Looking Again at Predictions Using Naive Bayes

Recall the log odds form of Naive Bayes:

$$\log \frac{P(y=c \,|\, x_1,\ldots,x_p)}{P(y=c' \,|\, x_1,\ldots,x_p)} \;=\; \log \frac{P(y=c)}{P(y=c')} + \sum_{j=1}^{p} \log \frac{P(x_j \,|\, y=c)}{P(x_j \,|\, y=c')}$$

Let's rewrite this using $x_j = 0$ as a reference point, by defining

$$\beta_0 \;=\; \log \frac{P(y=c)}{P(y=c')} + \sum_{j=1}^{p} \log \frac{P(x_j=0 \,|\, y=c)}{P(x_j=0 \,|\, y=c')}$$

and

$$\beta_j \;=\; \log \frac{P(x_j=1 \,|\, y=c)}{P(x_j=1 \,|\, y=c')} - \log \frac{P(x_j=0 \,|\, y=c)}{P(x_j=0 \,|\, y=c')}$$

Now, if every x_j is binary (ie, only has values 0 and 1), we can write the log odds in a linear form:

$$\log \frac{P(y=c \,|\, x_1,\ldots,x_p)}{P(y=c' \,|\, x_1,\ldots,x_p)} \;=\; \beta_0 + \sum_{j=1}^{p} \beta_j \, x_j$$

What About Real-Valued Inputs?

That's a neat trick for binary inputs, but what about real-valued ones?

If we model $P(x_j \,|\, y = c)$ as Gaussian, with mean $\mu_{c,j}$ and standard deviation σ_j, the same for all classes, then

$$\begin{aligned}
\log \frac{P(x_j \,|\, y=c)}{P(x_j \,|\, y=c')} &= \log \frac{(2\pi\sigma_j^2)^{-1/2} \exp(-(x_j - \mu_{c,j})^2/2\sigma_j^2)}{(2\pi\sigma_j^2)^{-1/2} \exp(-(x_j - \mu_{c',j})^2/2\sigma_j^2)} \\
&= \log \frac{\exp(-(x_j^2 - 2x_j\mu_{c,j} + \mu_{c,j}^2)/2\sigma_j^2)}{\exp(-(x_j^2 - 2x_j\mu_{c',j} + \mu_{c',j}^2)/2\sigma_j^2)} \\
&= \log \frac{\exp(x_j\,\mu_{c,j} \,/\, \sigma_j^2)}{\exp(x_j\,\mu_{c',j} \,/\, \sigma_j^2)} + \log \frac{\exp(-\mu_{c,j}^2 \,/\, 2\sigma_j^2)}{\exp(-\mu_{c',j}^2 \,/\, 2\sigma_j^2)}
\end{aligned}$$

So if we define $\beta_j = (\mu_{c,j} - \mu_{c',j}) \,/\, \sigma_j^2$, and include the second term above in β_0, we once again can write

$$\log \frac{P(y=c \,|\, x_1,\ldots,x_p)}{P(y=c' \,|\, x_1,\ldots,x_p)} = \beta_0 + \sum_{j=1}^{p} \beta_j\, x_j$$

From Log Odds to Probabilities

The log odds for every class versus every other class determine all the class probabilities, remembering that they also must sum to one.

Let $\beta_{c,0}$ and $\beta_{c,j}$ for $j = 1, \ldots, p$ be the coefficients for the log odds of class c versus class 0. (So $\beta_{0,0} = 0$ and $\beta_{0,j} = 0$ for $j = 1, \ldots, p$.)

The Naive Bayes probabilities can then be written as

$$P(y = c \,|\, x_1, \ldots, x_p) \;\;=\;\; \frac{\exp\left(\beta_{c,0} \;+\; \sum_{j=1}^{p} \beta_{c,j}\, x_j\right)}{\sum_{c'=0}^{C-1} \exp\left(\beta_{c',0} \;+\; \sum_{j=1}^{p} \beta_{c',j}\, x_j\right)}$$

When there are two classes, this simplifies — abbreviating $\beta_{1,j}$ to β_j, we have

$$P(y = 1 \,|\, x_1, \ldots, x_p) \;\;=\;\; 1 \Big/ \left[1 \;+\; \exp\left(-\left(\beta_0 \;+\; \sum_{j=1}^{p} \beta_j\, x_j\right)\right)\right]$$

Though βs for class 0 were all set to zero above, they don't have to be. Adding any constant to $\beta_{c,0}$ for all c leaves the probabilities unchanged. Similarly, probabilities wouldn't change if for any j, we added a constant to $\beta_{c,j}$ for all c.

From Naive Bayes to Logistic Regression

In these two common situations, predictions from Naive Bayes depend on a linear combination of the inputs, $\beta_0 + \sum_{j=1}^{p} \beta_j x_j$. The β values are chosen (indirectly) using models for $P(x_j \,|\, y = c)$, and assuming the x_j are independent given y.

If these models, or the independence assumption, are wrong, the β values found could be far from optimal. We might instead choose β values that produce conditional probabilities, $P(y \,|\, x_1, \ldots, x_p)$, that fit the training cases well.

This is known as *logistic regression*. It assumes that the log odds for class conditional probabilities are linear functions of the inputs. This could be true when the Naive Bayes assumptions are false.

Maximum Likelihood Estimation for Logistic Regression

The *likelihood* for a set of parameters is the probability that the model gives to the training data when those parameters are used. For supervised learning — regression and classification — we look only at the conditional probability of training targets given training inputs.

I'll usually assume that training cases are independent, so the likelihood is a product over training cases.

The *maximum likelihood* parameter values are the ones that maximize the likelihood. These estimates have some good properties. For most models, they converge to the correct parameter values as the training set gets bigger.

Maximizing the log of the likelihood is easier. For binary logistic regression, it is

$$\sum_{i=1}^{n} \log\, P(y^{(i)} \,|\, x_1^{(i)}, \ldots, x_p^{(i)})$$

$$= \sum_{i=1}^{n} \left\{ \begin{array}{ll} -\;\log\Big(1 \;+\; \exp\Big(-\beta_0 - \sum_{j=1}^{p} \beta_j\, x_j^{(i)}\Big)\Big) & \text{if } y^{(i)} = 1 \\ -\;\log\Big(1 \;+\; \exp\Big(\beta_0 \;+\; \sum_{j=1}^{p} \beta_j\, x_j^{(i)}\Big)\Big) & \text{if } y^{(i)} = 0 \end{array} \right\}$$

Finding the Maximum Likelihood Estimate

How can we find the maximum likelihood estimate for the parameters — ie, for $\beta_0, \beta_1, \ldots, \beta_p$ when the target is binary, or for $\beta_{1,0}, \beta_{2,0}, \ldots, \beta_{C-1,0}, \beta_{1,1}, \ldots$ when there are $C > 2$ classes?

There's no analytical solution, so an iterative optimization method must be used, that successively improves the solution from some starting point.

This is a relatively easy optimization problem — there is only on local maximum, and furthermore the log likelihood surface is concave. However, if the classes can be perfectly separated by a hyperplane (or a Voronoi tesselation for $C > 2$), the likelihood can be pushed to 1 by pushing the βs to infinity. This will likely be an "overfitted" estimate that won't work well on test cases.

Many optimization algorithms could be used. The traditional one pretends this is a least-squares regression problem, with each training case having a different residual variance. These variances are chosen to mimic the second derivatives of the log likelihood at the current estimate. When the next estimate is found, new variances are computed, another iteration is done, and so forth. This is called "Iteratively Reweighted Least Squares" (IRLS).

Naive Bayes vs. Maximum Likelihood Logistic Regression

Naive Bayes doesn't account for dependencies between inputs for cases within a class, but logistic regression can.

Example: Suppose we add an input, j', identical to an existing input, j. Naive Bayes will count that as more information, so probabilities will change to be closer to 0 or 1. Logistic regression will realize the new input is redundant. The new estimate for $\beta_j + \beta_{j'}$ will be the same as the old estimate for β_j alone.

Logistic regression with parameters found by maximum likelihood will fail if $p > n$, and often if p is close to n. With more inputs than training cases, getting a perfect fit to the training data will be possible, but this "perfect" fit will probably not work well on test cases. Naive Bayes may still work in this situation.

To solve this problem, we need to modify maximum likelihood — either add a "penalty" term to the log likelihood, or use a Bayesian method, which averages over parameter values rather than picking one.

Maximum Penalized Likelihood Estimation

Maximum likelihood estimates are usually good with lots of training data, but may not be so good with little data. For logistic regression, $\beta_j \to \infty$ is a problem.

One solution: Add a *penalty function* to the log likelihood to encourage the βs to stay fairly small. The estimates are then a compromise between fitting the training data and keeping βs small.

Some possible penalties (for binary logistic regression):

$$P_2(\beta) \;=\; -\lambda \sum_{j=1}^{p} \beta_j^2, \qquad P_1(\beta) \;=\; -\lambda \sum_{j=1}^{p} |\beta_j|, \qquad P_t(\beta) \;=\; -\lambda \sum_{j=1}^{p} \log(1+\beta_j^2/\nu^2)$$

We usually wouldn't penalize β_0. P_1 often makes some of the β_j exactly zero, so the corresponding input is completely ignored. These penalty functions have parameters (λ and ν) that we somehow have to set. Plots with $\lambda = \nu = 1$: