

What Is the Naive Bayes Algorithm?

Bayesian statistics (named after *Thomas Bayes*), describes the probability of an event, based on conditions that might be related to the event. At the core of Bayesian statistics is Bayes' theorem, which describes the outcome probabilities of related (dependent) events using the concept of conditional probability.

The Naive Bayes algorithm is a classification algorithm that is based on Bayes' theorem, which is a way of calculating the probability of an event based on its prior knowledge. The algorithm is called “**naive**” because it makes a simplifying assumption that the features are conditionally independent of each other given the class label.

The Naive Bayes algorithm can be used for binary as well as multi-class classification problems. It is commonly used in text classification tasks, such as spam filtering or sentiment analysis, but it can also be used in other applications where there are multiple classes and multiple features.

For example, if a particular illness is related to age and lifestyle, then by applying Bayes' theorem by considering a person's age and lifestyle the probability of that individual having the illness can be assessed more accurately.

Bayes theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

where A and B are events and $P(B) \neq 0$.

- $P(A)$ and $P(B)$ are the probabilities of observing A and B without regard to each other.
- $P(A | B)$, a conditional probability, is the probability of observing event A given that B is true.
- $P(B | A)$ is the probability of observing event B given that A is true.

For example, a doctor knows that lack of sleep causes a migraine 50% of the time.

The prior probability of any patient having lack of sleep is 10,000/50,000 and the prior probability of any patient having a migraine is 300/1,000.

If a patient has a sleep disorder, let's apply Bayes theorem to calculate the probability of he/she having a migraine.

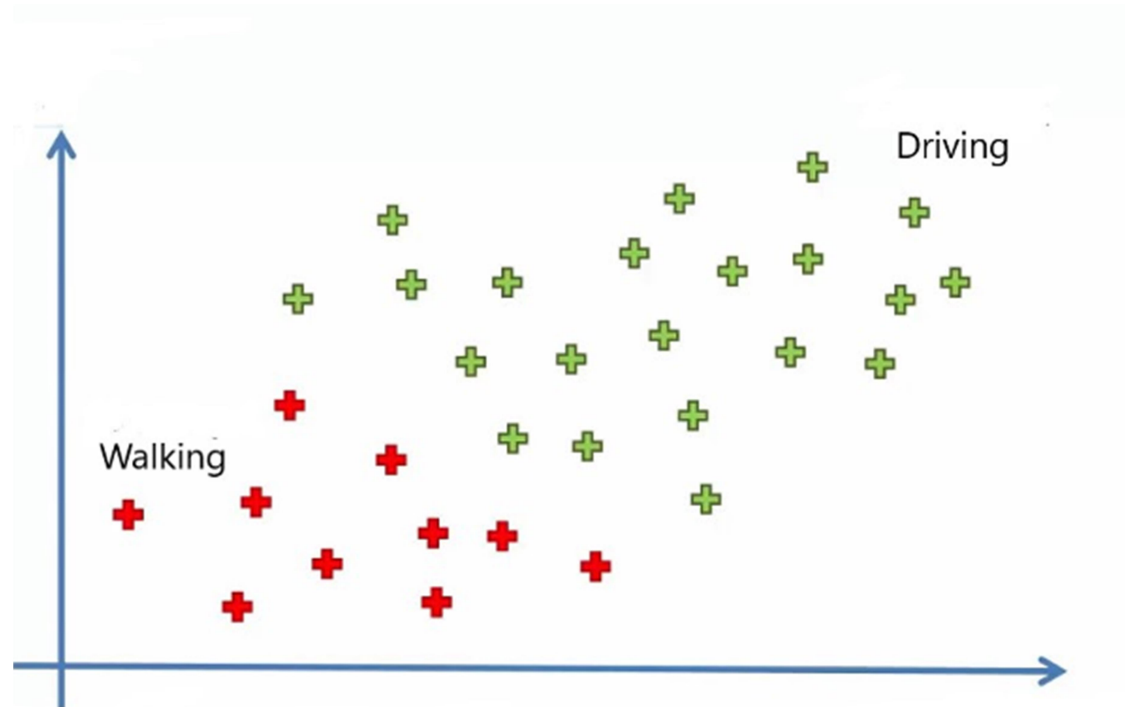
$$P(\text{Sleep disorder} | \text{Migraine}) = P(\text{Migraine} | \text{Sleep disorder}) * P(\text{Migraine}) / P(\text{Sleep disorder})$$

$$P(\text{Sleep disorder} | \text{Migraine}) = .5 * 10000/50000 / (300/1000) = 33\%$$

In the preceding scenario, there is a 33% chance that a patient with a sleep disorder will have a migraine problem.

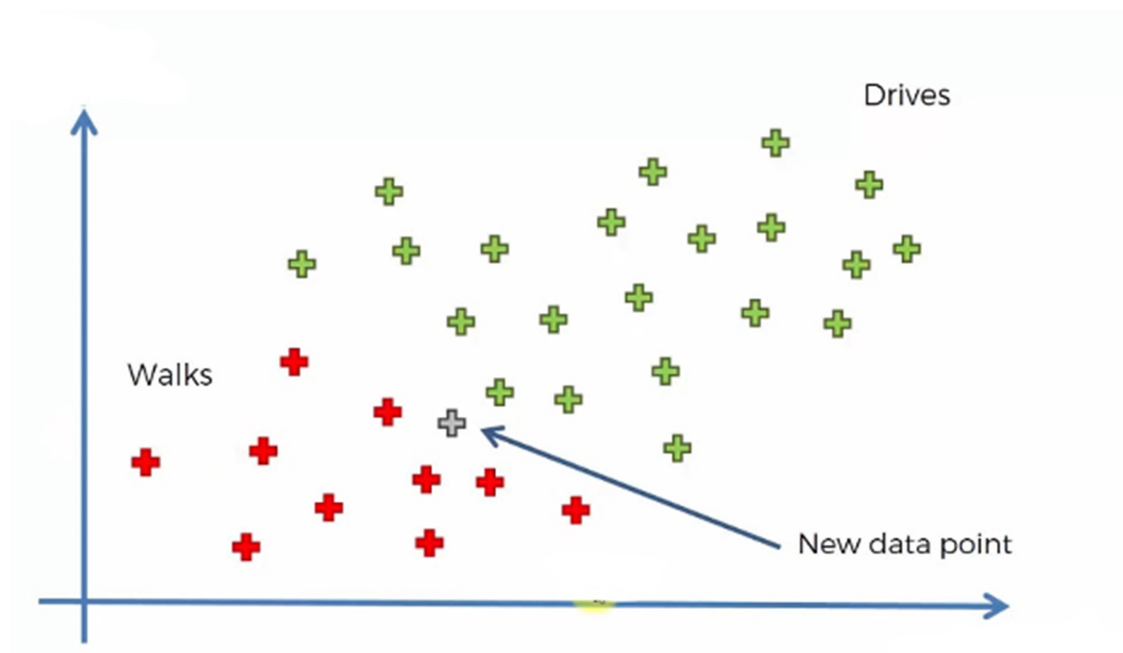
Naive Bayes Theorem: The Concept Behind the Algorithm

Let's understand the concept of the Naive Bayes Theorem and how it works through an example. We are taking a case study in which we have the dataset of employees in a company, our aim is to create a model to find whether a person is going to the office by driving or walking using the salary and age of the person.



In the above image, we can see 30 data points in which red points belong to those who are walking and green belong to those who are driving.

Now let's add a new data point to it. Our aim is to find the category that the new point belongs to.



Note that we are taking **age on the X-axis** and **Salary on the Y-axis**. We are using the Naive Bayes algorithm to find the category of the new data point. For this, we have to find the posterior probability of walking and driving for this data point. After comparing, the point belongs to the category having a higher probability.

Posterior Probability

Likelihood

Prior Probability

$$P(Walks|X) = \frac{P(X|Walks) * P(Walks)}{P(X)}$$

Marginal Likelihood

Detailed description: This diagram illustrates Bayes' theorem. The equation $P(Walks|X) = \frac{P(X|Walks) * P(Walks)}{P(X)}$ is centered. Four labels with arrows point to parts of the equation: 'Posterior Probability' points to $P(Walks|X)$; 'Likelihood' points to $P(X|Walks)$; 'Prior Probability' points to $P(Walks)$; and 'Marginal Likelihood' points to $P(X)$.

The posterior probability of walking for the new data point is:

Posterior Probability

Likelihood

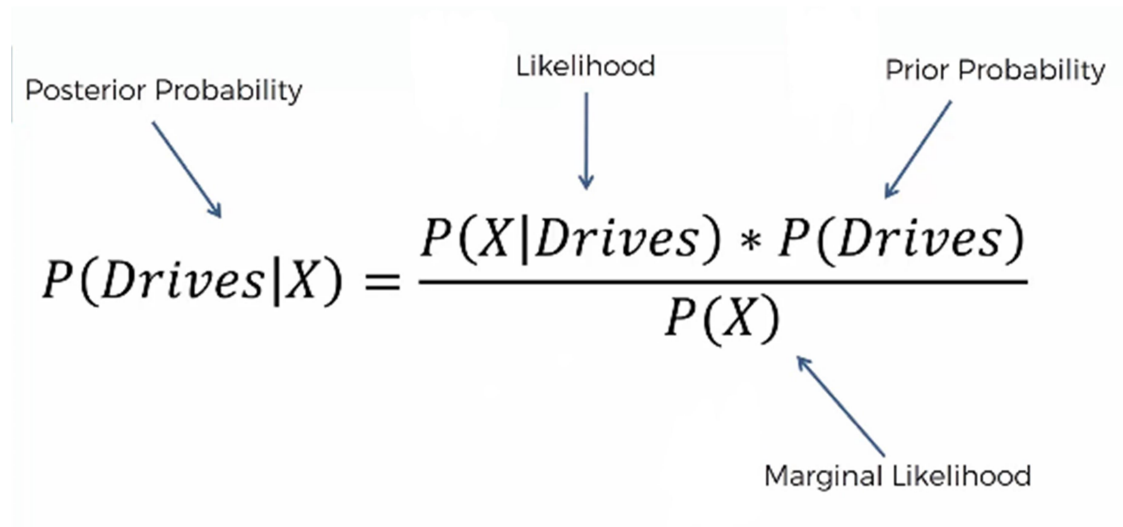
Prior Probability

$$P(Walks|X) = \frac{P(X|Walks) * P(Walks)}{P(X)}$$

Marginal Likelihood

Detailed description: This diagram is identical to the one above, showing the same equation and labels with arrows pointing to the corresponding terms in Bayes' theorem.

And, that for the driving is:



The diagram shows the Bayes' Theorem formula with labels and arrows indicating the components:

$$P(Drives|X) = \frac{P(X|Drives) * P(Drives)}{P(X)}$$

Labels and arrows:

- Posterior Probability** points to $P(Drives|X)$.
- Likelihood** points to $P(X|Drives)$.
- Prior Probability** points to $P(Drives)$.
- Marginal Likelihood** points to $P(X)$.

Steps Involved in the Naive Bayes Classifier Algorithm

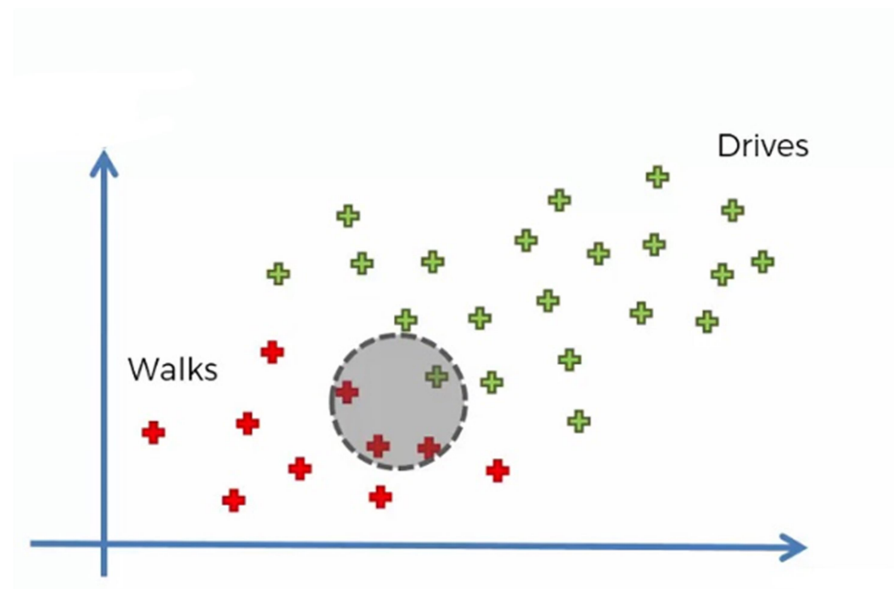
Step 1: We have to find all the probabilities required for the Bayes theorem for the calculation of posterior probability.

$P(\text{Walks})$ is simply the probability of those who walk among all.

$$P(\text{Walks}) = \frac{\text{Number of Walkers}}{\text{Total Observations}}$$

$$P(\text{Walks}) = \frac{10}{30}$$

In order to find the marginal likelihood, $P(X)$, we have to consider a circle around the new data point of any radii, including some red and green points.



$$P(X) = \frac{\text{Number of Similar Observations}}{\text{Total Observations}}$$

$$P(X) = \frac{4}{30}$$

$P(X|Walks)$ can be found by:

$$P(X|Walks) = \frac{\begin{matrix} \text{Number of Similar} \\ \text{Observations} \\ \text{Among those who Walk} \end{matrix}}{\text{Total number of Walkers}}$$

$$P(X|Walks) = \frac{3}{10}$$

Now we can find the posterior probability using the Bayes theorem,

$$P(Walks|X) = \frac{\frac{3}{10} * \frac{10}{30}}{\frac{4}{30}} = 0.75$$

Step 2: Similarly, we can find the posterior probability of Driving, and it is 0.25

Step 3: Compare both posterior probabilities. When comparing the posterior probability, we can find that $P(\text{walks} | X)$ has greater values, and the new point belongs to the walking category.

Gaussian Naive Bayes

Gaussian Naïve Bayes implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

Implementation of Naive Bayes (Python Programming)

Now let's implement Naive Bayes step by step using the python programming language

We are using the **Social network ad dataset**.

The dataset contains the details of users on a social networking site to find whether a user buys a product by clicking the ad on the site based on their salary, age, and gender.

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0

.....
.....

Step 1: Importing the libraries:

Let's start the programming by importing the essential libraries required.

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

```
import sklearn
```

Step 2: Importing the dataset:

Since our dataset contains character variables, we have to encode it using LabelEncoder.

```
from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()  
X[:,0] = le.fit_transform(X[:,0])
```

Step 3: Train test splitting:

We are splitting our data into train and test datasets using the scikit-learn library. We are providing the test size as 0.20, which means our training data contains 320 training sets, and the test sample contains 80 test sets.

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

Step 4: Feature scaling:

Next, we are doing **feature scaling** to the training and test set of independent variables.

```
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Step 5: Training the Naive Bayes model on the training set:

```
from sklearn.naive_bayes import GaussianNB
```

```
classifier = GaussianNB()
```

```
classifier.fit(X_train, y_train)
```

Let's predict the test results:

```
y_pred = classifier.predict(X_test)
```

Predicted and actual value –

y_pred

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0

y_test

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0

For the first 8 values, both are the same. We can evaluate our matrix using the confusion matrix and accuracy score by comparing the predicted and actual test values:

```
from sklearn.metrics import confusion_matrix, accuracy_score  
cm = confusion_matrix(y_test, y_pred)  
ac = accuracy_score(y_test, y_pred)
```

confusion matrix –

ac – 0.9125

	0	1
0	55	3
1	4	18

Accuracy is good. Note that you can achieve better results for this problem using different algorithms.

Full Python Code:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
# Predicting the Test set results
y_pred = classifier.predict(X_test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print(y_pred)
print(y_test)
print(ac)
cm
```

Output:

```
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0
 0 0 0 0 1 1]
```

```
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 0 1
 0 0 0 0 1 1]
```

0.9125

Out[10]:
array([[55, 3],
 [4, 18]], dtype=int64)

What Are the Assumptions Made by the Naive Bayes Algorithm?

There are several variants of Naive Bayes, such as Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes. Each variant has its own assumptions and is suited for different types of data. Here are some assumptions that the Naive Bayes algorithm makes:

1. The main assumption is that it assumes that the features are conditionally independent of each other.
2. Each of the features is equal in terms of weightage and importance.
3. The algorithm assumes that the features follow a normal distribution.
4. The algorithm also assumes that there is no or almost no correlation among features.

Conclusion:

The naive Bayes algorithm is a powerful and widely-used machine learning algorithm that is particularly useful for classification tasks. This tutorial explains the basic math behind the Naive Bayes algorithm and how it works for binary classification problems. Its simplicity and efficiency make it a popular choice for many data science applications. Most concepts of the algorithm and how to implement it in Python have been covered here.

Key Takeaways:

- Naive Bayes is a probabilistic classification algorithm(binary or multi-class) that is based on Bayes' theorem.
- There are different variants of Naive Bayes, which can be used for different tasks and can even be used for regression problems.
- Naive Bayes can be used for a variety of applications, such as spam filtering, sentiment analysis, and recommendation systems.

Frequently Asked Questions:

Q1. When should we use a naive Bayes classifier?

A. The naive Bayes classifier is a good choice when you want to solve a binary or multi-class classification problem when the dataset is relatively small and the features are conditionally independent. It is a fast and efficient algorithm that can often perform well, even when the assumptions of conditional independence do not strictly hold. Due to its high speed, it is well-suited for real-time applications. However, it may not be the best choice when the features are highly correlated or when the data is highly imbalanced.

Q2. What is the difference between Bayes Theorem and Naive Bayes Algorithm?

A. Bayes theorem provides a way to calculate the conditional probability of an event based on prior knowledge of related conditions. The naive Bayes algorithm, on the other hand, is a machine learning algorithm that is based on Bayes' theorem, which is used for classification problems.

Q3. Is Naive Bayes a regression technique or classification technique?

It is not a regression technique, although one of the three types of Naive Bayes, called Gaussian Naive Bayes, can be used for regression problems.

References:

1. <https://www.analyticsvidhya.com/blog/2021/01/a-guide-to-the-naive-bayes-algorithm/>
Surabhi S — Published On January 16, 2021 and Last Modified On March 3rd, 2023
2. <https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/>
3. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB
4. <http://i.stanford.edu/pub/ctr/reports/cs/tr/79/773/CS-TR-79-773.pdf>