1 Naive Bayes Classification

The naive-Bayes classifier is a simple but powerful machine learning tool. Though the model presumes class-conditioned independence between the observed features in the data, and thus ignores many helpful structures in the data, its results are often comparable to those achieved by considerably more complicated algorithms. Combine this with the fact that these independence assumptions greatly increase the feasibility bounds on problem size, and the naive Bayes classifier remains a popular choice for decision rules.

This paper concerns itself with a particular data domain: each point has n features, each of which is either 0 or 1. This point belongs to one of two classes. Classification of an unlabeled point is achieved by examining the point's features and comparing the odds of seeing such a pattern under class 0 to the odds under class 1.

Before the classifier is ready to make such a judgement, however, it must be primed with these class-dependent probabilities. This is accomplished through the provision of a training set. The Bernoulli parameter $p(f_i = 1|C = c) \equiv \theta_i^c$, the odds of feature i being 1 under class c, is found by maximizing the following likelihood function[1] across the N data point features available, x_j^c for $j = 1, 2, \ldots, N$:

$$L\left(\theta_i^c|x^c\right) = \prod_{j=1}^{N} \left(\theta_i^c\right)^{x_{ji}^c} \left(1 - \theta_i^c\right)^{\left(1-x_{ji}^c\right)}$$

Note the independence assumptions kick in here, whereby each parameter can be estimated alone.

The overall likelihood across all dimensions becomes:

$$L\left(\theta^c|x^c\right) = \prod_{i=1}^{n} \prod_{j=1}^{N} \left(\theta_i^c\right)^{x_{ji}^c} \left(1 - \theta_i^c\right)^{\left(1-x_{ji}^c\right)}$$

And under the monotonic log function, the same maximizing arguments are produced by:

$$\begin{aligned} logL\left(\theta^c|x\right) &= \sum_{i=1}^{n} \sum_{j=1}^{N} x_{ji}^c log\left(\theta_i^c\right) + \left(1 - x_{ji}^c\right) log\left(1 - \theta_i^c\right) \\ &= \sum_{i=1}^{n} \left(\sum_{j=1}^{N} x_{ji}^c \right) log\left(\theta_i^c\right) + \left(N - \sum_{j=1}^{N} x_{ji}^c \right) log\left(1 - \theta_i^c\right) \end{aligned}$$

[1] Bayesian approaches are also valid here, where a prior assumption of the Bernoulli parameter's distribution is made, only to be slowly refined as more data is made available.

As the argument of maximization is invariant to division by N, we can simplify to:

$$max_{\theta^0,\theta^1} \sum_{i=1}^{n} \mu_i^0 log\theta_i^0 + \left(1-\mu_i^0\right) log\left(1-\theta_i^0\right) + \sum_{i=1}^{n} \mu_i^1 log\theta_i^1 + \left(1-\mu_i^1\right) log\left(1-\theta_i^1\right)$$

It can be shown that this is maximized for the class's sample mean along dimension i:

$$\theta_i^{c*} = \frac{1}{N} \sum_{j=1}^{N} x_{ji}^c \equiv \mu_i^c$$

This estimation of the parameters leads to the next step of classification. The algorithm now has a firm grasp of θ_i^0 and θ_i^1 for each feature i. Good classification involves picking out which features are most meaningful in distinguishing one class from the other. We use the log of the ratio of the two class probabilites:

$$w_i \equiv log\left(\frac{\theta_i^0}{\theta_i^1}\right), v_i \equiv log\left(\frac{1-\theta_i^0}{1-\theta_i^1}\right)$$

When the two probabilities are equal, w_i is zero. This indicates the appearance of a 1 along that dimension is worthless – it's no more a sign of a class 1 point than a class 0 point. However, when the probabilities diverge, w_i moves towards positive or negative infinity (indicating 1's are more common for feature i in class 0 or class 1, respectively). This gives a proximate measure of how 'important' a feature is in terms of predicting a data label.

The actual classification takes this w vector and runs with it. An unlabelled point accumulates w_i 'points' for every dimension i that is a 1, and v_i if dimension i is a 0. The sign of the final sum of this number designates whether the point should be treated as a member of class 0 (positive sum), or class 1 (negative).

2 Sparsity

Given the capricious nature of statistics, it's quite unlikely that any w_i comes out to a value of zero. In this sense, the naive Bayes classifier considers every dimension somewhat useful, pushing the final judgment one way or the other. A simple way to pare down the number of non-zero weights is to pick the top k weights in order of largest magnitude $|w_i|$.

To encourage sparsity directly, the procedure needs to return to the estimation stage. The current estimation of θ^{0*}/ bases its selection on maximizing the likelihood of observing the sample mean μ^0. There's no downside to producing a non-zero w_i. If we desire $w_i = 0$, then we should counteract the benefit of more accuracte explanation of the data with a penalty proportional to $|\theta_i^0 - \theta_i^1|$. Now, the parameters will only diverge in cases where the increased explanatory power exceeds this penalty. Ramping up the proportional cost of divergence will cull greater and greater numbers of w_i's.

This optimization now becomes:

$$max_{\theta^0,\theta^1} \sum_{i=1}^{n} \mu_i^0 log\theta_i^0 + \left(1-\mu_i^0\right) log\left(1-\theta_i^0\right) + \sum_{i=1}^{n} \mu_i^1 log\theta_i^1 + \left(1-\mu_i^1\right) log\left(1-\theta_i^1\right) - \lambda \left|\theta^0 - \theta^1\right|_1$$

This can be solved in closed form. The absolute value function is not differentiable at 0, meaning it must be tested in two parts: first assuming divergence, second assuming convergence. The stationary values of these two hypotheses can be plugged into the penalized-likelihood equation, and the two results can be compared. Whichever scores higher determines the value of w_i. Let $s = sgn(\mu_i^0 - \mu_i^1)$. Under divergence, we have:

$$\begin{aligned} \theta_i^{0*} &= \frac{\lambda s + 1 - \sqrt{(\lambda s + 1)^2 - 4\lambda s \mu_i^0}}{2\lambda s} \\ \theta_i^{1*} &= \frac{\lambda s - 1 + \sqrt{(\lambda s - 1)^2 + 4\lambda s \mu_i^1}}{2\lambda s} \end{aligned}$$

Under convergence, we have:

$$\theta_i^{0*} = \theta_i^{1*} = \frac{\mu_i^0 + \mu_i^1}{2}$$

Notice that again the calculation of weight w_i is independent of the other dimensions. This sparse classifier requires the same order of computation, $O(n)$, as the threshold method above.

3 Testing

Data simulated in MATLAB reveals the l1-regularized optimization results in feature selection that differs from simple thresholding. In Figure 1, we see the magnitude of weights, $|w_i|$, for a sweep of the regularization parameter λ.

leave one out