Introduction to Python for Data Science

Coding the Naive Bayes Algorithm

Dan Wiesenthal
dan.wiesenthal@galvanize.com

If you would like to follow along, head here:
https://github.com/danwiesenthal/Naive_Bayes_Evening_Workshop

Naive Bayes

galvanize

Naive Bayes Classifier: $\underset{Y}{\text{argmax}}\ P(Y|\vec{X}) = \underset{Y}{\text{argmax}}\ P(x_1|Y)P(x_2|Y)...P(x_3|Y)P(Y)$

$$= \underset{Y}{\text{argmax}}\ P(Y)\prod_{i=1}^{k} P(x_i|Y)$$

Quick Probability

P(a): The probability event a occurs

P(a,b): The probability events a and b both occur

P(a|b): The probability event a occurs, given that event b occurs

---

P(a,b) = P(b,a)

P(a,b) = P(a|b)P(b)

P(a|b) != P(b|a) (…why?)

Bayes Theorem

I want P(a|b), but I don't have it already.

How could I get it?

$$P(a,b) = P(b,a)$$

$$P(a|b)P(b) = P(b|a)P(a)$$

$$P(a|b) = P(b|a)P(a) \ / \ P(b)$$

posterior = likelihood * prior / evidence

Intuitive?

We have a test that's 99% "accurate" (99% Sensitive, 99% Specific). Only 0.5% of the population have the condition (say, for example, lactose intolerance).

We pick Sally at random, and we test her. She tests positive. Does she have the condition?

Answer: Probably not!

It's only 33% likely. It's more likely that she **doesn't** have the condition than that she does!

P(cond | pos)

= P(pos | cond) * P(cond) / P(pos)

= P(pos | cond) * P(cond) / (P(pos | cond) * P(cond) + P(pos | no-cond) * P(no-cond))

= 0.99 * 0.005 / (0.99 * 0.005 + 0.01 * 0.995)

≈ .33

Naive Bayes: Class | Feature

So for one feature:

P(class I feature) = P(feature I class) * P(class) / P(feature)

Don’t worry about the denominator for a minute, trust me...

= (# feat in class / # in class) * (# class / # total data points)

For multiple features:

P(feature I class) * P(class)

= P(c, F1, F2, ..., Fn) (“joint”)

= P(c) * P(F1 I c) * P(F2 I c, F1) * P(F3 I c, F1, F2) * ...

But if we assume independence of features given class... P(Fi I c, Fj) = P(Fi I c)

= P(c) * P(F1 I c) * P(F2 I c) * P(F3 I c) *...

Which is way more tractable. Sweet!

Naive Bayes: argmax(Class)

Now that we can calculate the numerator for **some** class, we calculate it for **every** class and compare them. The largest one is the most likely class, and therefore the class we should predict.

For each class c:

pseudo_prob_data_is_in_this_class = P(c) * P(F1 | c) * P(F2 | c) * P(F3 | c) *...

Prediction = the c which led to max pseudo_prob_data_is_in_this_class

The denominator stays the same for all classes, so we don't have to calculate it. Awesome.

"Training" is simply the process of counting up occurrences (all the little terms in "(# feat in class / # in class) * (# class / # total data points)") so we can use them to predict. (In some models/algorithms, training is much, much more complicated.)

galvanize

Thanks!

Dan Wiesenthal
dan.wiesenthal@galvanize.com