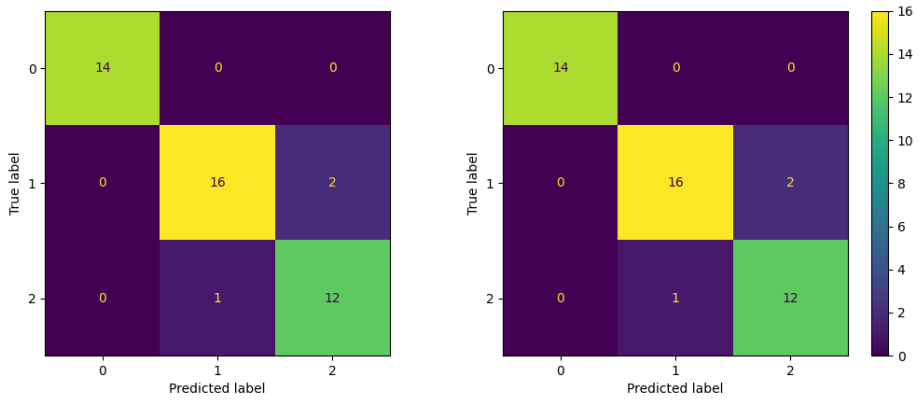
8 Results

We have checked the results obtained with our implementation and compared them with the already defined function of the sklearn library, we have used the famous Iris data.

Accuracy of our results	0.93%
Accuracy of GaussianNB() from sklearn	0.93%

As we can see, the results are exactly the same in both scenarios (done with our implementation and using the *sklearn* library calling the *GaussianNB()* function).

With both methods we achieve exactly the same accuracy. Furthermore, if we look at the confusion matrix (Figure 1a and 1b) of each of the two methods, we see that they are exactly the same, which means that the same samples that have not been well classified in one method, are the same misclassified in the other, so both functions work in exactly the same way.



(a) Our implementation (b) GaussianNB() from sklearn library

Figure 1: Comparison between confussion matrix

9 Conclusions

After all the in-depth analysis and research done on how the Naive Bayes classifier works we have seen that it is very simple to use and to be implemented. This is a clear example that sometimes the simplest things are the ones that work best.

It is a classifier that is very fast, the level of complexity in the algorithm is very low, with understanding the Bayes theorem we can understand quite well how the classifier works and how the probabilities of each of the samples belonging to one class or another are calculated.

We can also see that the implementation we have done gives us exactly the same results as the sklearn library. This may be because we do not use hyperparameters so there are no small details that may be different, and in the end it is just a formula.

10 Appendix

Here is written our class called NaiveBayes:

```
class NaiveBayes:
    def fit(self, X, y):
        self.num_samples = X.shape[0]
        self.num_features = X.shape[1]
        self.num_classes = len(np.unique(y))
        self.mean = np.zeros((self.num_classes, self.num_features))
```

```
        self.var = np.zeros((self.num_classes, self.num_features))
        self.priors = np.zeros(self.num_classes)
        for _class in range(self.num_classes):
            Xclass = X[y == _class]
            self.mean[_class, :] = np.mean(Xclass, axis=0)
            self.var[_class, :] = np.var(Xclass, axis=0)
            self.priors[_class] = Xclass.shape[0] / self.num_samples

    def prediction(self, X):
        y_pred = [self.class_prob(x) for x in X]
        return np.array(y_pred)

    def class_prob(self, x):
        post_list = list()
        for _class in range(self.num_classes):
            mean = self.mean[_class]
            var = self.var[_class]
            prior = np.log(self.priors[_class])
            post = np.sum(np.log(self.gaussian_density(x, mean, var)))
            post = prior + post
            post_list.append(post)
        return np.argmax(post_list)

    def gaussian_density(self, x, mean, var):
        density = (1 / np.sqrt(var * 2 * np.pi)) *
        (np.exp(-0.5 * ((x - mean) ** 2 / var)))
        return density
```

References

[1] "Naïve Bayes Algorithm: Everything You Need to Know", https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html, 2022.

[2] "Implementing Naïve Bayes Classification from Scratch with Python", https://blog.devgenius.io/implementing-na%C3%AFve-bayes-classification-from-scratch-with-python-badd5a9be9c3, 2022.

[3] "Bayes Theorem", https://iq.opengenus.org/gaussian-naive-bayes/

[4] "Bayes' Theorem", https://www.investopedia.com/terms/b/bayes-theorem.asp