Machine Learning, Spring 2019
EN 2105
MWF 9-9:50am

Chao Lan
EN 4084A
clan@uwyo.edu

Bayes Decision Rule and Bayes Error

We will introduce several classic and matured classification methods: logistic regression, Gaussian discriminant analysis (GDA), Naive Bayes (NB), k-nearest neighbor (kNN), support vector machine (SVM), decision tree and artificial neural network (ANN). Four of the classifiers adopt the Bayes decision rule, which assigns x to class k if this class has the highest posterior, i.e.,

$$p(y = k \mid x) \geq p(y = c \mid x), \ \forall k \neq c. \tag{1}$$

This rule minimizes the Bayes error. Assume $y \in \{1, 2\}$. If the true y is known, we can directly measure the classification error on x (e.g., using 0-1 loss). But if the true y is unknown, we can measure the 'probability' of the error, i.e., the bayes error on x is

$$er(x) = \begin{cases} p(y = 1 \mid x), & \text{if } x \text{ is assigned to class 2} \\ p(y = 2 \mid x), & \text{if } x \text{ is assigned to class 1} \end{cases} \tag{2}$$

The bayes error (averaged over the entire population) is

$$er = \int_{D(x \to C_1)} p(y = 2 \mid x)p(x)dx + \int_{D(x \to C_2)} p(y = 1 \mid x)p(x)dx, \tag{3}$$

where $D(x \to C_k)$ is the set of x assigned to class k.

[*Discussion*] How to interpret the Bayes error?

[*Discussion*] How does the Bayes decision rule minimize the Bayes error?

We will see logistic regression, GDA, NB and kNN adopt (or, can be interpreted as adopting) the bayes decision rule. They differ in the ways of constructing or learning the posterior.

Logistic Regression

Logistic regression directly constructs $p(y \mid x)$. It assumes binary classification (i.e., $y \in \{0, 1\}$) and constructs

$$p(y = 0 \mid x; \beta) = \frac{1}{1 + \exp(-x^T \beta)} = \frac{\exp(x^T \beta)}{1 + \exp(x^T \beta)}, \tag{4}$$

and

$$p(y = 1 \mid x; \beta) = 1 - P(y = 1 \mid x; \beta) = \frac{1}{1 + \exp(x^T \beta)}, \tag{5}$$

where β is the unknown parameter.[1] Now estimating $p(y \mid x)$ is equivalent to learning β.

[*Discussion*] Why bother to construct β? Can we directly estimate $p(y \mid x)$?

[1]It is called logistic regression because $\sigma(a) = \frac{1}{1+\exp(-a)}$ is known as the logistic sigmoid function.

We can apply MLE to learn β. The log-likelihood function of β over a sample $y_1, \ldots, y_n$ is

$$\begin{aligned} L_n(\beta) &= \sum\nolimits_{i=1}^{n} \log p(y_i \mid x_i; \beta) \\ &= \sum\nolimits_{i=1}^{n} (1-y_i) \log p(y_i = 0 \mid x_i; \beta) + y_i \log p(y_i = 1 \mid x_i; \beta) \\ &= \sum\nolimits_{i=1}^{n} (1-y_i)(x_i^T \beta) - \log(1 + \exp(x_i^T \beta)) \end{aligned} \tag{6}$$

[*Exercise*] Derive the second equation in (6). It takes a trick to rewrite the log posterior.

[*Exercise*] Derive the third equation in (6).

For convenience we will work with an equivalent minimization problem, and minimizes

$$G_n(\beta) = -L_n(\beta) = \sum\nolimits_{i=1}^{n} (y_i - 1)(x_i^T \beta) + \log(1 + \exp(x_i^T \beta)). \tag{7}$$

The critical point method does not work here, because $G_n'(\beta) = 0$ is hard to solve. Indeed,

$$\begin{aligned} G_n'(\beta) &= \sum\nolimits_{i=1}^{n} (y_i - 1) \cdot x_i + \frac{\exp(x_i^T \beta)}{1 + \exp(x_i^T \beta)} \cdot x_i \\ &= \sum\nolimits_{i=1}^{n} \left(y_i - \frac{1}{1 + \exp(x_i^T \beta)} \right) \cdot x_i \\ &= \sum\nolimits_{i=1}^{n} [y_i - p(y_i = 1 \mid x_i; \beta)] \cdot x_i. \end{aligned} \tag{8}$$

Instead, we can apply numerical optimization methods. An example is the Newton's method. It finds an approximate solution by iteratively shifting the current solution β_{old} by an amount of $\Delta\beta$ so that $\beta_{old} + \Delta\beta$ minimizes the second-order Taylor approximation of $G_n(\beta)$ at β_{old}, i.e.,

$$G_n(\beta) \approx G_n(\beta_{old}) + (\Delta\beta)^T G_n'(\beta_{old}) + \frac{1}{2} (\Delta\beta)^T G_n''(\beta_{old}) \, \Delta\beta = \hat{G}_n(\beta), \tag{9}$$

where $G_n''(\beta) \in \mathbb{R}^{(p+1)\times(p+1)}$ is the Hessian matrix.

Note $\hat{G}_n(\beta)$ is a quadratic function of $\Delta\beta$, and can be optimized by the critical point method.

$$\frac{\partial \hat{G}_n(\beta)}{\partial \Delta\beta} = G_n'(\beta_{old}) + G_n''(\beta_{old}) \, \Delta\beta. \tag{10}$$

Solving $\frac{\partial \hat{G}_n(\beta)}{\partial \Delta\beta} = 0$ for $\Delta\beta$ gives the update rule of $\Delta\beta$ at one iteration:

$$\Delta\beta = -\left(G_n''(\beta_{old}) \right)^{-1} G_n'(\beta_{old}). \tag{11}$$

We can write $\Delta\beta$ in a matrix form. Let $\vec{p} = [\, p(y_1 = 1 \mid x_1; \beta), \ldots, p(y_n = 1 \mid x_n; \beta) \,]^T$ and W be an n-by-n diagonal matrix with $W_{ii} = p(y_i = 1 \mid x_i; \beta) \cdot p(y_i = 0 \mid x_i; \beta)$. From (8) we have

$$G_n'(\beta) = X^T (Y - \vec{p}) \tag{12}$$

and

$$G_n''(\beta) = X^T W X. \tag{13}$$

[*Exercise*] Derive (12) and (13).

Combining (11) (12) (13) gives the Newton's method for logistic regression in Algorithm 1.

[*Discussion*] What is the geometric interpretation of Newton's method?

[*Discussion*] How is Newton's method different from the gradient descent method?

Algorithm 1 Newton's Method for Logistic Regression

```
0: initialize $\beta \in \mathbb{R}^{(p+1)}$
while stopping criterion is not met (e.g. max iteration number or $\Delta\beta < \epsilon$) do
    1: compute $G'_n(\beta)$ based on (12)
    2: compute $G''_n(\beta)$ based on (13)
    3: update $\beta$ based on (11), that is,
        $$\beta = \beta - \left(G''_n(\beta)\right)^{-1} G'_n(\beta) = \beta - (X^T W X)^{-1} X^T (Y - \vec{p}). \quad (14)$$
end while
```

Logistic Regression for Multi-Class Classification

There are two ways to extend logistic regression to more than two classes. One is to use one β for each class in posterior construction[2]. Another is the one-versus-one or one-versus-all strategies, both of which can be applied to extend any binary classifier for multi-class classification tasks.

Suppose the task is to classify examples into K classes $1, 2, \ldots, K$. One-versus-one decomposes this task into $\binom{K}{2}$ binary classification subtasks, each separating one class from another class, e.g., (1 vs 2), (3 vs 5). Then it applies any binary classifier on each subtask, and aggregates results by voting, e.g., if class 3 gets 1 vote and class 5 gets 3 (the highest) , then assign x to class 5. One-versus-all decomposes this task into K binary classification subtasks, each separating one class from the rest, e.g., 1 vs $\{2 \vee 3 \vee \ldots \vee K\}$. Then it applies any binary classifier on each subtask and aggregate results by voting (a vote on the rest classes is a vote on each of them).

[2]See [PRML, Chapter 4.3.4] for details.