Naïve Bayes Classification

1. Introduction
Naive Bayes is a probabilistic machine learning technique that can solve a variety of classification issues. Spam filtering, document classification, sentiment prediction, and other common applications are examples. It is based on Rev. Thomas Bayes' (1702-61) works and thus bears his name.

But why is it termed 'Naive'?

The phrase "naive" refers to the idea that the model's properties are self-contained. That is, changing the value of a single feature has no direct effect on the values of the other features used in the algorithm.

Alright. Naive Bayes appears to be a straightforward but effective algorithm. However, what makes it so popular?

This is because of NB's colossal edge. Due to the probabilistic nature of the model, the algorithm is simple to implement and the predictions may be made rapidly.
Rapid response in real time. As a result, it is easily scalable and is frequently used in real-world applications (apps) that must respond immediately to user requests.

However, before digging into Naive Bayes, it is necessary to comprehend the concepts of 'Conditional Probability' and the 'Bayes Rule.'

Additionally, by the conclusion of this tutorial, you will have learned:

A detailed explanation of the Naive Bayes Classifier's operation.
Gaussian Naive Bayes: What is it, when is it used, and how does it work?
How to implement this in Python How to improve your Naive Bayes models

Cool? Let's get started.

2. What is Conditional Probability?

Let's begin with the fundamentals by defining conditional probability.

Example of a Coin Toss and a Fair Dice

When a fair coin is flipped, there is an equal chance of receiving heads or tails. Thus, the probability of obtaining heads is 50%.

Similarly, what is the likelihood of getting a 1 when rolling a six-sided die? Assuming the dice are evenly distributed, the probability of 1/6 equals 0.166. Okay, one final illustration with playing cards.

Illustration of Playing Cards

Can you determine the likelihood of drawing a queen if you pick a card from the deck in light of the fact that the card is a spade?

To be sure, I have already specified that the card must be a spade. As a result, the numerator (eligible population) is 13 rather than 52. And, because there is only one queen in spades, the likelihood that it is a queen is 1/13 = 0.077.

This is a textbook illustration of conditional probability. As a result, when you speak of the conditional probability of A in the present situation B, it denotes the likelihood of A occurring in the given situation that B has already taken place.

The conditional probability of A given B can be calculated mathematically as: P(A|B).

P(A AND B) = P(A AND B) / P (B)

Example of a School

Consider the following slightly complex case. Consider a school that has a total enrollment of 100 students. These 100 individuals can be classified as either 'Students' or 'Teachers', or as a population of 'Males' and 'Females.

What is the conditional probability that a certain member of the school is a 'Teacher' based on the fact that he is a 'Man'?

This may be calculated intuitively by filtering the subpopulation of 60 males and concentrating on the 12 (male) teachers. As a result, the necessary conditional probability P(Teacher | Male) equals 12 / 60 = 0.2.

This is the intersection between Teacher (A) and Male (B), split by Male (B). Similarly, the conditional probability of B in the presence of A can be calculated.

These two notations can be used to derive the Bayes Rule that we utilise for Naive Bayes.

3. The Bayes Rule

The Bayes Rule is a method for determining P(Y|X) from P(X|Y), which is known from the training dataset.

This is accomplished by substituting feature X and response Y for A and B in the preceding formula.

In the case of test or scoring data observations, X is known but Y is uncertain. And you want to compute the likelihood of Y occurring given that X has already occurred for each row in the test dataset.

What if Y has more than two categories? We calculate the chance of each Y class and award the highest probability.

4. The Naive Bayes

The Bayes Rule defines the likelihood of Y given X. However, in real-world situations, numerous X variables are frequently present.

When the features are distinct, we can use what is known as Naive Bayes to extend the Bayes Rule.

It is referred to as 'Naive' due to the naive assumption that the X's are self-contained. Whatever its name, this is a really effective formula.

The left-hand side (LHS) of the equation, in technical parlance, is referred to as the posterior probability or simply the posterior.

Numerator of the RHS contains two terms.

The first is referred to as the ‘Likelihood of Evidence.' It is just the conditional probability of each X being of special class 'c' if Y is of that class.

Because all X's are believed to be independent, you can simply multiply the 'likelihoods' of all X's to obtain the 'Probability of likelihood of evidence'. This is determined by filtering records with Y=c from the training dataset.

The second term is referred to as the prior, and it refers to the overall likelihood of Y=c, where c denotes a class of Y. Prior = count(Y=c) / n .Records in the simplest words.

A practical demonstration is preferable to an hour of theory. Therefore, let us examine one.

5. Naive Bayes Example by Hand
Say you have 1000 fruits which could be either ‘banana’, ‘orange’ or ‘other’.
These are the 3 possible classes of the Y variable.

We have data for the following X variables, all of which are binary (1 or 0).

● Long
● Sweet
● Yellow

The first few rows of the training dataset look like this:

Fruit Long (x1) Sweet (x2) Yellow (x3)

Orange 0 1 0

Banana 1 0 1

Banana 1 1 1

Other 1 1 0

.. ..

For the sake of computing the probabilities, let's aggregate the training data to form a counts table like this.

So the objective of the classifier is to predict if a given fruit is a 'Banana' or 'Orange' or 'Other' when only the 3 features (long, sweet and yellow) are known.

Let's say you are given a fruit that is: Long, Sweet and Yellow, can you predict What fruit is it?

This is the same of predicting the Y when only the X variables in testing data are known. Let's solve it by hand using Naive Bayes.

The idea is to compute the 3 probabilities, that is the probability of the fruit being a banana, orange or other. Whichever fruit type gets the highest probability wins.

All the information to calculate these probabilities is present in the above tabulation.

Step 1: Compute the 'Prior' probabilities for each of the class of fruits.

That is, the proportion of each fruit class out of all the fruits from the population. You can provide the 'Priors' from prior information about the population. Otherwise, it can be computed from the training data.

For this case, let's compute the training data. Out of 1000 records in training data, you have 500 Bananas, 300 Oranges and 200 Others. So the respective priors are 0.5, 0.3 and 0.2.

P(Y=Banana) = 500 / 1000 = 0.50

P(Y=Orange) = 300 / 1000 = 0.30

P(Y=Other) = 200 / 1000 = 0.20

Step 2: Compute the probability of evidence that goes in the denominator.

This is nothing but the product of P of Xs for all X. This is an optional step because the denominator is the same for all the classes and so will not affect the probabilities.

P(x1=Long) = 500 / 1000 = 0.50

P(x2=Sweet) = 650 / 1000 = 0.65

P(x3=Yellow) = 800 / 1000 = 0.80

Step 3: Compute the probability of likelihood of evidences that goes in the numerator.

It is the product of conditional probabilities of the 3 features. If you refer back to the formula, it says P(X1 |Y=k). Here X1 is 'Long' and k is 'Banana'. That means the probability the fruit is 'Long' given that it is a Banana. In the above table, you have 500 Bananas. Out of that 400 is long. So, P(Long | Banana) = 400/500 = 0.8.

Here, I have done it for Banana alone.

Probability of Likelihood for Banana

P(x1=Long | Y=Banana) = 400 / 500 = 0.80

P(x2=Sweet | Y=Banana) = 350 / 500 = 0.70

P(x3=Yellow | Y=Banana) = 450 / 500 = 0.90

So, the overall probability of Likelihood of evidence for Banana = 0.8 * 0.7 * 0.9 = 0.504

Step 4: Substitute all the 3 equations into the Naive Bayes formula, to get the probability that it is a banana.

Similarly, you can compute the probabilities for 'Orange' and 'Other fruit'. The denominator is the same for all 3 cases, so it's optional to compute.

Clearly, Banana gets the highest probability, so that will be our predicted class.

6. What is Laplace Correction?
The value of P(Orange | Long, Sweet and Yellow) was zero in the above example, because, P(Long | Orange) was zero. That is, there were no ‘Long’ oranges in the training data.

It makes sense, but when you have a model with many features, the entire probability will become zero because one of the feature’s values was zero. To avoid this, we increase the count of the variable with zero to a small value (usually 1) in the numerator, so that the overall probability doesn’t become zero.

This correction is called ‘Laplace Correction’. Most Naive Bayes model implementations accept this or an equivalent form of correction as a parameter.

7. What is Gaussian Naive Bayes?
So far we’ve seen the computations when the X’s are categorical. But how to compute the probabilities when X is a continuous variable?

If we assume that the X follows a particular distribution, then you can plug in the probability density function of that distribution to compute the probability of likelihoods.

If you assume the X’s follow a Normal (aka Gaussian) Distribution, which is fairly common, we substitute the corresponding probability density of a Normal distribution and call it the Gaussian Naive Bayes. You need just the mean and variance of the X to compute this formula.

where mu and sigma are the mean and variance of the continuous X computed for a given class ‘c’ (of Y).

To make the features more Gaussian like, you might consider transforming the variable using something like the Power Transformation (See wikipedia) to achieve this.

That’s it. Now, let’s build a Naive Bayes classifier.

8. Building a Naive Bayes Classifier in R
Understanding Naive Bayes was the (slightly) tricky part. Implementing it is fairly straightforward.

In R, Naive Bayes classifier is implemented in packages such as e1071, klaR and bnlearn. In Python, it is implemented in scikit learn.

For sake of demonstration, let’s use the standard iris dataset to predict the Species of flower using 4 different features: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

```
# Import Data
training <- read.csv('iris_train.csv')
test <- read.csv('iris_test.csv')
```

The training data is now contained in training and test data in the test dataframe. Lets load the klaR package and build the naive bayes model.

```
# Using klaR for Naive Bayes
library(klaR)
nb_mod <- NaiveBayes(Species ~ ., data=training)
pred <- predict(nb_mod, test)
```

Let's see the confusion matrix.

```
# Confusion Matrix
tab <- table(pred$class, test$Species)
caret::confusionMatrix(tab)
#> Confusion Matrix and Statistics
#> setosa versicolor virginica
#> setosa 15 0 0
#> versicolor 0 11 0
#> virginica 0 4 15
#> Overall Statistics
#> Accuracy : 0.9111
#> 95% CI : (0.7878, 0.9752)
#> No Information Rate : 0.3333
#> P-Value [Acc > NIR] : 8.467e-16
#> Kappa : 0.8667
#> Mcnemar's Test P-Value : NA
#> Statistics by Class:
#> Class: setosa Class: versicolor Class: virginica
#> Sensitivity 1.0000 0.7333 1.0000
#> Specificity 1.0000 1.0000 0.8667
#> Pos Pred Value 1.0000 1.0000 0.7895
#> Neg Pred Value 1.0000 0.8824 1.0000
#> Prevalence 0.3333 0.3333 0.3333
#> Detection Rate 0.3333 0.2444 0.3333
#> Detection Prevalence 0.3333 0.2444 0.4222
#> Balanced Accuracy 1.0000 0.8667 0.9333
# Plot density of each feature using nb_mod
opar = par(mfrow=c(2, 2), mar=c(4,0,0,0))
plot(nb_mod, main="")
par(opar)

# Plot the Confusion Matrix
library(ggplot2)
test$pred <- pred$class
ggplot(test, aes(Species, pred, color = Species)) +
geom_jitter(width = 0.2, height = 0.1, size=2) +
labs(title="Confusion Matrix",
subtitle="Predicted vs. Observed from Iris dataset",
y="Predicted",
x="Truth")
```

9. Building Naive Bayes Classifier in Python

```
# Import packages
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
# Import data
training = pd.read_csv('iris_train.csv')
test = pd.read_csv('iris_test.csv')

# Create the X, Y, Training and Test
xtrain = training.drop('Species', axis=1)
ytrain = training.loc[:, 'Species']

xtest = test.drop('Species', axis=1)
ytest = test.loc[:, 'Species']
```

```
# Init the Gaussian Classifier
model = GaussianNB()
# Train the model
model.fit(xtrain, ytrain)
# Predict Output
pred = model.predict(xtest)
# Plot Confusion Matrix
mat = confusion_matrix(pred, ytest)
names = np.unique(pred)
sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,
xticklabels=names, yticklabels=names)
plt.xlabel('Truth')
plt.ylabel('Predicted')
```

10. Tips to improve the model

1. Try transforming the variables using transformations like BoxCox or Yeo Johnson to make the features near Normal.
2. Try applying Laplace correction to handle records with zeros values in X variables.
3. Check for correlated features and try removing the highly correlated ones. Naive Bayes is based on the assumption that the features are independent.
4. Feature engineering. Combining features (a product) to form new ones that make intuitive sense might help.
5. Try providing more realistic prior probabilities to the algorithm based on knowledge from business, instead of letting the algo calculate the priors based on the training sample.

For this case, ensemble methods like bagging, boosting will help a lot by reducing the variance.

11. Find the correct distribution function
One last step remains to begin to implement a classifier. How to model the probability functions P(f_i| Survival)? There are three available models in the Sklearn python library:

• Gaussian : This assumes that continuous feature follow the normal distribution.

• Multinomial: It is useful if your features are discrete.
• Bernoulli: This is useful if your features are binary.
12. Pro and cons of Naive Bayes Classifiers

Pros:
• Computationally fast
• Simple to implement
• Works well with small datasets
• Works well with high dimensions
• Perform well even if the Naive Assumption is not perfectly met.
In many cases, the approximation is enough to build a good classifier.
Cons:
• Require to remove correlated features because they are voted twice in the model and it can lead to over inflating importance.
• If a categorical variable has a category in test data set which was not observed in training data set, then the model will assign a zero probability. It will not be able to make a prediction. This is often known as “Zero Frequency”. To solve This way, we can use the smoothing technique. One of the simplest Smoothing techniques are called. Sklearn applies Laplace smoothing by default when you train a Naive Bayes classifier.