Naive Bayes Algorithm?

It is a classification technique based on Bayes' Theorem with an independence assumption among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

The **Naïve Bayes classifier** is a popular supervised machine learning algorithm used for classification tasks such as text classification. It belongs to the family of generative learning algorithms, which means that it models the distribution of inputs for a given class or category. This approach is based on the assumption that the features of the input data are conditionally independent given the class, allowing the algorithm to make predictions quickly and accurately.

In statistics, naive Bayes are simple probabilistic classifiers that apply Bayes' theorem. This theorem is based on the probability of a hypothesis, given the data and some prior knowledge. The naive Bayes classifier assumes that all features in the input data are independent of each other, which is often not true in real-world scenarios. However, despite this simplifying assumption, the naive Bayes classifier is widely used because of its efficiency and good performance in many real-world applications.

Moreover, it is worth noting that naive Bayes classifiers are among the simplest Bayesian network models, yet they can achieve high accuracy

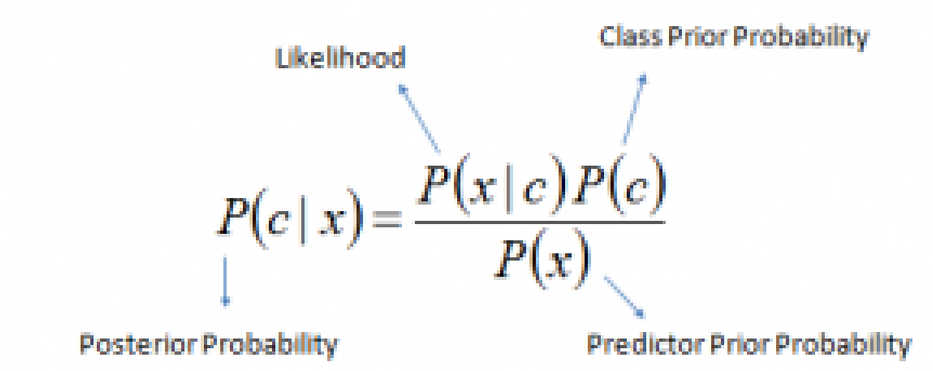
levels when coupled with kernel density estimation. This technique involves using a kernel function to estimate the probability density function of the input data, allowing the classifier to improve its performance in complex scenarios where the data distribution is not well-defined. As a result, the naive Bayes classifier is a powerful tool in machine learning, particularly in text classification, spam filtering, and sentiment analysis, among others.

Example of Naive Bayes Algorithm

For example, if a fruit is red, round, and about 3 inches wide, we might call it an apple. Even if these things are related, each one helps us decide it’ s probably an apple. That’ s why it’ s called ‘ Naive.

An **NB model** is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of computing posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

P(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).

P(*c*) is the prior probability of *class*.

P(*x*|*c*) is the likelihood which is the probability of the *predictor* given *class*.

P(*x*) is the prior probability of the *predictor*.

How Do Naive Bayes Algorithms Work?

1. **Convert the data set into a frequency table**

   In this first step data set is converted into a frequency table

2. **Create Likelihood table by finding the probabilities**

   Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

3. **Use Naive Bayesian equation to calculate the posterior probability**

   Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of the prediction.

Problem: Players will play if the weather is sunny. Is this statement correct?

We can solve it using the above-discussed method of posterior probability.

P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)

Here P(Sunny | Yes) * P(Yes) is in the numerator, and P (Sunny) is in the denominator.

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

The Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in **text classification** (nlp) and with problems having multiple classes.

The Pros and Cons of Naive Bayes?

Pros:

It is easy and fast to predict class of test data set. It also perform well in multi class prediction

When assumption of independence holds, the classifier performs better compared to other **machine learning models** like **logistic regression** or decision tree, and requires less training data.

It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “ Zero Frequency” . To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.

On the other side, **Naive Bayes** is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

Another limitation of this algorithm is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Applications of Naive Bayes Algorithms

Real-time Prediction: Naive Bayesian classifier is an eager learning classifier and it is super fast. Thus, it could be used for making predictions in real time.

Multi-class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.

Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayesian classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)

Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not.

Naive Bayes classifiers

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

One of the most simple and effective classification algorithms, the Naïve Bayes classifier aids in the rapid development of machine learning models with rapid prediction capabilities.

Assumption of Naive Bayes

The fundamental Naive Bayes assumption is that each feature makes an:

Feature independence: The features of the data are conditionally independent of each other, given the class label.

Continuous features are normally distributed: If a feature is continuous, then it is assumed to be normally distributed within each class.

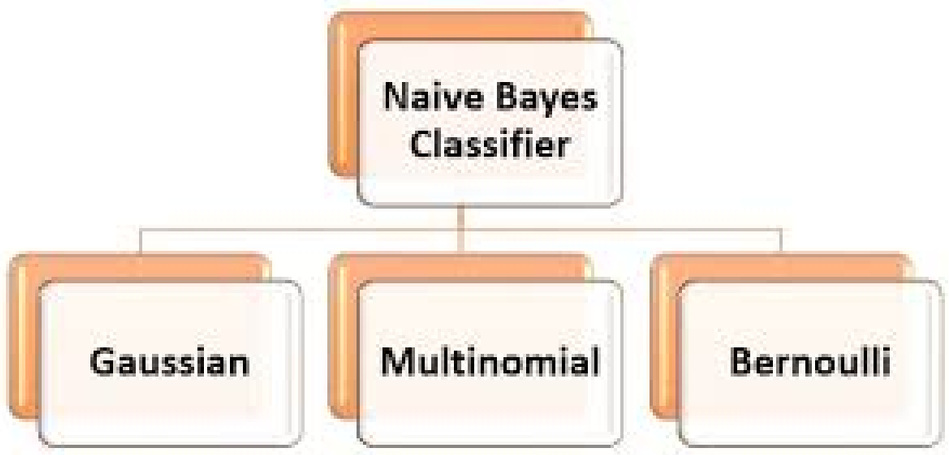
Discrete features have multinomial distributions: If a feature is discrete, then it is assumed to have a multinomial distribution within each class.

Features are equally important: All features are assumed to contribute equally to the prediction of the class label.

No missing data: The data should not contain any missing values.

Types of Naive Bayes

There are mainly a total of three types of naive byes algorithms. Different types of naive Bayes are used for different use cases. Let us try to understand them one by one.



1. Bernoulli Naive Bayes

This naive Bayes algorithm is used when there is a **boolean** type of dependent or target variable present in the dataset. For example, a dataset has target column categories as Yes and No. This type of Naive is mainly used in a binary categorical tagete column where the problem statement is to predict only **Yes or No**. For Example, sentiment analysis with Positive and Negative Categories, A specific ord id present in the text or not, etc.

Code Example:

```
from sklearn.datasets import make_classification
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split
nb_samples = 100
```

```
X, Y = make_classification(n_samples=nb_samples, n_features=2,
n_informative=2, n_redundant=0)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.25)
bnb = BernoulliNB(binarize=0.0)
bnb.fit(X_train, Y_train)
bnb.score(X_test, Y_test)
```

2. Multinomial Naive Bayes

This type of naive Bayes is used where the data is multinomial distributed. This type of naive Bayes is mainly used when there is a **text classification** problem.

For Example, if you want to predict whether a text belongs to which tag, education, politics, e-tech, or some other tag, you can use the multinomial naive Bayes to classify the same.

This naive base **outperforms** text classification problems and is used the most out of all the other naive Bayes algorithms.

Code Example:

```
from sklearn.feature_extraction import DictVectorizer
from sklearn.naive_bayes import MultinomialNB
data = [
{'parth1': 100, 'parth2': 50, 'parth3': 25, 'parth4': 100, 'parth5': 20},
{'parth1': 5, 'parth2': 5, 'parth3': 0, 'parth4': 10, 'parth5': 500, 'parth6':
1}
]
dv = DictVectorizer(sparse=False)
```

```
X = dv.fit_transform(data)

Y = np.array([1, 0])

mnb = MultinomialNB()

mnb.fit(X, Y)

test_data = data = [

{'parth1': 80, 'parth2': 20, 'parth3': 15, 'parth4': 70, 'parth5': 10,

'parth6':

1},

]

{'parth1': 10, 'parth2': 5, 'parth3': 1, 'parth4': 8, 'parth5': 300, 'parth6':

0}

mnb.predict(dv.fit_transform(test_data))
```

3. Gaussian Naive Bayes

This type of naive is used when the predictor variables have **continuous values** instead of discrete ones. Here it is assumed that the distribution of the data is **Gaussian distribution**.

Code Example:

```
from sklearn.datasets import make_classification

from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import train_test_split

nb_samples = 100

X, Y = make_classification(n_samples=nb_samples, n_features=2,

n_informative=2, n_redundant=0)
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.25)
gnb = GaussianNB(binarize=0.0)
gnb.fit(X_train, Y_train)
bngnb.score(X_test, Y_test)
```

Advantages and Disadvantages of Naive Bayes

Advantages

1. Faster Algorithms:

The Naive Bayes algorithm is a parametric algorithm that tries to assume certain things while training and using the knowledge for prediction. Hence it takes significantly less time for prophecy and is a **faster algorithm**.

2. Less Training Data:

The naive Bayes algorithm assumes the **independent features to be independent** of each other, and if it exists, then the naive Bayes needs less data for training and performs better.

3. Performance:

The Naive Bayes algorithm performs **faster and more accurately** on fewer data, and the algorithm' s results on categorical text data are overpowered, which can not be compared to other algorithms.

Disadvantages

1. Independent Features:

In a real-time dataset, it is almost impossible to get independent features that are entirely independent of each other, there are nearly two to three features that are correlated with each other, and hence the assumption is **not fully satisfied** all the time.

2. Zero Frequency Error:

The zero frequency error in naive Bayes is one of the most critical CONs of the Naive Bayes algorithm. According to this error, if a category is not present in the training data and in the test data, then the Naive Bayes algorithm will assign it zero probability, and the error is called **Zero Frequency** error in Naive Bayes. To address this kind of issue, we can use Laplace smoothing techniques.

When to Use Naive Bayes?

Well, the Naive Bayes algorithm is the best-performing and faster algorithm compared to other algorithms. However, still, there are cases where it cannot perform well, and some different algorithms should be used to handle such cases.

The Naive Bayes algorithm can be used if there is no multicollinearity in the independent features and if the features' probabilities provide some valuable information to the algorithms. This algorithm should also be preferred for text classification problems. One should avoid using the Naive Bayes algorithm when the data is entirely numeric and multicollinearity is present in the dataset.

If it is necessary to use the Naive Bayes algorithm, then one can use the following steps to improve the performance of Naive Bayes algorithms.

How to Improve Naive Bayes?

1. Remove Correlated Features:

Naive Bayes algorithms perform well on datasets with no correlations in independent features. **removing the correlated features** may improve the performance of the algorithm

2. Feature Engineering:

Try to apply feature engineering to the dataset and its features, **combine some** of the elements, and **extract some parts** of them out of existing ones. This may help the Naive Bayes algorithm learn the data quickly and results in an accurate model.

3. Use Some Domain Knowledge:

Oe should always try to apply some **domain knowledge** to the dataset and its features and take steps according to it. It may help the algorithm to make decisions faster and achieve higher accuracies.

4. Probabilistic Features:

The Naive Bayes algorithm works on the concept of probabilities, so try to improve the features that give **more weightage to the algorithms and their probabilities**, try to implement those, and run the roses in a loop to know which features are best for the algorithm.

5. Laplace Transform:

In some cases, the category may be present in the test dataset and was not present while training and the model will assign it with zero probability. Here we should handle this issue by **Laplace transform**.

6. Feature Transformation:

It is always better to have normal distributions in the datasets and try to apply **box-cox and yeo-johnson** feature transformation techniques to achieve the normal distributions in the dataset.

Conclusion

In this article, we discussed the naive Bayes algorithm, the probabilities, conditional probabilities, the bayesian theorem, the core intuition and working mechanism of the algorithm with their types, code examples, applications, PROs, and CONs associated with some of the key takeaways from this article. This article’ s complete knowledge will help one understand the Naive Bayes algorithm from scratch to an in-depth level. It will help answer the interviews related to it very efficiently.