Naive Bayes

Sylesh Suresh

December 2018

1 Introduction

Naive Bayes is a simple supervised classifier based on Bayes' theorem that, despite its assumption that there is independence between every pair of features of the input given its classification, tends to perform quite well.

2 Derivation of Bayes' Theorem

We can start off with one of the basic probability equalities:

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

This reads: the conditional probability of event A given that event B has occurred is equal to the probability of both event A and event B occurring divided by the probability of event B. What we want to find is a relationship between $P(A|B)$ and $P(B|A)$. You can imagine that finding this relationship would be useful in machine learning, as it would very nice to be able to calculate the probability of a data sample belonging to a certain class given the features of that data sample (which is what we'd like a machine learning model to be able to calculate) by using the probability that a data sample has certain features given that it belongs to a particular class (which is something we can calculate from training data).

So, getting back to the equation, if we multiply both sides by $P(B)$, we get:

$$P(A|B)P(B) = P(A \wedge B)$$

Note that $P(A \wedge B) = P(B \wedge A)$, and furthermore, $P(B \wedge A) = P(B|A)P(A)$. Thus,

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

Dividing both sides by B, we arrive at Bayes' theorem:

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

3 Bayes' Theorem

We can apply Bayes' Theorem to machine learning. Specifically, say we are given a data sample with n features, $x_1, x_2, x_3, ..., x_n$. We want to find the probability that this data sample belongs to a particular class y. In other words, we want to calculate $P(y|x_1 \wedge x_2 \wedge x_3 ... \wedge x_n)$. (We can use commas instead of the 'and' symbols for shorthand, writing the probability instead as $P(y|x_1, x_2, x_3, ..., x_n)$). Using a dataset of samples which are each labeled with the class y and features $x_1, ..., x_n$, we want to calculate this probability. We can apply Bayes' Theorem:

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)}$$

The numerator of the right-hand side of the equation can be condensed as $P(A|B)P(B) = P(A, B)$:

$$P(y|x_1, ..., x_n) = \frac{P(x_1, ..., x_n, y)}{P(x_1, ..., x_n)}$$

Using the same law $(P(A|B)P(B) = P(A, B)$, we can decompose the numerator like so:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|x_2, ..., x_n, y)P(x_2, ..., x_n, y)}{P(x_1, ..., x_n)}$$

Then, we can decompose the second factor of the numerator in a similar fashion:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|x_2, ..., x_n, y)P(x_2|x_3, ..., x_n, y)P(x_3, ..., x_n, y)}{P(x_1, ..., x_n)}$$

We can keep decomposing the last factor in the numerator until we arrive at:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|x_2, ..., x_n, y)P(x_2|x_3, ..., x_n, y)P(x_3|x_4, ..., x_n, y)P(x_4|x_5, ..., x_n, y)...P(x_n|y)P(y)}{P(x_1, ..., x_n)}$$

This is nice, but we usually cannot calculate the numerator directly if we are given an unseen sample. If we could, it would mean that we already have a sample in our training dataset with the exact same features $x_1, ..., x_n$. The goal is to be able to arrive at an expression that allows us to handle the general case.

At this point, we will have to make a (somewhat fallacious) assumption about the data in order to arrive at a meaningful expression. The assumption is that each of the features of the data are independent from every other feature. The probability that a certain feature takes on a certain value is independent of the values that other features take on. To give a concrete example, say we were trying to classify e-mails as being spam or not spam, with the nth feature being the nth word in the e-mail. Our assumption says, for example, that the probability of the presence of the word "Nigerian" is independent of the presence of the word "prince". This is definitely not true for some examples as you can see, which is why the classifier is called naive. It turns out, however, that the classifier tends to make fairly accurate predictions, even when applied to situations like detecting spam e-mail.

The reason we make this assumption is because we really don't like all the conditionals in each probability. This assumption will make all those conditionals go away - if a feature is independent of every other feature, then we can simply remove the features from the conditional part of the probability. For example, let's look at $P(x_1|x_2, ..., x_n, y)$. Our assumption says that feature x_1 is independent of features x_3, x_4, x_5 and so on. This means that $P(x_1|x_2, ..., x_n, y) = P(x_1|y)$ - we can take out all the other features from the conditional. Moreover, we can do this for every single probability with features in the conditional. This is useful because we can then simplify our equation to:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)P(x_3|y)...P(x_n|y)P(y)}{P(x_1, ..., x_n)}$$

We can use capital pi notation to write the n factors in the numerator:

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)}$$

This is the fundamental equation of naive Bayes. Specifically, when we are given a sample whose class is unknown with features $x_1, ..., x_n$, we can calculate $P(y|x_1, ..., x_n)$ for each possible class y and choose the class that returns the highest probability when plugged into the equation as being the correct classification for the sample.

Note that $P(x_1, ..., x_n)$ does not depend on y, so we can simply maximize $P(y)\prod_{i=1}^{n} P(x_i|y)$ with respect to y, which will yield our class prediction.

4 Continuous Features

The expression we derived above will work cleanly for categorical features; we can calculate $P(x_i|y)$ for any specific value of categorical feature x_i and specific class y by dividing the number of samples in our training dataset that have that categorical feature and class y by the number of samples with the class y. On the other hand, for continuous features, we must define $P(x_i|y)$ with a probability distribution. Continuous features are a completely different kind of beast. It is very likely that the samples in our dataset will have continuous features that take on values close to but not exactly equal to continuous features we will see in the real world. For example, say we are trying to predict what kind of flower a particular flower is based on the height of that flower. We may have a flower in our training dataset with a height of 30 cm. If we are asked to classify a flower with a height of 29.5 cm, it is likely that that flower will be of the same type of the 30 cm flower in our dataset. But the heights are not exactly the same. We don't have a flower with a height of 29.5 cm, so when we try to calculate $P(x_1|y)$, the probability will turn out to be 0, which seems like a pretty bad prediction. That is why we need to introduce a probability distribution, which will tell us that hey, 29.5 cm and 30 cm are pretty close, so it's likely that the 29.5 cm flower is of the same type as the 30 cm flower. One way to define this probability distribution is with the Gaussian distribution. If the training data has a continuous feature x_i, first, we will separate the training data samples based on its class y. For each set of samples with class y, we must calculate the mean μ_y and the variance σ_y^2. We can now define $P(x_i|y)$ as:

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

This allows us to solve the previously defined maximization problem to calculate our class prediction. Of course, assuming a normal distribution may not always be the best answer when dealing with continuous features, so there are other probability distributions we can define to increase the accuracy of our model, but for the sake of simplicity, the normal distribution is a decent approximation.