Reading: Naive Bayes classifiers

Sometimes, the simplest solutions are the most powerful ones. When it comes to supervised machine learning techniques, Naive Bayes is a perfect example of that. The theoretical foundations of the model date back nearly 300 years, and though the field of data science and machine learning has grown immensely in recent years, Naive Bayes models remain relevant because they are simple, fast, and good predictors. In certain situations, Naive Bayes is also known to outperform much more advanced classification methods. Even if a more advanced model is required, producing a Naive Bayes model can also be a great starting point. Therefore, the Naive Bayes classifier is something that every data professional needs in their machine learning skill set.

How do Naive Bayes models work?

A Naive Bayes model is a supervised learning technique used for classification problems. As with all supervised learning techniques, to create a Naive Bayes model you must have a response variable and a set of predictor variables to train the model.

The Naive Bayes algorithm is based on Bayes' Theorem, an equation that can be used to calculate the probability of an outcome or class, given the values of predictor variables. This value is known as the posterior probability.

That probability is calculated using three values:

- The probability of the outcome overall P(A)
- The probability of the value of the predictor variable P(B)
- The conditional probability P(B|A) (Note: P(B|A) is interpreted as *the probability of B, given A*.)

The probability of the outcome overall, P(A), is multiplied by the conditional probability, P(B|A). This result is then divided by the probability of the predictor variable, P(B), to obtain the posterior probability.

Bayes' Theorem:

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

The goal of Bayes' Theorem is to find the probability of an event, A, given that another event B is true. In the context of a predictive model, the class label would be A and the predictor variable would be B. P(A) is considered the prior probability of event A before any evidence (feature) is seen. Then, P(A|B) is the posterior probability, or the probability of the class label after the evidence (feature) has been seen.

In a predictive model, this calculation is carried out for each feature, for each class. Then the probabilities are multiplied together. The class with the highest resulting product is the model's final prediction for that sample.

These models make a number of assumptions about the data to work properly. One of the most important is the assumption that each predictor variable (different Bs in the formula) is independent from the others, conditional on the class. This is called **conditional independence**. Variables B and C are independent of one another on the condition that a third variable, A, exists such that:

$$P(B|C,A) = P(B|A))$$

This equation can be interpreted as "the probability of B, given C and A, is equal to the probability of B, given A." In other words, given A, introducing C does not change the probability of B. Note that two features can only be considered conditionally independent of each other when considered in relation to a third variable. Furthermore, it's possible for variables B and C to be conditionally independent of one another with respect to A, but not with respect to another variable, say, Z.

In Naive Bayes, the predictor variables (B and C in the equation above) are assumed to be conditionally independent of each other, given the target variable (A). This is an assumption that very often is not actually true. However, Naive Bayes models still often perform well in spite of the data violating the assumption. The assumption is made to simplify the model. Otherwise, long probabilistic chains would have to be calculated to determine the probability of a feature's value with respect to the values of every other variable.

Another assumption of the data is that no predictor variable has any more predictive power than any other predictor. In other words, the individual predictor variables are assumed to contribute equally to the model's prediction. Like the assumption of class-conditional independence between the features, this assumption is also often violated by real-world data, but Naive Bayes still often proves a good model in spite of this.

Positives and negatives

Of all the classification algorithms that are still used today, Naive Bayes is one of the simplest. However, it is still able to produce valuable results. Its simplicity comes as an asset because it is one of the most straightforward algorithms to implement. In spite of their assumptions, Naive Bayes classifiers work quite well in many industry problems, most famously for document analysis/classification and spam filtering.

Additionally, the training time for a Naive Bayes model can sometimes be drastically lower than for other models because the calculations that are needed to make it work are relatively cheap in terms of computer resource consumption. This also means it is highly scalable and able to work with large increases in the amount of data it must handle.

One of the biggest problems with Naive Bayes is the data assumptions that were mentioned earlier. Few datasets have truly conditionally independent features—it is something that is very rare in the world today. However, Naive Bayes models can still perform well even if the assumption of conditional independence is violated.

Another issue that could arise is what is known as the “zero frequency” problem. This occurs when the dataset you’re using has no occurrences of a class label and some value of a predictor variable together. This would mean that there is a probability of zero. Since the final posterior probability is found by multiplying all of the individual probabilities together, the probability of zero would automatically make the result zero. Library implementations of the algorithms account for this by adding a negligible value to each variable count (usually 1) to ensure a non-zero probability.

Implementations in scikit-learn

There are several implementations of Naive Bayes in scikit-learn, all of which are found in the **sklearn.naive_bayes** module. Each is optimized for different conditions of the predictor variables. This reading will not delve into the mechanics of each variation. It is intended as a basic guide to using these models. Feel free to explore them on your own!

BernoulliNB:	Used for binary/Boolean features
CategoricalNB:	Used for categorical features
ComplementNB:	Used for imbalanced datasets, often for text classification tasks
GaussianNB:	Used for continuous features, normally distributed features
MultinomialNB:	Used for multinomial (discrete) features

Of course, datasets aren’t always limited to features of just a single type. In these cases, it’s often best to try the one that makes the most sense given your data. Often it’s useful to try several. It might also be the case that none of them work very well. Remember that modeling can be a messy process. Things will break. Assumptions will be violated. Nothing will be perfect, so don’t let perfect be the enemy of good. Careful planning, sound decision-making, and a lot of perseverance can go a long way.

Key Takeaways

- Naive Bayes is a classification technique that is based on Bayes’ Theorem.
- The model will calculate the posterior probability of an event, given the values of the predictor variables.
- This model assumes class-conditional independence of the predictor variables, which can sometimes lead to poor performance in conditions when this assumption is violated.
- Naive Bayes models can be good to use because they are relatively simple to create and are highly scalable depending on the business need.

- scikit-learn has different implementations of the algorithm, each optimized for different conditions of the data.

Resources for more information

- BernoulliNB: scikit-learn documentation for the BernoulliNB model implementation
- CategoricalNB: scikit-learn documentation for the CategoricalNB model implementation
- ComplementNB: scikit-learn documentation for the ComplementNB model implementation
- GaussianNB: scikit-learn documentation for the GaussianNB model implementation
- MultinomialNB: scikit-learn documentation for the MultinomialNB model implementation