What are Naïve Bayes classifiers?

Learn how Naïve Bayes classifiers uses principles of probability to perform classification tasks.

Explore watsonx.ai →

Subscribe to the IBM newsletter →

Naïve Bayes classifiers

The Naïve Bayes classifier is a supervised machine learning algorithm, which is used for classification tasks, like text classification. It is also part of a family of generative learning algorithms, meaning that it seeks to model the distribution of inputs of a given class or category. Unlike discriminative classifiers, like logistic regression, it does not learn which features are most important to differentiate between classes.

Check out our tutorial to learn how to apply this classifier using Python →

A brief review of Bayesian statistics

Naïve Bayes is also known as a probabilistic classifier since it is based on Bayes' Theorem. It would be difficult to explain this algorithm without explaining the basics of Bayesian statistics. This theorem, also known as Bayes' Rule, allows us to "invert" conditional probabilities. As a reminder, conditional probabilities represent the probability of an event given some other event has occurred, which is represented with the following formula:

$$P(Y|X) = \frac{P(X \; and \; Y)}{P(X)}$$

Bayes' Theorem is distinguished by its use of sequential events, where additional information later acquired impacts the initial probability. These probabilities are denoted as the prior probability and the posterior probability. The prior probability is the initial probability of an event before it is contextualized under a certain condition, or the marginal probability. The posterior probability is the probability of an event after observing a piece of data.

A popular example in statistics and machine learning literature (link resides outside ibm.com) to demonstrate this concept is medical testing. For instance, imagine there is an individual, named Jane, who takes a test to determine if she has diabetes. Let’s say that the overall probability having diabetes is 5%; this would be our prior probability. However, if she obtains a positive result from her test, the prior probability is updated to account for this additional information, and it then becomes our posterior probability. This example can be represented with the following equation, using Bayes’ Theorem:

$$p(diabetes\,|+\,test) = \frac{p(+\,test\,|\,diabetes)p(diabetes)}{P(+\,test|diabetes)p(diabetes) + p(+\,test\,|\,no\ diabetes)\ p(no\ diabetes)}$$

However, since our knowledge of prior probabilities is not likely to exact given other variables, such as diet, age, family history, et cetera, we typically leverage probability distributions from random samples, simplifying the equation to P(Y|X) = P(X|Y)P(Y) / P(X)

The return to Naïve Bayes

Naïve Bayes classifiers work differently in that they operate under a couple of key assumptions, earning it the title of “naïve”. It assumes that predictors in a Naïve Bayes model are conditionally independent, or unrelated to any of the other feature in the model. It also assumes that all features contribute equally to the outcome. While these assumptions are often violated in real-world scenarios (e.g. a subsequent word in an e-mail is dependent upon the word that precedes it), it simplifies a classification problem by making it more computationally tractable. That is, only a single probability will now be required for each variable, which, in turn, makes the model computation easier. Despite this unrealistic independence assumption, the classification algorithm performs well, particularly with small sample sizes.

With that assumption in mind, we can now reexamine the parts of a Naïve Bayes classifier more closely. Similar to Bayes’ Theorem, it’ll use conditional and prior probabilities to calculate the posterior probabilities using the following formula:

$$\text{posterior probability} = \frac{(\text{conditional probability})(\text{prior probability})}{\text{evidence (a. k. a. “stabilizer”)}}$$

Now, let’s imagine text classification use case to illustrate how the Naïve Bayes algorithm works. Picture an e-mail provider that is looking to improve their spam filter. The training data would consist of words from e-mails that have been classified as either “spam” or “not spam”. From there, the class conditional probabilities and the prior probabilities are calculated to yield the posterior probability. The Naïve Bayes classifier will operate by returning the class, which has the maximum posterior probability out of a group of classes (i.e. “spam” or “not spam”) for a given e-mail. This calculation is represented with the following formula:

$$\hat{y} = arg \max_{y \in Y} P(y|x) = argmax \frac{P(x|y)P(y)}{P(x)}$$

Since each class is referring to the same piece of text, we can actually eliminate the denominator from this equation, simplifying it to:

$$\hat{y} = argmax\ P(x|y)P(y)$$

The accuracy of the learning algorithm based on the training dataset is then evaluated based on the performance of the test dataset.

Class-conditional probabilities

To unpack this a little more, we'll go a level deeper to the individual parts, which comprise this formula. The class-conditional probabilities are the individual likelihoods of each word in an e-mail. These are calculated by determining the frequency of each word for each category—i.e. "spam" or "not spam", which is also known as the maximum likelihood estimation (MLE). In this example, if we were examining if the phrase, "Dear Sir", we'd just calculate how often those words occur within all spam and non-spam e-mails. This can be represented by the formula below, where y is "Dear Sir" and x is "spam".

$$\hat{P}(y = [Dear\ Sir] | x = spam) = P(dear \mid spam)\ P(sir \mid spam)$$

Prior probabilities

The prior probabilities are exactly what we described earlier with Bayes’ Theorem. Based on the training set, we can calculate the overall probability that an e-mail is “spam” or “not spam”. The prior probability for class label, “spam”, would be represented within the following formula:

$$\hat{P}(spam) = \frac{\#\ of\ spam\ messages\ in\ training\ set}{\#\ of\ all\ messages\ in\ training\ set}$$

The prior probability acts as a “weight” to the class-conditional probability when the two values are multiplied together, yielding the individual posterior probabilities. From there, the maximum a posteriori (MAP) estimate is calculated to assign a class label of either spam or not spam. The final equation for the Naïve Bayesian equation can be represented in the following ways:

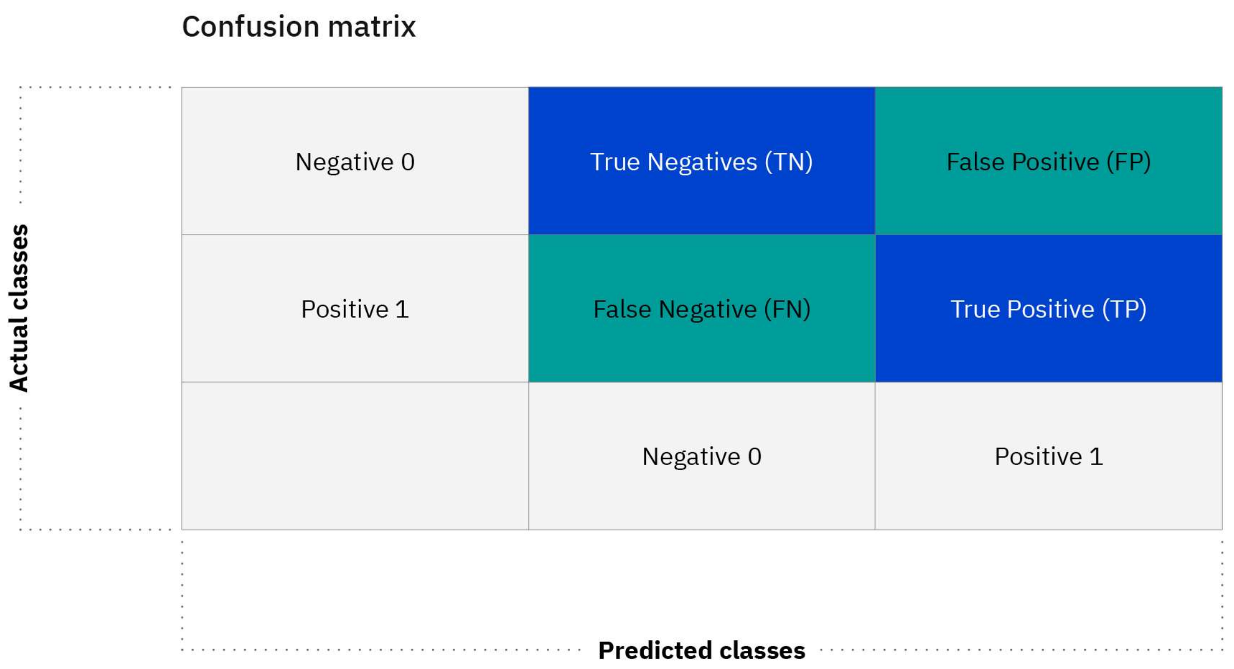
$$\widehat{class\ label} = arg \max_{y \in Y} P(class\ label) \prod_{i \in I} P(\ word_i \mid class\ label)$$

Alternatively, it can be represented in the log space as naïve bayes is commonly used in this form:

$$\widehat{class\ label} = arg\ \max_{y \in Y} \log P(class\ label) + \sum_{i \in I} \log P(\ word_i \mid class\ label)$$

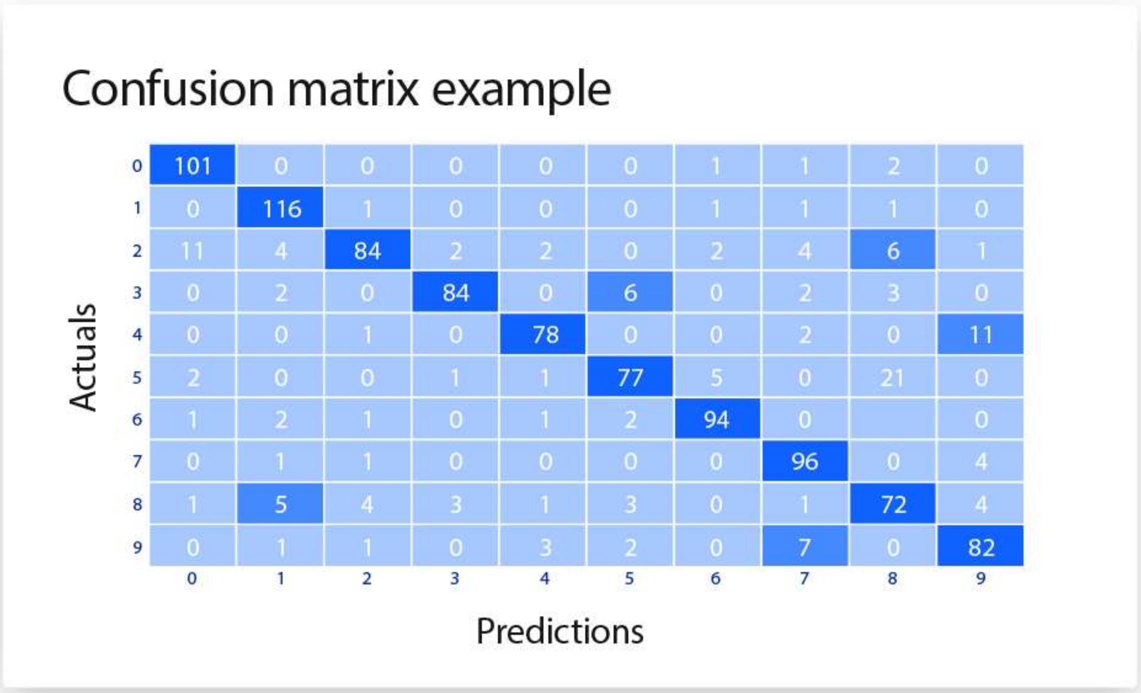
Evaluating your Naïve Bayes classifier

One way to evaluate your classifier is to plot a confusion matrix, which will plot the actual and predicted values within a matrix. Rows generally represent the actual values while columns represent the predicted values. Many guides will illustrate this figure as a 2 x 2 plot, such as the below:


Confusion matrix

Actual classes		
Negative 0	True Negatives (TN)	False Positive (FP)
Positive 1	False Negative (FN)	True Positive (TP)
	Negative 0	Positive 1
	Predicted classes	

However, if you were predicting images from zero through 9, you'd have a 10 x 10 plot. If you wanted to know the number of times that classifier "confused" images of 4s with 9s, you'd only need to check the 4th row and the 9th column.


Confusion matrix example

Actuals \ Predictions	0	1	2	3	4	5	6	7	8	9
0	101	0	0	0	0	0	1	1	2	0
1	0	116	1	0	0	0	1	1	1	0
2	11	4	84	2	2	0	2	4	6	1
3	0	2	0	84	0	6	0	2	3	0
4	0	0	1	0	78	0	0	2	0	11
5	2	0	0	1	1	77	5	0	21	0
6	1	2	1	0	1	2	94	0		0
7	0	1	1	0	0	0	0	96	0	4
8	1	5	4	3	1	3	0	1	72	4
9	0	1	1	0	3	2	0	7	0	82

Types of Naïve Bayes classifiers

There isn't just one type of Naïve Bayes classifier. The most popular types differ based on the distributions of the feature values. Some of these include:

- **Gaussian Naïve Bayes (GaussianNB)**: This is a variant of the Naïve Bayes classifier, which is used with Gaussian distributions—i.e. normal distributions—and continuous variables. This model is fitted by finding the mean and standard deviation of each class.
- **Multinomial Naïve Bayes (MultinomialNB)**: This type of Naïve Bayes classifier assumes that the features are from multinomial distributions. This variant is useful when using discrete data, such as frequency counts, and it is typically applied within natural language processing use cases, like spam classification.
- **Bernoulli Naïve Bayes (BernoulliNB)**: This is another variant of the Naïve Bayes classifier, which is used with Boolean variables—that is, variables with two values, such as True and False or 1 and 0.

All of these can be implemented through the Scikit Learn (link resides outside ibm.com) Python library (also known as sklearn).

Advantages and disadvantages of the Naïve Bayes classifier

Advantages

- **Less complex**: Compared to other classifiers, Naïve Bayes is considered a simpler classifier since the parameters are easier to estimate. As a result, it's one of the first algorithms learned within data science and machine learning courses.
- **Scales well**: Compared to logistic regression, Naïve Bayes is considered a fast and efficient classifier that is fairly accurate when the conditional independence assumption holds. It also has low storage requirements.
- **Can handle high-dimensional data**: Use cases, such document classification, can have a high number of dimensions, which can be difficult for other classifiers to manage.

Disadvantages:

- **Subject to Zero frequency**: Zero frequency occurs when a categorical variable does not exist within the training set. For example, imagine that we're trying to find the maximum likelihood estimator for the word, "sir" given class "spam", but the word, "sir" doesn't exist in the training data. The probability in this case would zero, and since this classifier multiplies all the conditional probabilities together, this also means that posterior probability will be zero. To avoid this issue, laplace smoothing can be leveraged.
- **Unrealistic core assumption**: While the conditional independence assumption overall performs well, the assumption does not always hold, leading to incorrect classifications.

Applications of the Naïve Bayes classifier

Along with a number of other algorithms, Naïve Bayes belongs to a family of data mining algorithms which turn large volumes of data into useful information. Some applications of Naïve Bayes include:

- **Spam filtering**: Spam classification is one of the most popular applications of Naïve Bayes cited in literature. For a deeper read on this use case, check out this chapter from **Oreilly** (link resides outside ibm.com).
- **Document classification**: Document and text classification go hand in hand. Another popular use case of Naïve Bayes is content classification. Imagine the content categories of a News media website. All the content categories can be classified under a topic taxonomy based on the each article on the site. Federick Mosteller and David Wallace are credited with the first application of Bayesian inference within their 1963 **paper** (link resides outside ibm.com).

– **Sentiment analysis**: While this is another form of text classification, sentiment analysis is commonly leveraged within marketing to better understand and quantify opinions and attitudes around specific products and brands.
– **Mental state predictions**: Using fMRI data, naïve bayes has been leveraged to predict different cognitive states among humans. The goal of this **research** (link resides outside ibm.com) was to assist in better understanding hidden cognitive states, particularly among brain injury patients.

Related products

IBM Cloud Pak for Data

IBM Cloud Pak for Data is an open, extensible data platform that provides a data fabric to make all data available for AI and analytics, on any cloud.

Learn more about IBM Cloud Pak for Data →

IBM Watson Studio

Build, run and manage AI models. Prepare data and build models on any cloud using open source code or visual modeling. Predict and optimize your outcomes.

Learn more about IBM Watson Studio →

watsonx.ai

Take the next step to start operationalizing and scaling generative AI and machine learning for business.

Explore watsonx.ai →

Resources

How-to

Free, hands-on learning for generative AI technologies

Learn the fundamental concepts for AI and generative AI, including prompt enginee

Research

An Analysis of Naive Bayes Classifier on Low-Entropy Distributions

In this paper, IBM Research demonstrates empirically how decreasing entropy of class-

Research

An Analysis of Data Characteristics that Affect Naive Bayes Performance

Using Monte Carlo simulations, IBM Research shows that Naive Bayes works best in

Tutorial

Learn classification algorithms using Python and scikit-learn

Explore the basics of solving a classification-based machine learning problem, and get

Tutorial

Classifying data using the Multinomial Naive Bayes algorithm

Use scikit-learn to complete a popular text classification task (spam filtering) using

Take the next step

Train, validate, tune and deploy generative AI, foundation models and machine learning capabilities with IBM watsonx.ai™, a next generation enterprise studio for AI builders. Build AI applications in a fraction of the time with a fraction of the data.

Explore watsonx.ai →

Request a demo →