Notes for *Machine Learning*

Fei Li*

*Email: fei.li.best@gmail.com.

Contents

1 The Bias-Variance Trade-off

In James et al. 2014, the model of the data as

$$Y = f(X) + \epsilon.$$

The irreducible error ϵ represents something that cannot be captured by variables in X, no matter how complicated f is. The goal of statistical learning is to obtain a good approximation $\hat{f}$ of f, using training data available. The estimation, $\hat{f}$, can have *variance* and *bias*.

- *Variance* refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set. Since the training data are used to fit the statistical learning method, different training data sets will result in a different $\hat{f}$. But ideally the estimate for f should not vary too much between training sets. However, if a method has high variance then small changes in the training data can result in large changes in $\hat{f}$. In general, more flexible statistical methods have higher variance.
- *Bias* refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model. The definition is $\text{bias}(\hat{f}) = \mathbb{E}(\hat{f}) - f$. For example, linear regression assumes that there is a linear relationship between Y and $X_1, \ldots, X_p$. It is unlikely that any real-life problem truly has such a simple linear relationship, and so performing linear regression will undoubtedly result in some bias in the estimate of f.

The mean squared error over training data $\texttt{Tr} = \{(x_i, y_i)\}_{i=1}^n$ is

$$\text{MSE}_{\texttt{Tr}} = \text{Ave}_{i \in \texttt{Tr}}[y_i - \hat{f}(x_i)]^2.$$

The mean squared error over *test data* $\texttt{Te} = \{(x_i, y_i)\}_{i=1}^m$ is

$$\text{MSE}_{\texttt{Te}} = \text{Ave}_{i \in \texttt{Te}}[y_i - \hat{f}(x_i)]^2.$$

The expected test MSE can be decomposed as variance of the estimation, plus bias squared, plus the variance of the irreducible error:

$$\mathbb{E}\left(y - \hat{f}(x)\right)^2 = \text{var}(\hat{f}(x)) + \text{bias}(\hat{f}(x))^2 + \text{var}(\epsilon)$$

where (x, y) is an unseen example, and we assumed that the error ϵ has mean zero and fixed variance $\text{var}(\epsilon)$, so that $\mathbb{E}(y) = f(x)$. Suppose for a moment that the estimate is unbiased, and there is also no irreducible error. Then the mean squared error would be solely due to the variance in our estimate. In general, three factors can contribute to the test MSE: the

variance of the estimate, how well our function $\hat{f}$ approximate the true f on average, and errors that are out of our control.

A less flexible model typically has more bias, e.g. a constant estimate, but these models can also have low variance, i.e. they do not have sensitive response to different training sets. Conversely, a more flexible model typically has more variance and less bias. Depending on the true data-generating process,as we increase model flexibility three cases can happen:

(1) Typically, the variance should increase and the bias decrease, but as the model becomes over-flexible, there is little further reduction in bias, but the variance can continue to increase. This results in a U-shaped test MSE curve as a function of flexibility.

(2) If the true model is linear, then as we increase flexibility starting from linear model, there can be almost no gain in bias reduction, but the variance continue to increase. This results in an upper-shaped test MSE curve.

(3) If the true model is highly non-linear, then as we increase flexibility there can be dramatic reduction in bias, so that the test MSE can be L-shaped.

2 Linear Methods for Regression

2.1 Linear Regression

In linear regression, we assume a linear relationship between inputs and outputs: $y = \boldsymbol{\beta}^T \boldsymbol{x}$. We choose $\boldsymbol{\beta}$ so as to minimize the RSS

$$\ell(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 \tag{1}$$

where $\boldsymbol{y}$ is the n-dimensional response data, $\boldsymbol{X}$ is the $n \times p$ feature data. The gradient and Hessian of $\ell(\boldsymbol{\beta})$ are

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}), \qquad \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = 2\boldsymbol{X}^T\boldsymbol{X}.$$

Assume $\boldsymbol{X}^T\boldsymbol{X}$ is positive definite ($\Leftrightarrow \boldsymbol{X}$ is full-rank $\Leftrightarrow \boldsymbol{X}^T\boldsymbol{X}$ is full-rank), and let $\partial \ell(\boldsymbol{\beta})/\partial \boldsymbol{\beta} = 0$ we get the solution

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

Let $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$. We see that $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{H}\boldsymbol{y}$, so the predicted value for each response in the training data is a linear combination of all the responses. $\boldsymbol{H}$ is the projection matrix, which projects $\boldsymbol{y}$ into the space spanned by columns of $\boldsymbol{X}$. Note that $\ell(\boldsymbol{\beta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$ is the squared norm between the two vectors $\boldsymbol{y}$ and $\boldsymbol{X}\boldsymbol{\beta}$, so we are minimizing the distance between them. The solution is the projection, namely $\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}$ should be perpendicular to columns of $\boldsymbol{X}$, so we should have $\boldsymbol{X}_i \cdot (\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}) = 0$ for each column $\boldsymbol{X}_i, i = 1, \ldots, p$ of $\boldsymbol{X}$, so $\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}) = 0$.

2.1.1 Online least squares

We want to recompute the parameters $\boldsymbol{\beta}$ as new data point $(\boldsymbol{x}_{n+1}, y_{n+1})$ arrives, without appending the new data point to $\boldsymbol{X}$ and re-applying the formula. Let

$$\boldsymbol{X}_{(n+1)} = \begin{pmatrix} \boldsymbol{X}_{(n)} \\ \boldsymbol{x}_{n+1}^T \end{pmatrix}, \quad \boldsymbol{y}_{(n+1)} = \begin{pmatrix} \boldsymbol{y}_{(n)} \\ y_{n+1} \end{pmatrix}, \quad \boldsymbol{X}_{(n+1)}^T\boldsymbol{X}_{(n+1)} = \left(\boldsymbol{X}_{(n)}^T\boldsymbol{X}_{(n)} + \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T\right).$$

Using the Sherman–Morrison formula

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

we have

$$\begin{aligned} \left(\boldsymbol{X}_{(n+1)}^T\boldsymbol{X}_{(n+1)}\right)^{-1} &= \left(\boldsymbol{X}_{(n)}^T\boldsymbol{X}_{(n)} + \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T\right)^{-1} \\ &= \left(\boldsymbol{X}_{(n)}^T\boldsymbol{X}_{(n)}\right)^{-1} - \frac{(\boldsymbol{X}_{(n)}^T\boldsymbol{X}_{(n)})^{-1}\boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T\left(\boldsymbol{X}_{(n)}^T\boldsymbol{X}_{(n)}\right)^{-1}}{1 + \boldsymbol{x}_{n+1}^T\left(\boldsymbol{X}_{(n)}^T\boldsymbol{X}_{(n)}\right)^{-1}\boldsymbol{x}_{n+1}}. \end{aligned} \tag{2}$$

We let γ_n denote one over the denominator in Eq. (2). The formula for $\boldsymbol{\beta}_{(n+1)}$ is

$$
\begin{aligned}
\boldsymbol{\beta}_{(n+1)} &= \left(\boldsymbol{X}_{(n+1)}^T \boldsymbol{X}_{(n+1)}\right)^{-1} \boldsymbol{X}_{(n+1)}^T \boldsymbol{y}_{(n+1)} \\
&= \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)} + \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T\right)^{-1} \left(\boldsymbol{X}_{(n)}^T \boldsymbol{y}_{(n)} + \boldsymbol{x}_{n+1} y_{n+1}\right) \\
&= \left[\left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} - \gamma_n \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1}\right] \left(\boldsymbol{X}_{(n)}^T \boldsymbol{y}_{(n)} + \boldsymbol{x}_{n+1} y_{n+1}\right) \\
&= \underbrace{\left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{X}_{(n)}^T \boldsymbol{y}_{(n)}}_{\boldsymbol{\beta}_{(n)}} + \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1} y_{n+1} \\
&\quad - \gamma_n \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T \underbrace{\left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \left(\boldsymbol{X}_{(n)}^T \boldsymbol{y}_{(n)}\right)}_{\boldsymbol{\beta}_{(n)}} \\
&\quad - \gamma_n \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1}\boldsymbol{x}_{n+1}^T \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1} y_{n+1} \\
&= \boldsymbol{\beta}_{(n)} + \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1} \left(\underbrace{y_{n+1} - \gamma_n \boldsymbol{x}_{n+1}^T \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1} y_{n+1}}_{= \gamma_n y_{n+1} \text{ according to the definition of } \gamma_n} - \gamma_n \boldsymbol{x}_{n+1}^T \boldsymbol{\beta}_{(n)} \right) \\
&= \boldsymbol{\beta}_{(n)} + \boldsymbol{\kappa}_n (y_{n+1} - \boldsymbol{x}_{n+1}^T \boldsymbol{\beta}_{(n)})
\end{aligned}
$$

with $\boldsymbol{\kappa}_n = \gamma_n \left(\boldsymbol{X}_{(n)}^T \boldsymbol{X}_{(n)}\right)^{-1} \boldsymbol{x}_{n+1}$. Thus we can adjust the parameters by the prediction error of the new point.

2.2 Ridge Regression

In ridge regression, the objective function to be minimized is

$$\ell(\boldsymbol{\beta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|^2. \tag{3}$$

We can view $\ell(\boldsymbol{\beta})$ as a Lagrangian, so that minimizing Eq. (5) is equivalent to

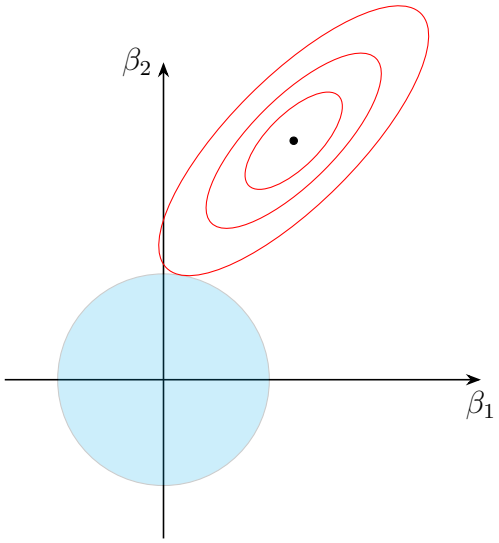
$$\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 \qquad s.t. \quad \|\boldsymbol{\beta}\|^2 \leq s \tag{4}$$

for some $s \geq 0$. For every $\lambda \geq 0$ there is a $s \geq 0$ for which the two minimization problems are equivalent. Thus we are minimizing a quadratic function over a closed ball of radius $\sqrt{s}$ centered at the origin. See Fig. 1a for illustration. We see that the constrained optimization indeed shrinks the coefficients toward the origin.

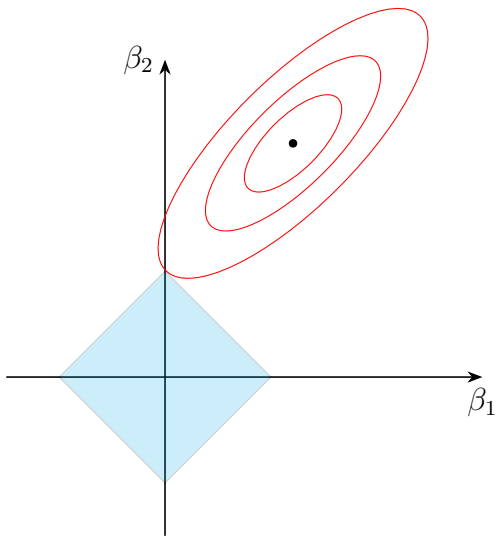
The gradient and Hessian of $\ell(\boldsymbol{\beta})$ are

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + 2\lambda \boldsymbol{I}_p \boldsymbol{\beta}, \qquad \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = 2\boldsymbol{X}^T\boldsymbol{X} + 2\lambda \boldsymbol{I}_p.$$

Equating the gradient to zero we obtain the ridge coefficients



(a) Illustration of the Ridge regression. The constraint is $\|\boldsymbol{\beta}\|_2^2 \leq s$.



(b) Illustration of the Lasso regression. The constraint is $\|\boldsymbol{\beta}\|_1 \leq s$.

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}_p)\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y} \quad \Rightarrow \quad \hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}_p)^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

We see that the term $\lambda \boldsymbol{I}_p$ is added to the denominator compared to OLS, so indeed $\boldsymbol{\beta}$ is now smaller.

2.3 Lasso Regression

In lasso regression, the objective function to be minimized is

$$\ell(\boldsymbol{\beta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1. \tag{5}$$

where $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^p |\beta_j|$. We see from Fig. 1b that since the constraint is a polygon, lasso regression may force some of the coefficients to be exactly zero. We comment that there is no closed-form formula for the solution, and to train the model we have to rely on algorithms such as coordinate descent.

It is also possible to give a Bayesian interpretation of ridge and lasso. Recall the posterior of the parameter $\boldsymbol{\beta}$ given data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ is

$$\begin{aligned} p\left(\boldsymbol{\beta} \mid \{\boldsymbol{x}_i, y_i\}_{i=1}^n\right) &= \frac{p(\boldsymbol{\beta}) p\left(\{\boldsymbol{x}_i, y_i\}_{i=1}^n \mid \boldsymbol{\beta}\right)}{p\left(\{\boldsymbol{x}_i, y_i\}_{i=1}^n\right)} = \frac{p(\boldsymbol{\beta}) p\left(\{y_i\}_{i=1}^n \mid \{\boldsymbol{x}_i\}_{i=1}^n, \boldsymbol{\beta}\right)}{p\left(\{y_i\}_{i=1}^n \mid \{\boldsymbol{x}_i\}_{i=1}^n\right)} \\ &= \frac{p(\boldsymbol{\beta}) \prod_{i=1}^n p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\beta})}{p\left(\{y_i\}_{i=1}^n \mid \{\boldsymbol{x}_i\}_{i=1}^n\right)} \propto p(\boldsymbol{\beta}) \prod_{i=1}^n p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\beta}). \end{aligned}$$

Assuming $p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\beta})$ is Gaussian with mean $\boldsymbol{x}_i^T\boldsymbol{\beta}$ and variance σ^2,

- if we assume independent uniform priors $\beta_j \sim U(-M, M)$ for $j = 1, \ldots, p$ then the density of β_j is constant, so

$$\begin{aligned}\arg\max_{\boldsymbol{\beta}} p\left(\boldsymbol{\beta} \mid \{\boldsymbol{x}_i, y_i\}_{i=1}^n\right) &= \arg\max_{\boldsymbol{\beta}} \left[\text{constant} \cdot \exp\left\{-\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2/2\sigma^2\right\}\right] \\ &= \arg\min_{\boldsymbol{\beta}} \left[\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2\right].\end{aligned}$$

  We get the OLS estimates of the coefficients.

- if we assume independent *Gaussian* priors $\beta_j \sim \mathcal{N}(0, c)$ for $j = 1, \ldots, p$ then

$$\begin{aligned}\arg\max_{\boldsymbol{\beta}} p\left(\boldsymbol{\beta} \mid \{\boldsymbol{x}_i, y_i\}_{i=1}^n\right) &= \arg\max_{\boldsymbol{\beta}} \left[\exp\left\{-\sum_{j=1}^p \beta_j^2/2c\right\} \cdot \exp\left\{-\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2/2\sigma^2\right\}\right] \\ &= \arg\min_{\boldsymbol{\beta}} \left[\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + (\sigma^2/c)\sum_{j=1}^p \beta_j^2\right]\end{aligned}$$

  so that we recover the ridge regression.

- if we assume independent *Laplace* priors $p(\beta_j) \sim L(0, b)$ with $p(\beta_j) = e^{-\sum_{j=1}^p |\beta_j|/b}$ for $j = 1, \ldots, p$, then we will get

$$\begin{aligned}\arg\max_{\boldsymbol{\beta}} p\left(\boldsymbol{\beta} \mid \{\boldsymbol{x}_i, y_i\}_{i=1}^n\right) &= \arg\max_{\boldsymbol{\beta}} \left[\exp\left\{-\sum_{j=1}^p |\beta_j|/b\right\} \cdot \exp\left\{-\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2/2\sigma^2\right\}\right] \\ &= \arg\min_{\boldsymbol{\beta}} \left[\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + (2\sigma^2/b)\sum_{j=1}^p |\beta_j|\right]\end{aligned}$$

  so that we recover the lasso regression.

3 Linear Methods for Classification

3.1 Logistic Regression

We first model the probability of the two labels, so we represent them as $\{0, 1\}$ instead of $\{-1, 1\}$. In logistic regression, we model $p(\boldsymbol{x}) = \mathbb{P}\{y = 1 \mid \boldsymbol{x}\} = 1 - \mathbb{P}\{y = 0 \mid \boldsymbol{x}\}$ as

$$p(\boldsymbol{x}) = \sigma(\boldsymbol{\beta}^T \boldsymbol{x})$$

where $\boldsymbol{x} = (x_1, \ldots, x_p)^T$, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T$, and $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function, so that $p(\boldsymbol{x}) \in [0, 1]$. Alternatively, we model the log-odds as a linear combination of the inputs:

$$\log\left[\frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})}\right] = \boldsymbol{\beta}^T \boldsymbol{x}.$$

After we trained the model, we can use it to predict the class of a new data point by

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } p(\boldsymbol{x}) > \tau \\ 0 & \text{if } p(\boldsymbol{x}) \leq \tau \end{cases}$$

where $\tau \in [0, 1]$ is some threshold. It is clear that logistic regression draws a hyperplane in the input space: σ is monotonically increasing, so $\sigma(\boldsymbol{\beta}^T \boldsymbol{x}_1) = \sigma(\boldsymbol{\beta}^T \boldsymbol{x}_2)$ if and only if $\boldsymbol{\beta}^T \boldsymbol{x}_1 = \boldsymbol{\beta}^T \boldsymbol{x}_2$.

3.1.1 Estimation of Parameters

MLE We can estimate $\boldsymbol{\beta}$ using maximum likelihood. The log-likelihood is

$$\begin{aligned} \ell(\boldsymbol{\beta}) &= \log \prod_{i=1}^{n} p(\boldsymbol{x}_i)^{y_i} (1 - p(\boldsymbol{x}_i))^{1 - y_i} \\ &= \sum_{i=1}^{n} \{y_i \log p(\boldsymbol{x}_i) + (1 - y_i) \log[1 - p(\boldsymbol{x}_i)]\} \\ &= \sum_{i=1}^{n} \left\{y_i \boldsymbol{\beta}^T \boldsymbol{x}_i + \log[1 - p(\boldsymbol{x}_i)]\right\}. \end{aligned} \tag{6}$$

Recalling $\sigma' = \sigma(1 - \sigma)$, the gradient and Hessian matrix of $\ell(\boldsymbol{\beta})$ is easily computed as

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} \boldsymbol{x}_i [y_i - p(\boldsymbol{x}_i)] = \boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{p}), \quad \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = -\sum_{i=1}^{n} \boldsymbol{x}_i \boldsymbol{x}_i^T p(\boldsymbol{x}_i)[1 - p(\boldsymbol{x}_i)] = -\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X},$$

where

- $\boldsymbol{X}$ is the $n \times (p + 1)$ data matrix with rows $\boldsymbol{x}_i^T$, $i = 1, \ldots, n$;

- $\boldsymbol{y} = (y_1, \ldots, y_n)^T$ and $\boldsymbol{p} = [p(x_1), \ldots, p(x_n)]^T$;
- $\boldsymbol{W}$ is the $(n \times n)$ diagonal matrix with $w_{ii} = p(\boldsymbol{x}_i)[1 - p(\boldsymbol{x}_i)]$.

Since the gradient is not a linear function of $\boldsymbol{\beta}$, it is not easy to obtain a closed-form formula for $\boldsymbol{\beta}$ by setting the gradient to 0. It is informative to see how we maximize $\ell(\boldsymbol{\beta})$ iteratively via Newton-Raphson:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \left[\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T}\right]^{-1}_{\boldsymbol{\beta}^{(t)}} \left[\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}\right]_{\boldsymbol{\beta}^{(t)}}.$$

We find

$$\begin{aligned}
\boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} + (\boldsymbol{X}^T \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{p}^{(t)}) \\
&= \left[(\boldsymbol{X}^T \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} (\boldsymbol{X}^T \boldsymbol{W}^{(t)} \boldsymbol{X})\right] \boldsymbol{\beta}^{(t)} + (\boldsymbol{X}^T \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^T \left[\boldsymbol{W}^{(t)} \boldsymbol{W}^{(t)^{-1}}\right] (\boldsymbol{y} - \boldsymbol{p}^{(t)}) \\
&= (\boldsymbol{X}^T \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W}^{(t)} [\boldsymbol{X} \boldsymbol{\beta}^{(t)} + \boldsymbol{W}^{(t)^{-1}} (\boldsymbol{y} - \boldsymbol{p}^{(t)})] \\
&= (\boldsymbol{X}^T \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W}^{(t)} \boldsymbol{z}^{(t)}, \qquad \text{where } \boldsymbol{z}^{(t)} = [\boldsymbol{X} \boldsymbol{\beta}^{(t)} + \boldsymbol{W}^{(t)^{-1}} (\boldsymbol{y} - \boldsymbol{p}^{(t)})]
\end{aligned}$$

and $\boldsymbol{W}^{(t)}$ and $\boldsymbol{p}^{(t)}$ are $\boldsymbol{W}$ and $\boldsymbol{p}$ evaluated at $\boldsymbol{\beta}^{(t)}$. This is the *iteratively re-weighted least squares*: at each step we select $\boldsymbol{\beta}$ that minimizes the weighted least squares

$$\sum_{i=1}^{n} w_{ii}^{(t)} (z_i^{(t)} - \boldsymbol{x}_i^T \boldsymbol{\beta})^2 = (\boldsymbol{z}^{(t)} - \boldsymbol{X} \boldsymbol{\beta})^T \boldsymbol{W}^{(t)} (\boldsymbol{z}^{(t)} - \boldsymbol{X} \boldsymbol{\beta})$$

where $\boldsymbol{z}^{(t)}$ is the *adjusted response*.

So we can view the training process of logistic regression as repeatedly fitting least squares (drawing straight lines) until convergence, each time updating the response data and the weights.

Loss Function We can also define the $-\ell(\boldsymbol{\beta})$ as the loss function, and maximizing the log-likelihood is equivalent to minimizing the loss function. For each term in Eq. (6),

$$\begin{aligned}
L_i(\boldsymbol{\beta}) = -y_i \boldsymbol{\beta}^T \boldsymbol{x}_i - \log[1 - p(\boldsymbol{x}_i)] &= -y_i \boldsymbol{\beta}^T \boldsymbol{x}_i - \log(1 + e^{\boldsymbol{\beta}^T \boldsymbol{x}_i})^{-1} \\
&= -y_i \boldsymbol{\beta}^T \boldsymbol{x}_i + \log(1 + e^{\boldsymbol{\beta}^T \boldsymbol{x}_i}) \\
&= \log(e^{-y_i \boldsymbol{\beta}^T \boldsymbol{x}_i}) + \log(1 + e^{\boldsymbol{\beta}^T \boldsymbol{x}_i}) \\
&= \log\left(e^{-y_i \boldsymbol{\beta}^T \boldsymbol{x}_i} + e^{(1-y_i) \boldsymbol{\beta}^T \boldsymbol{x}_i}\right).
\end{aligned}$$

We can see that letting $y \in \{0, 1\}$ does not give us a very easy-to-remember formula. So at this point we try another representation of the response: we let $y \in \{-1, 1\}$, and let $p(\boldsymbol{x}) = \mathbb{P}\{y = 1 \mid \boldsymbol{x}\} = \sigma(\boldsymbol{\beta}^T \boldsymbol{x})$. Because of the property of the sigmoid function $\sigma(-x) = 1 - \sigma(x)$,

we have $\mathbb{P}\{y = -1 \mid \boldsymbol{x}\} = 1 - \mathbb{P}\{y = 1 \mid \boldsymbol{x}\} = p(-\boldsymbol{x})$. So using this representation the likelihood of each data point is $p(\boldsymbol{x}_i, y_i) = \sigma(y_i \boldsymbol{\beta}^T \boldsymbol{x}_i)$. The log-likelihood of the data is

$$\begin{aligned}\ell(\boldsymbol{\beta}) &= \log \prod_{i=1}^{n} p(\boldsymbol{x}_i, y_i) = \log \prod_{i=1}^{n} \sigma(y_i \boldsymbol{\beta}^T \boldsymbol{x}_i) = \sum_{i=1}^{n} \log \sigma(y_i \boldsymbol{\beta}^T \boldsymbol{x}_i) \\ &= \sum_{i=1}^{n} \log \left(1 + e^{-y_i \boldsymbol{\beta}^T \boldsymbol{x}_i}\right)^{-1} \\ &= -\sum_{i=1}^{n} \log \left(1 + e^{-y_i \boldsymbol{\beta}^T \boldsymbol{x}_i}\right).\end{aligned}$$

Now we can see that

$$L_i(\boldsymbol{\beta}) = \log \left(1 + e^{-y_i \boldsymbol{\beta}^T \boldsymbol{x}_i}\right)$$

is a natural loss function. Maximizing the likelihood is equivalent to minimizing the loss. The gradient of this new $\ell(\boldsymbol{\beta})$ is $\nabla \ell(\boldsymbol{\beta}) = \boldsymbol{X}^T \boldsymbol{Y}(\boldsymbol{1} - \boldsymbol{p})$ and the Hessian is $\nabla^2 \ell(\boldsymbol{\beta}) = -(\boldsymbol{X}^T \boldsymbol{Y}) \boldsymbol{W} (\boldsymbol{X}^T \boldsymbol{Y})$, where $\boldsymbol{Y}$ is the diagonal matrix with y_i, $i = 1, \ldots, n$ on its diagonal. So under this representation we can still interpret the training process as repeatedly doing least-squares.

3.1.2 More than two classes

Multinomial logistic regression is also called softmax regression. When we have to give probabilities on k different classes, we have to specify at least $k-1$ probabilities. The model is

$$p_j(\boldsymbol{x}) = \frac{\exp(\boldsymbol{\beta}_j^T \boldsymbol{x})}{\sum_{i=1}^{k} \exp(\boldsymbol{\beta}_i^T \boldsymbol{x})}, \qquad j = 1, \ldots, k.$$

To train the model we can use MLE, or equivalently use cross-entropy as the loss function.

3.2 Generative Learning

In generative learning, we take an indirect approach to do classification. Rather than learning a function $f : \mathbb{R}^p \to [0, 1]$ and use it to model the probability $\mathbb{P}\{y \mid x\}$, we model the distribution of the inputs $\mathbb{P}\{x \mid y\}$ under different classes y, as well as the distribution of the labels $\mathbb{P}\{y\}$, then we use Bayes's rule

$$\mathbb{P}\{y \mid x\} = \frac{\mathbb{P}\{x \mid y\}\mathbb{P}\{y\}}{\mathbb{P}\{x\}} = \frac{\mathbb{P}\{x \mid y\}\mathbb{P}\{y\}}{\sum\limits_{y} \mathbb{P}\{x \mid y\}\mathbb{P}\{y\}} \tag{7}$$

to model the probability of y given x. To make prediction, we select the largest $\mathbb{P}\{y \mid x\}$, i.e.

$$C(x) = \arg\max_{y} \mathbb{P}\{y \mid x\} = \arg\max_{y} \frac{\mathbb{P}\{x \mid y\}\mathbb{P}\{y\}}{\mathbb{P}\{x\}} = \arg\max_{y} \mathbb{P}\{x \mid y\}\mathbb{P}\{y\}.$$

Model training is simple: given our setup, what we need to learn is the parameters of the distributions of x and y. So we can use maximum likelihood estimation to get the parameters from the data, and then we can use the estimated parameters to make predictions.

3.2.1 Gaussian Discriminant Analysis

In Gaussian discriminant analysis (GDA), we assume the features in each class have multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y)$ and the labels have categorical distribution $(p_1, \ldots, p_k)$. In *linear discriminant analysis (LDA)* it is assumed that the normal distributions for each class has the same variance, but only differ in their means. In *quadratic discriminant analysis (QDA)* the normal distribution of each class can have different means and variances.

The (maximum likelihood) estimation of parameters is of no difficulty:

- $\hat{\boldsymbol{\mu}}_y$ is the mean of the features in each class;
- $\hat{p}_i = n_i/n$ where n_i is the count of class i in the training data;
- $\hat{\boldsymbol{\Sigma}} = (1/n)\sum_{i=1}^n (\boldsymbol{x}_i - \boldsymbol{\mu}_{y_i})(\boldsymbol{x}_i - \boldsymbol{\mu}_{y_i})^T$ for LDA;
- $\hat{\boldsymbol{\Sigma}}_y = (1/n)\sum_{i=1}^n \mathbb{I}(y_i = y)(\boldsymbol{x}_i - \boldsymbol{\mu}_{y_i})(\boldsymbol{x}_i - \boldsymbol{\mu}_{y_i})^T$ for QDA.

Let's first see the case for one predictor and two classes, and then for $p > 1$ and $y \in \{1, \ldots, k\}$.

LDA, $p = 1$ and $y \in \{1, 2\}$ The Gaussian density is

$$f_y(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu_y)^2}{2\sigma^2}\right\}, \quad y = 1, 2.$$

We predict $y = 1$ if and only if

$$\begin{gathered}
p_1 f_1(x) > p_2 f_2(x) \\
\Downarrow \\
\log p_1 + \log f_1(x) > \log p_2 + \log f_2(x) \\
\Downarrow \\
\log p_1 - \frac{(x-\mu_1)^2}{2\sigma^2} > \log p_2 - \frac{(x-\mu_2)^2}{2\sigma^2} \\
\Downarrow \\
\left(\log p_1 - \frac{1}{2}\frac{\mu_1^2}{\sigma^2}\right) + \frac{\mu_1}{\sigma^2}x > \left(\log p_2 - \frac{1}{2}\frac{\mu_2^2}{\sigma^2}\right) + \frac{\mu_2}{\sigma^2}x.
\end{gathered}$$

We see that the decision boundary is linear.

LDA, $p = 1$ and $y \in \{1, \dots, k\}$ The Gaussian density is

$$f_y(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(x-\mu_y)^2}{2\sigma^2}\right\}, \quad y = 1, \dots, k.$$

We predict label $j \in \{1, \dots, k\}$ if and only if

$$\begin{gathered} p_j f_j(x) > p_i f_i(x) \quad \forall i \neq j \\ \Downarrow \\ \left(\log p_j - \frac{1}{2}\frac{\mu_j^2}{\sigma^2}\right) + \frac{\mu_j}{\sigma^2} x > \left(\log p_i - \frac{1}{2}\frac{\mu_i^2}{\sigma^2}\right) + \frac{\mu_i}{\sigma^2} x \quad \forall i \neq j. \end{gathered}$$

LDA, $p > 1$ and $y \in \{1, \dots, k\}$ The multivariate Gaussian density is

$$f_y(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{(\boldsymbol{x}-\boldsymbol{\mu}_y)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_y)}{2}\right\}, y = 1, \dots, k.$$

Similar to the previous cases, we predict j if

$$\begin{gathered} p_j f_j(\boldsymbol{x}) > p_i f_i(\boldsymbol{x}) \quad \forall i \neq j \\ \Downarrow \\ \log p_j + \log f_j(\boldsymbol{x}) > \log p_i + \log f_i(\boldsymbol{x}) \\ \Downarrow \\ \log p_j - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j) > \log p_i - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) \\ \Downarrow \\ \left(\log p_j - \frac{1}{2}\boldsymbol{\mu}_j^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j\right) + \boldsymbol{x}^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j > \left(\log p_i - \frac{1}{2}\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i\right) + \boldsymbol{x}^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_i. \end{gathered}$$

The decision boundary is again linear. Let us call

$$\delta_y(\boldsymbol{x}) := \log(p_y f_y(\boldsymbol{x})) = \left(\log p_y - \frac{1}{2}\boldsymbol{\mu}_y^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_y\right) + \boldsymbol{x}^T \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_y$$

the *discriminant function*. It is a linear function of $\boldsymbol{x}$. The prediction rule is

$$C(\boldsymbol{x}) = \arg\max\{\delta_1(\boldsymbol{x}), \delta_2(\boldsymbol{x}), \dots, \delta_k(\boldsymbol{x})\}.$$

QDA, $p > 1$ and $y \in \{1, \dots, k\}$ If we allow the Gaussian distribution in each class to have a different variance, then the discriminant function becomes

$$\delta_y(\boldsymbol{x}) := \log(p_y f_y(\boldsymbol{x})) = \left(\log p_y - \frac{1}{2}\log|\boldsymbol{\Sigma}_y|\right) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_y)^T \boldsymbol{\Sigma}_y^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_y).$$

We see that the decision boundary is now quadratic.

Once we have the discriminant functions, we can get the probabilities of y given x (Eq. (7)) as

$$\mathbb{P}\{y \mid x\} = \frac{p_y f_y(\boldsymbol{x})}{\sum_{y=1}^{k} p_y f_y(\boldsymbol{x})} = \frac{e^{\delta_y(\boldsymbol{x})}}{\sum_{y=1}^{k} e^{\delta_y(\boldsymbol{x})}}.$$

We see that the model of GDA has the same form as logistic regression. The two model differ in their estimation of parameters. In logistic regression no assumption is made on distribution of the inputs, and we use maximum likelihood to estimate the parameters. Thus logistic regression is more robust, but it also requires more training data.

3.2.2 Naive Bayes

In naive Bayes model, we take the same strategy as explained at the beginning of Section 3.2, but the additional assumption is that in each class, the distributions of the features are independent, namely for $\boldsymbol{x} = (x_1, \ldots, x_p)$

$$\mathbb{P}\{\boldsymbol{x} \mid y\} = \mathbb{P}\{x_1 \mid y\}\mathbb{P}\{x_2 \mid y\} \cdots \mathbb{P}\{x_p \mid y\}.$$

Thus, we can specify a distribution $f_y(x_i)$ for each feature $x_i, i = 1, \ldots, p$. Then we can estimate its parameters using MLE, ignoring other variables. When we get the estimated distribution for each feature, we can multiply them up to get the distribution of the inputs. To make prediction, we calculate

$$\begin{aligned}\delta_y(\boldsymbol{x}) &= \log(p_y f_y(\boldsymbol{x})) = \log(p_y) + \log(f_y(\boldsymbol{x})) = \log(p_y) + \log \prod_{i=1}^{p} f_y(x_i) \\ &= \log(p_y) + \sum_{i=1}^{p} \log f_y(x_i)\end{aligned}$$

and we select the largest $\delta_y(\boldsymbol{x})$:

$$C(\boldsymbol{x}) = \arg\max\{\delta_1(\boldsymbol{x}), \delta_2(\boldsymbol{x}), \ldots, \delta_k(\boldsymbol{x})\}.$$

For example, if we assume each feature follow a Gaussian distribution with different mean and variance, then the discriminant function is

$$\delta_y(\boldsymbol{x}) = \log p_y - \sum_{i=1}^{p} \frac{(x_i - \mu_y)^2}{2\sigma_y^2}$$

so the decision boundary is quadratic.

We can see that naive Bayes model can conveniently handle data with both quantitative and qualitative variables.

4 Beyond Linearity

In this section, we assume the input x is one-dimensional.

The basis expansion model is

$$y = \beta_0 + \beta_1 b_1(x) + \cdots + \beta_p b_p(x) + \epsilon$$

where $b_1(\cdot), \ldots, b_p(\cdot)$ are fixed and known.

- Polynomial regression: $b_j(x) = x^j$.
- Step function regression: $b_j(x) = 1(c_j \leq x \leq c_{j+1})$.
- Piecewise polynomial regression: $b_j(x) = x^j 1(c_j \leq x \leq c_{j+1})$. The problem is that there are non-continuities at the knots. To address this problem we introduce regression and smoothing splines.

Splines use different polynomials to fit different regions in the input space, but at the same time it ensures that the boundaries between those different polynomials are continuous or smooth. Thus, it adds flexibility to the linear regression model, while maintaining the local constancy assumption.

Formally, letting $\{\xi_1, \ldots, \xi_K\}$ denote a set of knots, a spline $f(x)$ of order $d+1$ is a piecewise polynomial of degree d that has continuous derivatives up to order $d-1$. We can use the truncated power basis

$$b_j(x) = x^j,\ j = 1, \ldots, d, \qquad b_{d+k}(x) = (x - \xi_k)_+^d = \begin{cases} (x-\xi_k)^d, x > \xi_k \\ 0, x \leq \xi_k, \end{cases} \quad k = 1, \ldots, K.$$

so that the model is

$$y = \beta_0 + \beta_1 x + \cdots + \beta_d x_d + \beta_{d+1}(x - \xi_1)_+^d + \cdots + \beta_{d+K}(x - \xi_K)_+^d + \epsilon.$$

For example, a linear spline with one knot ξ_1 is $f(x) = \beta_0 + \beta_1 x + \beta_2 (x - \xi_1)_+$. When $x \leq \xi_1$ it is $f(x) = \beta_0 + \beta_1 x$; when $x \geq \xi_1$ it is $f(x) = \beta_0 + \beta_1 x + \beta_2(x - \xi_1) = (\beta_0 - \beta_2 \xi_1) + (\beta_1 + \beta_2)x$.

4.1 Cubic Splines

The *cubic spline* model with knots $\{\xi_1, \ldots, \xi_K\}$ is

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi_1)_+^3 + \cdots + \beta_{3+K}(x - \xi_K)_+^3 \tag{8}$$

From Eq. (8) one can check that

$$f(\xi_k^-) = \beta_0 + \sum_{i=1}^{3} \beta_i \xi_k^i + \sum_{i<k} \beta_{4+i}(\xi_k - \xi_i)_+^3 = f(\xi_k^+),$$
$$f'(\xi_k^-) = \beta_1 + \sum_{i=2}^{3} i\beta_i \xi_k^{i-1} + \sum_{i<k} 3\beta_{4+i}(\xi_k - \xi_i)_+^2 = f'(\xi_k^+),$$
$$f''(\xi_k^-) = 2\beta_2 + 6\beta_3 \xi_k + \sum_{i<k} 6\beta_{4+i}(\xi_k - \xi_i)_+ = f''(\xi_k^+).$$

for $k = 1, \ldots, K$.

4.2 Natural Cubic Splines

A problem with cubic spline is that the 3rd degree polynomial at the end of the data (i.e. when $x \to \pm\infty$) is very erratic. *Natural cubic spline* adds the constraint on top of cubic spline that the function should be *linear* outside the range $[\xi_1, \xi_K]$.

One can obtain the natural cubic spline model by using the following basis

$$N_1(x) = 1, N_2(x) = x, N_{2+k}(x) = \frac{(x-\xi_k)_+^3 - (x-\xi_K)_+^3}{\xi_K - \xi_k} - \frac{(x-\xi_{K-1})_+^3 - (x-\xi_K)_+^3}{\xi_K - \xi_{K-1}}$$

for $k = 1, \ldots, K-2$. The model is

$$y = \beta_0 N_1(x) + \beta_1 N_2(x) + \beta_2 N_3(x) + \cdots + \beta_{K-1} N_K(x) + \epsilon$$

and from this we can use least squares to estimate the coefficients.

The basis is obtained as follows. We start from a cubic spline

$$f(x) = \sum_{i=0}^{3} \beta_i x^i + \sum_{k=1}^{K} \theta_k (x - \xi_k)_+^3 \tag{9}$$

and add the two constraints

$$x \le \xi_1 \Rightarrow f''(x) = 2\beta_2 + 6\beta_3 x = 0 \tag{10}$$
$$x \ge \xi_K \Rightarrow f''(x) = 2\beta_2 + 6\beta_3 x + 6\sum_{k=1}^{K} \theta_k (x - \xi_k) = 0 \tag{11}$$

From Eq. (10) $\beta_2 = \beta_3 = 0$, and from Eq. (11) $x \sum_{k=1}^{K} \theta_k = \sum_{k=1}^{K} \theta_k \xi_k = 0$ we get $\sum_{k=1}^{K} \theta_k = 0$ and $\sum_{k=1}^{K} \theta_k \xi_k = 0$. We further have

$$\theta_K = -\sum_{k=1}^{K-1} \theta_k \tag{12}$$

and

$$0 = \sum_{k=1}^{K} \theta_k(\xi_K - \xi_k) = \sum_{k=1}^{K-1} \theta_k(\xi_K - \xi_k) \quad \Rightarrow \quad \theta_{K-1} = -\sum_{k=1}^{K-2} \theta_k \frac{\xi_K - \xi_k}{\xi_K - \xi_{K-1}}. \tag{13}$$

Take Eq. (12) and Eq. (13) into Eq. (9), we get

$$\begin{aligned}
f(x) &= \beta_0 + \beta_1 x + \sum_{k=1}^{K-1} \theta_k (x - \xi_k)_+^3 - (x - \xi_K)_+^3 \sum_{k=1}^{K-1} \theta_k \\
&= \beta_0 + \beta_1 x + \sum_{k=1}^{K-1} \theta_k [(x - \xi_k)_+^3 - (x - \xi_K)_+^3] \\
&= \beta_0 + \beta_1 x + \sum_{k=1}^{K-1} \theta_k (\xi_K - \xi_k) \cdot \frac{[(x - \xi_k)_+^3 - (x - \xi_K)_+^3]}{\xi_K - \xi_k} \\
&= \beta_0 + \beta_1 x + \sum_{k=1}^{K-2} \theta_k (\xi_K - \xi_k) \cdot \frac{[(x - \xi_k)_+^3 - (x - \xi_K)_+^3]}{\xi_K - \xi_k} \\
&\quad + \theta_{K-1}(\xi_K - \xi_{K-1}) \cdot \frac{[(x - \xi_{K-1})_+^3 - (x - \xi_K)_+^3]}{\xi_K - \xi_{K-1}} \\
&= \beta_0 + \beta_1 x + \sum_{k=1}^{K-2} \beta_{1+k} \left[\frac{(x - \xi_k)_+^3 - (x - \xi_K)_+^3}{\xi_K - \xi_k} - \frac{(x - \xi_{K-1})_+^3 - (x - \xi_K)_+^3}{\xi_K - \xi_{K-1}} \right] \\
&= \beta_0 N_1(x) + \beta_1 N_2(x) + \beta_2 N_3(x) + \cdots + \beta_{K-1} N_K(x).
\end{aligned}$$

How do we determine the knots? In practice it is more common to choose the *number* of knots via cross–validation and then place knots at *uniform quantiles* of the input.

4.3 Smoothing Splines

Smoothing splines are obtained by minimizing the penalized RSS function

$$\sum_{i=1}^{n} [y_i - f(x_i)]^2 + \lambda \int_{-\infty}^{\infty} f''(x)^2 dx, \quad \lambda \geq 0$$

within the space of C^2 functions. It can be shown that the solution is a natural cubic spline with knots at $x_1, \ldots, x_n$. Thus, knowing that the solution has the form $f(x) = \sum_{j=1}^{n} \beta_{j-1} N_j(x)$, we can re-formulate the problem as minimizing

$$(\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{N}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\Omega}_N \boldsymbol{\beta}$$

where $\boldsymbol{N}_{[ij]} = N_j(x_i)$ and $\boldsymbol{\Omega}_{N[jk]} = \int N_j''(x) N_k''(x) dx$. The solution can be easily obtained by noticing that the above function is the same as the one of the ridge regression, with the only difference that $\boldsymbol{I}_p$ is replaced by $\boldsymbol{\Omega}_N$. Hence

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{N}^T \boldsymbol{N} + \lambda \boldsymbol{\Omega}_N)^{-1} \boldsymbol{N}^T \boldsymbol{y}, \quad \hat{f}(x) = \sum_{j=1}^{n} \hat{\beta}_{j-1} N_j(x).$$

Note that this is not the same natural cubic spline that one would get if one applied the basis function approach with knots at $x_1, \ldots, x_n$. Rather, it is a shrunken version of such a natural cubic spline, where the value of the tuning parameter λ controls the level of shrinkage.

The predictions for the training set are

$$\hat{\boldsymbol{y}} = \boldsymbol{N}(\boldsymbol{N}^T\boldsymbol{N} + \lambda\boldsymbol{\Omega}_N)^{-1}\boldsymbol{N}^T\boldsymbol{y} = \boldsymbol{S}_\lambda\boldsymbol{y}.$$

Hence, smoothing splines are linear smoothers and $\boldsymbol{S}_\lambda$ plays the same role of the projection matrix $\boldsymbol{H}$ in linear regression. The effective degrees of freedom of smoothing splines is defined as

$$\text{trace}(\boldsymbol{S}_\lambda) = \sum_{i=1}^{n} \boldsymbol{S}_{\lambda[ii]}.$$

The effective degrees of freedom depends on λ. As $\lambda \to \infty$ the effective degrees of freedom go to 2, while when $\lambda = 0$ the effective degrees of freedom are n.

Similar to linear regression, there is also a formula for computing the leave-one-out cross-validation MSE for selecting λ:

$$MSE(\lambda) = \frac{1}{n}\sum_{i=1}^{n}[y_i - f_\lambda^{(-i)}(x_i)]^2 = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{y_i - \hat{y}_i}{1 - \boldsymbol{S}_{\lambda[ii]}}\right]^2.$$

4.4 Local Regressions

There are some non-parametric form of regressions, all making use of local information:

- K-nearest neighbors regression:

  $$\hat{y} = \hat{f}(x) = \text{Ave}[y_i \mid \boldsymbol{x}_i \in \mathcal{N}_K(\boldsymbol{x})]$$

  where $\mathcal{N}_K(\boldsymbol{x})$ is the neighborhood of $\boldsymbol{x}$ containing the K closest training data. K-nearest neighbors estimates change in a discrete way leading to a non-smooth $\hat{f}$.

- Nadaraya–Watson kernel–weighted average:

  $$\hat{y} = \hat{f}(\boldsymbol{x}) = \frac{\sum_{i=1}^{n} K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)y_i}{\sum_{i=1}^{n} K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)} \quad \text{with} \quad K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i) = D(\|\boldsymbol{x} - \boldsymbol{x}_i\|/\lambda).$$

  $D : \mathbb{R} \to [0, 1]$ is a kernel symmetric around the origin, whereas λ is a bandwidth or smoothing parameter which defines the width of the local neighborhood. Larger λ implies lower variance (smoothness) but higher bias.

  Unlike K–nearest neighbors, kernel methods rely on a weighted average to predict y at a given x. In this average, the closer $\boldsymbol{x}_i$ to $\boldsymbol{x}$, the higher y_i contribute to the estimate.

This gives us a smoother estimate. However, unlike K–nearest neighbors, the width of the neighborhood does not depend on the sparsity of the data. If we want to include this feature, we can allow the smoothing parameter to vary with x ("loess").

- Locally weighted linear regression: there is a problem with locally weighted averages: they can be badly biased on the boundaries of the domain, because of the asymmetry of the kernel in that region. *Locally weighted linear regression* solves a separate weighted least squares problem at each target point x:

$$\arg\min_{\boldsymbol{\beta}_{\boldsymbol{x}}} \sum_{i=1} K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)(y_i - \boldsymbol{\beta}_{\boldsymbol{x}}^T \boldsymbol{x}_i)^2 = \operatorname*{argmin}_{\boldsymbol{\beta}_{\boldsymbol{x}}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_{\boldsymbol{x}})^T \boldsymbol{W}_{\boldsymbol{x}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_{\boldsymbol{x}})$$

  where $\boldsymbol{W}_{\boldsymbol{x}}$ is a diagonal matrix with $\boldsymbol{W}_{\boldsymbol{x}[ii]} = K_\lambda(\boldsymbol{x}, \boldsymbol{x}_i)$ for $i = 1, \ldots, n$. The solution is

$$\hat{\boldsymbol{\beta}}_{\boldsymbol{x}} = (\boldsymbol{X}^T \boldsymbol{W}_{\boldsymbol{x}} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W}_{\boldsymbol{x}} \boldsymbol{y}.$$

5 Decision Tree Learning

5.1 Decision Trees

Regression and classification trees divide the input space into M (a hyperparameter) distinct non-overlapping regions $R_1, \ldots, R_M$. For every new input that falls into the region R_m, we make the same prediction c_m, which is the mean or mode for the training samples in R_m. Namely, we aim to learn a function of the form

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m 1(\boldsymbol{x} \in R_m).$$

For building a regression tree, we would like to divide the input space in so as to minimize the RSS

$$RSS = \sum_{m=1}^{M} RSS_m = \sum_{m=1}^{M} \sum_{i:\boldsymbol{x}_i \in R_m} (y_i - c_m)^2.$$

However, this is computationally infeasible (NP-complete), so we use a top-down greedy method, the *recursive binary splitting* method, to train our decision tree model. We first do a single split along a single variable so as to minimize the RSS resulted from the split. Namely, we find a variable $x_i \in \{x_1, \ldots, x_p\}$ and a split s (among all n splits) so as to minimize

$$\sum_{i:\boldsymbol{x}_i \in R_1(j,s)} (y_i - c_{R_1})^2 + \sum_{i:\boldsymbol{x}_i \in R_2(j,s)} (y_i - c_{R_2})^2$$

where $R_1(j,s) = \{\boldsymbol{x} : x_j < s\}$ and $R_2(j,s) = \{\boldsymbol{x} : x_j \geq s\}$. Thus, in the first step we have to do pn search (p is the number of variables, n is the number of training samples). We can then do the same thing recursively in each region generated by the first step. Note that sometimes we may prefer not to split a node: if all the split of a node offer no reduction in the loss, then we do not have reason to perform the split. Thus, the decision tree built in the end is not necessary a complete binary tree. It may be unbalanced.

For building a classification tree, we can replace RSS by other criteria like the miss-classification rate. We generally predict the label of a sample falling in region R_m as the mode of the labels in R_m, or equivalently the label with the largest proportion:

$$c_m = \arg\max_k \{p_{m,k}\} \quad \text{with } p_{m,k} = \frac{1}{|R_m|} \sum_{i:\boldsymbol{x}_i \in R_m} 1(y_i = k), k = 1, \ldots, K.$$

Hence, for the training samples falling in region R_m, we will classify correctly all samples with the most common label, and incorrectly for samples with all the other labels. The miss-classification rate in region R_m is thus one minus the proportion of the most common

label:

$$E_m = \frac{1}{|R_m|} \sum_{i:\boldsymbol{x}_i \in R_m} \mathbb{I}(y_i \neq \arg\max_k \{p_{m,k}\}) = 1 - \max_k \{p_{m,k}\}.$$

We could use the miss-classification rate in place of the RSS to fit the classification tree. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

- The *Gini index* is

$$G_m = \sum_{k=1}^{M} p_{m,k}(1 - p_{m,k}).$$

  The Gini index is a measure of total variance across the K classes. It is not hard to see that the Gini index takes on a small value if all of the $p_{m,k}$'s are close to zero or one. For this reason the Gini index is referred to as a measure of node *purity* – a small value indicates that a node contains predominantly observations from a single class.

- The *cross-entropy* is

$$D_m = -\sum_{k=1}^{K} p_{m,k} \log(p_{m,k}).$$

  One can show that the entropy will take on a value near zero if the $p_{m,k}$'s are all near zero or near one. Therefore, like the Gini index, the entropy will take on a small value if the node is pure. In fact, it turns out that the Gini index and the entropy are quite similar numerically.

To see the advantages and disadvantages of trees, we quote the following from https://scikit-learn.org/stable/modules/tree.html:

> Some advantages of decision trees are:
>
> - Simple to understand and to interpret. Trees can be visualized.
> - Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
> - The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
> - Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable.
> - Able to handle multi-output problems.
> - Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By

contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.

- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

5.2 Bagging

Bagging (bootstrap aggregating) can be used to reduce variance. In bagging , B bootstrap sets of samples from the training data are generated. A tree $f_b(\boldsymbol{x})$ is then fit to each of them.

For regression, the final prediction is the average of the predictions made by the B trees:

$$f(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} f_b(\boldsymbol{x}).$$

For classification, the final prediction is the mode of the B predictions

$$f(\boldsymbol{x}) = \text{mode}\{f_1(\boldsymbol{x}), \ldots, f_b(\boldsymbol{x})\}.$$

On average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out-of-bag (OOB)* observations. We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the ith observation. In order to obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the ith observation. An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for a regression problem) or classification error (for a classification problem) can be computed. The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation.

5.3 Random Forests

If the trees constructed in bagging are highly correlated, then averaging over them does not yield a great reduction in variance. Random forest is an improvement over bagging that attempts to de-correlate the trees. In random forest method, we bootstrap B data sets from the training data set, and grow a tree on each data set. When building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$ – that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors.

The main difference between bagging and random forests is the choice of predictor subset size m. For instance, if a random forest is built using $m = p$, then this amounts simply to bagging.

6 Gradient Boosting

6.1 Motivations

We follow intuitions from Friedman 2001. Recall that in gradient descent, when we want to minimize some function $\Phi(\boldsymbol{P})$, we solve the problem as

$$\boldsymbol{P}^* = \sum_{m=0}^{M} \boldsymbol{p}_m$$

where $\boldsymbol{p}_0$ is an initial guess and

$$\boldsymbol{p}_m = -\rho_m \nabla\Phi(\boldsymbol{P}_{m-1}) \quad \text{with} \quad \boldsymbol{P}_{m-1} = \sum_{i=0}^{m-1} \boldsymbol{p}_i$$

and $\rho_m = \arg\min_\rho \Phi(\boldsymbol{P}_{m-1} - \rho\nabla\Phi(\boldsymbol{P}_{m-1}))$. We can take an analogous approach when we want to find a function $F : \mathbb{R}^p \to \mathbb{R}$ that minimizes the (expected) loss $\Phi(F) = \mathbb{E}_{(\boldsymbol{x},y)} L(y, F)$. We can let

$$F^* = \sum_{m=0}^{M} f_m$$

where f_0 is an initial guess and

$$f_m = -\rho_m \nabla\Phi(F_{m-1}) \quad \text{with} \quad F_{m-1} = \sum_{i=0}^{m-1} f_i$$

and $\rho_m = \arg\min_\rho \Phi(F_{m-1} - \rho\nabla\Phi(F_{m-1}))$.

In reality, where we only have finite amount of data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, we replace the (unknown) objective function by the empirical loss $\Phi(F) = \sum_{i=1}^n L(y_i, F(\boldsymbol{x}_i))$. However, now we are only able to calculate the first derivative at finite set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$:

$$f_m(\boldsymbol{x}_i) = -\rho_m \cdot \left.\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right|_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})} \qquad i = 1, \ldots, n$$

so we do not know the value of f_m at other points $x \in \mathbb{R}^p$. So at this point we have to use some parametric function $h(\boldsymbol{x}; \boldsymbol{a}_m)$ to approximate $f_m(\boldsymbol{x})$. We choose $\boldsymbol{a}_m$ so that $\{h(\boldsymbol{x}_i; \boldsymbol{a}_m)\}_{i=1}^n$ is as parallel to the negative gradient $\{-\partial L(y_i, F(\boldsymbol{x}_i))/\partial F(\boldsymbol{x}_i)\}_{i=1}^n$ as possible. This we can do using least squares. This gives rise to the gradient boosting algorithm Algorithm 6.1.1.

6.2 Boosting Schemes

Let's now see this algorithm with different loss functions and base learners $h(\boldsymbol{x}; \boldsymbol{a})$.

Algorithm 6.1.1 Gradient Boosting

1: $F_0(\boldsymbol{x}) = \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, \rho)$
2: **for** $m = 1$ to M **do**
3: $\quad \tilde{y}_i = -\left.\dfrac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right|_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}, \quad i = 1, \ldots, n$
4: $\quad \boldsymbol{a}_m = \arg\min_{\boldsymbol{a},\beta} \sum_{i=1}^{n} [\tilde{y}_i - \beta h(\boldsymbol{x}_i; \boldsymbol{a})]^2$
5: $\quad \rho_m = \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m))$
6: $\quad F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$
7: **return** $F_M(\boldsymbol{x}) = F_0(\boldsymbol{x}) + \sum_{m=1}^{M} \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$.

6.2.1 LS-Boost

If our loss function is $L(y, F) = (y - F)^2/2$, then the negative gradient is $\tilde{y}_i = y_i - F_{m-1}(\boldsymbol{x}_i)$, i.e. the residuals. So with this loss function the algorithm performs iterative fitting of current residuals.

1: LS-Boost
2: $\quad F_0 = \bar{y}$
3: $\quad$ **for** $m = 1$ to M **do**
4: $\quad\quad \tilde{y}_i = y_i - F_{m-1}(\boldsymbol{x}_i), \quad i = 1, \ldots, n$
5: $\quad\quad (\rho_m, \boldsymbol{a}_m) = \arg\min_{\boldsymbol{a},\rho} \sum_{i=1}^{n} [\tilde{y}_i - \rho h(\boldsymbol{x}_i; \boldsymbol{a})]^2$
6: $\quad\quad F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$
7: $\quad$ **return** $F_M(\boldsymbol{x})$

6.2.2 LAD-Boost

For the least absolute deviation $L(y, F) = |y - F|$, one has

$$\tilde{y}_i = -\left.\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right|_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})} = \operatorname{sign}(y_i - F_{m-1}(\boldsymbol{x}_i)).$$

This implies that $h(\boldsymbol{x};\boldsymbol{a})$ is fit (by least-squares) to the *sign* of the current residuals. Also the line search is

$$\begin{aligned}\rho_m &= \arg\min_{\rho}\sum_{i=1}^{n}|y_i - F_{m-1}(\boldsymbol{x}_i) - \rho h(\boldsymbol{x}_i;\boldsymbol{a}_m)| \\ &= \arg\min_{\rho}\sum_{i=1}^{n}|h(\boldsymbol{x}_i;\boldsymbol{a}_m)|\cdot\left|\frac{y_i - F_{m-1}(\boldsymbol{x}_i)}{h(\boldsymbol{x}_i;\boldsymbol{a}_m)} - \rho\right| \\ &= \text{median}_W\left\{\frac{y_i - F_{m-1}(\boldsymbol{x}_i)}{h(\boldsymbol{x}_i;\boldsymbol{a}_m)}\right\}_{i=1}^{n}, \quad w_i = |h(\boldsymbol{x}_i;\boldsymbol{a}_m)|.\end{aligned}$$

Inserting these results into Algorithm 6.1.1 yields an algorithm for least absolute deviation boosting, using any base learner $h(\boldsymbol{x};\boldsymbol{a})$.

1: LAD-Boost
2: $F_0 = \text{median}\,\{y_i\}_1^n$
3: **for** $m = 1$ to M **do**
4: $\tilde{y}_i = \text{sign}(y_i - F_{m-1}(\boldsymbol{x}_i)), \quad i = 1,\ldots,n$
5: $\boldsymbol{a}_m = \arg\min_{\boldsymbol{a},\rho}\sum_{i=1}^{n}[\tilde{y}_i - \rho h(\boldsymbol{x}_i;\boldsymbol{a})]^2$
6: $\rho_m = \text{median}_W\,\{(y_i - F_{m-1}(\boldsymbol{x}_i))/h(\boldsymbol{x}_i;\boldsymbol{a}_m)\}_{i=1}^{n}, \quad w_i = |h(\boldsymbol{x}_i;\boldsymbol{a}_m)|.$
7: $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m h(\boldsymbol{x};\boldsymbol{a}_m)$
8: **return** $F_M(\boldsymbol{x})$

6.2.3 AdaBoost

What about the exponential loss function $L(y,F) = e^{-yF}$ when $y \in \{-1,1\}$? When y and F have the same sign and F has large absolute value, then $yF >> 0$ so the loss function is close to zero, while if y and F have different signs, $-yF > 0$ so the loss can be very large. The negative gradient of L is

$$\tilde{y}_i = -\left.\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right|_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})} = y_i e^{-y_i F_{m-1}(\boldsymbol{x}_i)}.$$

Let $\omega_i^{(m)} = e^{-y_i F_{m-1}(\boldsymbol{x}_i)} = L(y_i, F_{m-1})$ for $i = 1,\ldots,n$. We see that $\tilde{y}_i = y_i\omega_i^{(m)}$ for $i = 1,\ldots,n$, namely the negative gradient at each step is the (positive or negative) loss at the previous step. This makes sense: for example, if $y_i = 1$ is positive and the loss $\omega_i^{(m)}$ is large, this then implies $F_{m-1}(\boldsymbol{x}_i) < 0$ has a large absolute value, so on the next iteration we should increase $F(\boldsymbol{x}_i)$ by a large amount, to steer it at the positive direction.

We next fit $\left\{\tilde{y}_i = y_i\omega_i^{(m)}\right\}_{i=1}^{n}$ as in line 4 of Algorithm 6.1.1

$$\boldsymbol{a}_m = \arg\min_{\boldsymbol{a},\beta}\sum_{i=1}^{n}[\tilde{y}_i - \beta h(\boldsymbol{x}_i;\boldsymbol{a})]^2 = \arg\min_{\boldsymbol{a},\beta}\sum_{i=1}^{n}\left[y_i\omega_i^{(m)} - \beta h(\boldsymbol{x}_i;\boldsymbol{a})\right]^2. \tag{14}$$

We can show that for any $\beta > 0$ the solution to Eq. (14) is the parameter $\boldsymbol{a}_m$ such that

$$\boldsymbol{a}_m = \arg\min_{\boldsymbol{a}} \sum_{i=1}^{n} \omega_i^{(m)} \mathbb{I}(y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a})), \tag{15}$$

i.e. we should select $\boldsymbol{a}$ so as to minimize the weighted number of misclassifications. Indeed, writing out the objective function it is (we omit the superscript in $\omega_i^{(m)}$)

$$\begin{aligned}
\sum_{i=1}^{n} [y_i \omega_i - \beta h(\boldsymbol{x}_i; \boldsymbol{a})]^2 &= \sum_{y_i = h(\boldsymbol{x}_i; \boldsymbol{a})} (\omega_i - \beta)^2 + \sum_{y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a})} (\omega_i + \beta)^2 \\
&= \sum_{y_i = h(\boldsymbol{x}_i; \boldsymbol{a})} \left(\omega_i^2 - 2\omega_i + \beta^2\right) + \sum_{y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a})} \left(\omega_i^2 + 2\omega_i + \beta^2\right) \\
&= 2 \left(\sum_{y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a})} \omega_i - \sum_{y_i = h(\boldsymbol{x}_i; \boldsymbol{a})} \omega_i \right) + \sum_{i=1}^{n} \omega_i^2 + n\beta^2.
\end{aligned}$$

The term in the parenthesis is

$$\begin{aligned}
\sum_{y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a})} \omega_i - \sum_{y_i = h(\boldsymbol{x}_i; \boldsymbol{a})} \omega_i &= \sum_{i=1}^{n} \omega_i \mathbb{I}(y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a})) - \sum_{i=1}^{n} \omega_i \left(1 - \mathbb{I}(y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a}))\right) \\
&= 2 \sum_{i=1}^{n} \omega_i \mathbb{I}(y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a})) - \sum_{i=1}^{n} \omega_i
\end{aligned}$$

so we see that minimizing the objective function in Eq. (14) is equivalent to minimizing the objective function in Eq. (15). After we found such a parameter $\boldsymbol{a}_m^*$, we can then choose the best ρ_m in line 5 of Algorithm 6.1.1:

$$\begin{aligned}
\rho_m &= \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m^*)) \\
&= \arg\min_{\rho} \sum_{i=1}^{n} e^{-y_i (F_{m-1}(\boldsymbol{x}_i) + \rho h(\boldsymbol{x}_i; \boldsymbol{a}_m^*))} \\
&= \arg\min_{\rho} \sum_{i=1}^{n} \omega_i^{(m)} e^{[-y_i h(\boldsymbol{x}_i; \boldsymbol{a}_m^*)] \cdot \rho}.
\end{aligned}$$

Let $f(\rho) = \sum_{i=1}^{n} \omega_i^{(m)} e^{[-y_i h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)]\cdot\rho}$. We set $f'(\rho) = 0$:

$$f'(\rho) = \sum_{i=1}^{n} [-y_i h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)]\, \omega_i^{(m)} e^{[-y_i h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)]\cdot\rho} = -\sum_{y_i = h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)} e^{-\rho} + \sum_{y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)} e^{\rho} = 0$$

$$\Downarrow$$

$$e^{\rho} \left(\sum_{y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)} \right) = e^{-\rho} \left(\sum_{y_i = h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)} \right)$$

$$\Downarrow$$

$$e^{2\rho} = \left(\sum_{y_i = h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)} \right) \Big/ \left(\sum_{y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)} \right)$$

$$\Downarrow$$

$$\rho_m^* = \frac{1}{2} \log \frac{\sum_{y_i = h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)}}{\sum_{y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)} \omega_i^{(m)}} = \frac{1}{2} \log \frac{\sum_{i=1}^{n} \omega_i^{(m)} \left[1 - \mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*))\right]}{\sum_{i=1}^{n} \omega_i^{(m)} \mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*))} = \frac{1}{2} \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m}$$

where $\mathrm{err}_m = \left(\sum_{i=1}^{n} \omega_i^{(m)} \mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)) \right) / \left(\sum_{i=1}^{n} \omega_i^{(m)} \right)$ is the weighted error rate.

After we obtained $\boldsymbol{a}_m^*$ and ρ_m^*, we then update the approximation as $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m^* h(\boldsymbol{x}, \boldsymbol{a}_m^*)$. Let's see what is the next gradient $\left\{ y_i \omega_i^{(m+1)} \right\}_{i=1}^{n}$. Recall we have let $\omega_i^{(m)} = e^{-y_i F_{m-1}(\boldsymbol{x}_i)}$, so

$$\begin{aligned}
\omega_i^{(m+1)} &= e^{-y_i F_m(\boldsymbol{x}_i)} = e^{-y_i [F_{m-1}(\boldsymbol{x}) + \rho_m^* h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)]} \\
&= \omega_i^{(m)} \cdot e^{\rho_m^* \cdot [-y_i h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)]} \\
&= \omega_i^{(m)} \cdot e^{\rho_m^* \cdot [2\mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)) - 1]} \\
&= \omega_i^{(m)} \cdot e^{2\rho_m^* \mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*))} e^{-\rho_m^*}.
\end{aligned}$$

Removing the constant factor, this is equivalent to updating $\omega_i^{(m)}$ as

$$\omega_i^{(m+1)} = \omega_i^{(m)} \cdot \exp\left\{ \log \frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \cdot \mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)) \right\}. \tag{16}$$

For all data points $\{(\boldsymbol{x}_i, y_i)\}$ that are correctly classified ($\mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)) = 0$), the weight stays the same: $\omega_i^{(m+1)} = \omega_i^{(m)}$, while for all data points that are wrongly classified ($\mathbb{I}(y_i \neq h(\boldsymbol{x}_i;\boldsymbol{a}_m^*)) = 1$), their weights are all multiplied by $\exp\left[\log\left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right)\right] = \frac{1-\mathrm{err}_m}{\mathrm{err}_m}$. If the error rate err_m is low at the current stage, then $\frac{1-\mathrm{err}_m}{\mathrm{err}_m}$ is large, so we magnify those wrongly classified samples. We require that the base learner is at least slightly better than random guessing, so err_m should be lower than $1/2$, and consequently $\frac{1-\mathrm{err}_m}{\mathrm{err}_m}$ should always be larger than 1.

In summary, when we use the exponential loss to classify $y_i \in \{-1, 1\}$, the gradient is the labels times the losses, so when we fit the gradient we are fitting the labels magnified by the loss. To calculate the loss of the current step we just need to multiply the previous loss by a factor as in Eq. (16).

AdaBoost
  Initialize the weights $\omega_i = 1/n$, $i = 1, \ldots, n$
  for $m = 1$ to M **do**
    fit $h(\boldsymbol{x}, \boldsymbol{a}_m)$ to $\left\{y_i \omega_i^{(m)}\right\}_{i=1}^n$ that minimizes err_m;
    let $\rho_m = \log \frac{1-\text{err}_m}{\text{err}_m}$;
    update the weight as $\omega_i^{(m+1)} = \omega_i^{(m)} \cdot \exp\left\{\rho_m \cdot \mathbb{I}(y_i \neq h(\boldsymbol{x}_i; \boldsymbol{a}_m))\right\}$ for $i = 1, \ldots, n$.
  return $F_M(\boldsymbol{x}) = \sum_{m=1}^M \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m)$.

6.3 Gradient Tree Boosting

Gradient tree boosting is also called *gradient boosted regression trees* (GBRT). When the base learner is tree $h(\boldsymbol{x}; \boldsymbol{a}) = h(\boldsymbol{x}; \{b_j, R_j\}_{j=1}^J) = \sum_{j=1}^J b_j 1(\boldsymbol{x} \in R_j)$, we can make a slight improvement of the gradient boosting procedure. Recall we update our approximation as

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \rho_m \sum_{j=1}^J b_{jm} 1(\boldsymbol{x} \in R_{jm}),$$

but we can write it as

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \sum_{j=1}^J \gamma_{jm} 1(\boldsymbol{x} \in R_{jm})$$

with $\gamma_{jm} = \rho_m b_{jm}$. We can first fit the tree to the gradient, obtaining $\{R_{jm}\}_{j=1}^J$ and $\{b_{jm}\}_{j=1}^J$, and then adjust the coefficient b_{jm} of each region R_{jm} to γ_{jm}, so that

$$\{\gamma_{jm}\} = \arg\min_{\{\gamma_j\}_{j=1}^J} \sum_{i=1}^n L\left(y_i, F_{m-1}(\boldsymbol{x}_i) + \sum_{j=1}^J \gamma_j 1(\boldsymbol{x} \in R_{jm})\right).$$

Since regions are disjoint, we further have

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{jm}} L(y_i, F_{m-1}(\boldsymbol{x}_i) + \gamma).$$

To summarize, when we do gradient boosting with trees as base learners, we can first fit the tree to the negative gradient using least-squares as usual, but we can then *adjust the prediction in each region* so as to further decrease the loss, instead of doing a line search and multiply the predictions in all regions by the same constant. For example, if we use trees

in LAD-Boost (Section 6.2.2), at each step we can fit a tree with J terminal nodes to the current *sign* of the residuals $\{\tilde{y}_i = \text{sign}(y_i - F_{m-1}(\boldsymbol{x}_i))\}_{i=1,\ldots,n}$ using least-squares criterion, and then we change the prediction in each region from the *mean* of the predicted signs to

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{jm}} |y_i - F_{m-1}(\boldsymbol{x}_i) - \gamma| = \text{median}_{\boldsymbol{x} \in R_{jm}} \{y_i - F_{m-1}(\boldsymbol{x}_i)\},$$

the *median* of the current residuals in the region.

LAD-TreeBoost

$F_0 = \text{median}\,\{y_i\}_1^n$

for $m = 1$ to M **do**

$\tilde{y}_i = \text{sign}(y_i - F_{m-1}(\boldsymbol{x}_i)), \quad i = 1, \ldots, n$

$\{R_{jm}\}_1^J = J\text{-terminal node } tree\,(\{\boldsymbol{x}_i, \tilde{y}_i\}_1^n)$

$\gamma_{jm} = \text{median}_{\boldsymbol{x} \in R_{jm}} \{y_i - F_{m-1}(\boldsymbol{x}_i)\}$

$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \sum_{j=1}^J \gamma_{jm} 1(\boldsymbol{x} \in R_{jm})$

return $F_M(\boldsymbol{x})$

6.4 Comments

Other Loss Functions One can also do gradient boosting or gradient tree boosting with other loss functions, for example the Huber loss

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \le \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta, \end{cases}$$

for which the negative gradient is

$$\begin{aligned} \tilde{y}_i &= -\left.\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right|_{F(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x})} \\ &= \begin{cases} y_i - F_{m-1}(\boldsymbol{x}_i), & |y_i - F_{m-1}(\boldsymbol{x}_i)| \le \delta, \\ \delta \cdot \text{sign}(y_i - F_{m-1}(\boldsymbol{x}_i)), & |y_i - F_{m-1}(\boldsymbol{x}_i)| > \delta, \end{cases} \end{aligned}$$

or the negative binomial log-likelihood

$$L(y, F) = \log(1 + \exp(-2yF)), \qquad y \in \{-1, 1\},$$

or the multi-class cross-entropy

$$L(\{y_k, F_k(\boldsymbol{x})\}_1^K) = -\sum_{k=1}^K y_k \log p_k(\boldsymbol{x})$$

where $y_k = 1(\text{class} = k) \in \{0, 1\}$ and $p_k(\boldsymbol{x}) = \mathbb{P}(y_k = 1 \mid \boldsymbol{x}) = \exp(F_k(\boldsymbol{x}))/\sum_{j=1}^{K} \exp(F_j(\boldsymbol{x}))$. But the specific updates are somewhat complicated, so we refer to the original paper Friedman 2001 for details.

Regularizations To control for overfitting, we can use shrinkage:

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \nu \cdot \rho_m h(\boldsymbol{x}; \boldsymbol{a}_m), \qquad 0 < \nu \leq 1.$$

The two hyper-parameters ν and M are related: for smaller ν we need larger M.

7 Support Vector Machines

Support vector machine (SVM) is a classification method that aims to find a hyperplane to separate the data as best as possible. When the data is not linearly separable, soft margins can be used. Also, we can use kernel tricks to map the input data to high dimensional space, in which the data may become easier to separate. In this section, we assume our data is $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ where $y_i \in \{-1, 1\}$.

A hyperplane in p-dimensional space is defined via

$$f(\boldsymbol{x}) = w_1 x_1 + \cdots + w_p x_p + b = \boldsymbol{w}^T \boldsymbol{x} + b = 0.$$

In SVM, after we found an appropriate hyperplane, the decision rule is

$$C(\boldsymbol{x}) = \text{sign}(f(\boldsymbol{x})).$$

We have $y_i f(\boldsymbol{x}_i) > 0$ for correctly classified examples, so we want $y_i f(\boldsymbol{x}_i)$ to be as large as possible. Our optimization problem is

$$\begin{aligned} &\max_{\boldsymbol{w}, b} M \\ s.t. \quad &\|\boldsymbol{w}\| = 1, \\ &y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq M, \quad i = 1, \ldots, n. \end{aligned}$$

We can get rid of the constraint $\|\boldsymbol{w}\| = 1$ by requiring

$$\frac{y_i(\boldsymbol{w}^T \boldsymbol{x} + b)}{\|\boldsymbol{w}\|} = y_i \left[\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \right)^T \boldsymbol{x} + \frac{b}{\|\boldsymbol{w}\|} \right] \geq M, \quad i = 1, \ldots, n$$

or

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq M\|\boldsymbol{w}\|, \quad i = 1, \ldots, n$$

If we replace $\boldsymbol{w}$ by $k\boldsymbol{w}$ for any $k > 0$, then the inequality does not change:

$$\begin{gathered} y_i((k\boldsymbol{w})^T \boldsymbol{x} + b) \geq M\|k\boldsymbol{w}\|, \quad i = 1, \ldots, n \\ \Downarrow \\ \cancel{k} \cdot y_i(\boldsymbol{w}^T \boldsymbol{x} + b) \geq \cancel{k} \cdot M\|\boldsymbol{w}\|, \quad i = 1, \ldots, n \quad \checkmark \end{gathered}$$

So we can set $\|\boldsymbol{w}\| = 1/M$ (i.e. let $k = 1/(M\|\boldsymbol{w}\|)$). Maximizing M is equivalent to minimizing $\|\boldsymbol{w}\|$ or $\|\boldsymbol{w}\|^2$. The optimization problem can now be transformed as

$$\begin{aligned} &\min_{\boldsymbol{w}, b} \frac{1}{2}\|w\|^2 \\ s.t. \quad &y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1, \quad i = 1, \ldots, n. \end{aligned}$$

This is a quadratic programming problem. The Lagrangian is

$$\mathcal{L}(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1). \tag{17}$$

We set the gradient of the Lagrangian to zero:

$$\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0 \tag{18}$$

$$\frac{\partial}{\partial b} \mathcal{L}(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{19}$$

Take Eq. (18) and plug it into Eq. (17), we have

$$\begin{aligned}
\mathcal{L}(\boldsymbol{w}, b; \boldsymbol{\alpha}) = \frac{1}{2} &\left[(\alpha_1 y_1 \boldsymbol{x}_1 + \cdots + \alpha_n y_n \boldsymbol{x}_n)^T (\alpha_1 y_1 \boldsymbol{x}_1 + \cdots + \alpha_n y_n \boldsymbol{x}_n)\right] \\
&- \sum_{i=1}^{n} \alpha_i \left\{ y_i \left[\left(\sum_{j=1}^{n} \alpha_j y_j \boldsymbol{x}_j \right)^T \boldsymbol{x}_i + b \right] - 1 \right\} \\
= \frac{1}{2} &\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - b \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i \\
= &\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j.
\end{aligned}$$

We thus arrived at the *dual problem*

$$\begin{aligned}
\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\
s.t. \quad \alpha_i \geq 0, \quad i = 1, \ldots, n \\
\sum_{i=1}^{n} \alpha_i y_i = 0.
\end{aligned}$$

The algorithm for solving the dual problem is presented in Section 7.2.

7.1 Soft Margins

To fit a hyperplane when the data is not linearly-separable, and to be less sensitive to outliers, we reformulate the problem as

$$\begin{aligned}
\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i \\
s.t. \quad y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, n. \\
\xi_i \geq 0, \quad i = 1, \ldots, n.
\end{aligned}$$

The Lagrangian of the problem is

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\xi}, b; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(\boldsymbol{x}_i^T\boldsymbol{w} + b) - 1 + \xi) - \sum_{i=1}^{n}\beta_i\xi_i \tag{20}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$ are Lagrangian multipliers with $\alpha_i \geq 0$ and $\beta_i \geq 0$ for $i = 1, \ldots, n$. As before we set the gradient of the Lagrangian to zero, to get

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}, \boldsymbol{\xi}, b; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{w} - \sum_{i=1}^{n}\alpha_i y_i \boldsymbol{x}_i = 0 \tag{21}$$

$$\nabla_{\boldsymbol{\xi}}\mathcal{L}(\boldsymbol{w}, \boldsymbol{\xi}, b; \boldsymbol{\alpha}, \boldsymbol{\beta}) = C - \boldsymbol{\alpha} - \boldsymbol{\beta} = 0 \tag{22}$$

$$\frac{\partial}{\partial b}\mathcal{L}(\boldsymbol{w}, \boldsymbol{\xi}, b; \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i=1}^{n}\alpha_i y_i = 0. \tag{23}$$

We plug these into Eq. (20), to obtain the dual problem

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \tag{24}$$

$$s.t. \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, n \tag{25}$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0. \tag{26}$$

After adding regularization, the only change to the dual problem is that the constraint $\alpha_i \geq 0$ becomes $0 \leq \alpha_i \leq C$.

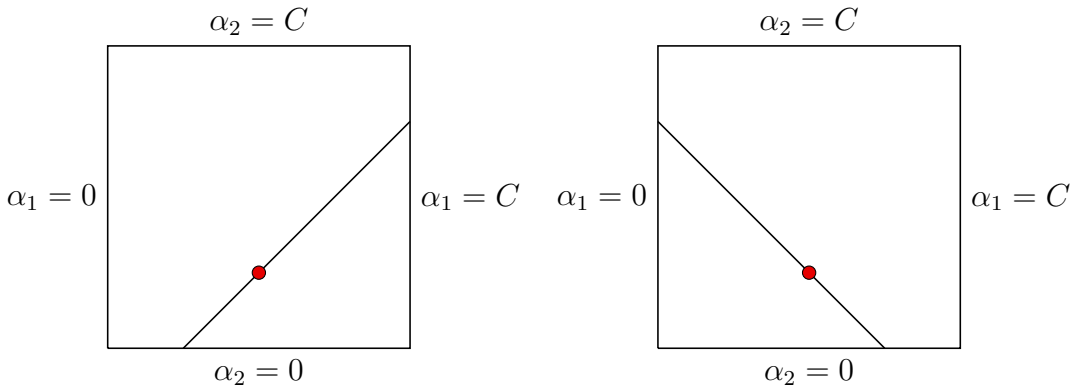
7.2 The SMO Algorithm

We motivate the sequential minimal optimization algorithm (Platt 1998) by coordinate descent. In coordinate descent, to maximize some function $W(\alpha_1, \alpha_2, \ldots, \alpha_n)$, we can first optimize W with respect to α_1, holding all other variables fixed. Then we optimize with respect to α_2 while holding $\alpha_1, \alpha_3, \ldots$ fixed. We always go parallel to the axes.

Return to our optimization problem Eqs. (24) to (26). Can we optimize Eq. (24) while with respect to α_1 while holding all other variables fixed? The answer is no because we have the constraint Eq. (26), so that

$$\alpha_1 y_1 = -\sum_{i=2}^{n}\alpha_i y_i \quad \Rightarrow \quad \alpha_1 = -y_1\sum_{i=2}^{n}\alpha_i y_i$$

by the fact that $y_1 \in \{-1, 1\}$ and hence $y_1^2 = 1$. Hence α_1 is determined by all other α_i's, so if we were to hold $\alpha_2, \ldots, \alpha_n$ fixed then we can't move α_1 as well.



(a) $y_1 \neq y_2 \Rightarrow \alpha_2 = \alpha_1 \pm k$ (b) $y_1 = y_2 \Rightarrow \alpha_2 = -\alpha_1 \pm k$

Figure 2: Illustration of the relationship between α_1 and α_2. The optimizer (α_1^*, α_2^*) lies on the line segment inside the box $[0, C] \times [0, C]$.

So we must update *at least* two of them simultaneously. The idea of the SMO algorithm is to exactly update two of the parameters at each iteration, and the key is that in this situation we have an analytic solution to the optimization problem. Let's say at some point we want to optimize $W(\alpha_1, \alpha_2, \ldots, \alpha_n)$ with respect to α_1 and α_2, holding $\alpha_3, \ldots, \alpha_n$ fixed. From Eq. (26) we first need to ensure

$$\alpha_1 y_1 + \alpha_2 y_2 = -\sum_{i=3}^{n} \alpha_i y_i = k \tag{27}$$

for some constant k. Specifically, When $y_1 \neq y_2$, we have

$$\begin{cases} y_1 = 1, y_2 = -1 & \Longrightarrow \quad \alpha_1 - \alpha_2 = k \Rightarrow \quad \alpha_2 = \alpha_1 - k \\ y_1 = -1, y_2 = 1 & \Longrightarrow \quad -\alpha_1 + \alpha_2 = k \Rightarrow \quad \alpha_2 = \alpha_1 + k \end{cases}$$

and the graph of α_1 and α_2 is as Fig. 2a. When $y_1 = y_2$ we have

$$\begin{cases} y_1 = 1, y_2 = 1 & \Longrightarrow \quad \alpha_1 + \alpha_2 = k \Rightarrow \quad \alpha_2 = -\alpha_1 + k \\ y_1 = -1, y_2 = -1 & \Longrightarrow \quad -\alpha_1 - \alpha_2 = k \Rightarrow \quad \alpha_2 = -\alpha_1 - k \end{cases}$$

and the graph corresponds to Fig. 2b.

From $\alpha_1 y_1 + \alpha_2 y_2 = k$ we can write $\alpha_2 = (k - \alpha_1 y_1) y_2$. We substitute it into W to get

$$W(\alpha_1, \alpha_2, \ldots, \alpha_n) = W(\alpha_1, (k - \alpha_1 y_1) y_2, \bar{\alpha}_3, \ldots, \bar{\alpha}_n) = W(\alpha_1),$$

which is a quadratic function of α_1. We can set its derivative to zero and obtain the "unclipped" solution α_1^*. From Fig. 2 we see that $L \leq \alpha_1 \leq H$ for some lower bound L and upper bound H. If $\alpha_1^* < L$, then we set it to L, while if $\alpha_1^* > H$ then we set it to H. Having found α_1^*, we can then use Eq. (27) to obtain the optimal solution α_2^*.

We see that the SMO algorithm is very straightforward: at each step, we pick some α_i and α_j to update next. We fix other variables, and then we will obtain an analytic solution (α_i^*, α_j^*). We keep doing this until some convergence criteria is met.

7.3 Kernels

After we obtained the solution $\boldsymbol{\alpha}^* = (\alpha_1^*, \ldots, \alpha_n^*)$ for the dual problem, we can obtain the solution for $\boldsymbol{w}$ according to Eq. (18) as

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i.$$

The optimal intercept b^* can also be obtained. To make prediction, we calculate

$$\begin{aligned} f(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x} + b^* &= \left(\sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i\right)^T \boldsymbol{x} + b^* \\ &= \sum_{i=1}^{n} \alpha_i^* y_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + b^*. \end{aligned} \tag{28}$$

Observe that, in solving the dual problem Eqs. (24) to (26), and in the prediction Eq. (28), we only calculated the inner products between the inputs. The idea of kernels is to substitute the inner product $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ by a kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \varphi(\boldsymbol{x}_i), \varphi(\boldsymbol{x}_j) \rangle$ for some mapping $\varphi : \mathbb{R}^n \to \mathbb{R}^N$, where N is typically much higher than n, and is possibly infinity. In this way, we map the input to some higher dimensional space, so that the data may become linearly separable, but without first calculating and storing the transformations. Mercer's theorem says that, roughly, if K is symmetric and positive definite, meaning

$$\int \int K(x, y) f(x) f(y) dx dy \geq 0$$

for square integrable functions f, then there is φ such that $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$.

Popular kernels:

- d-th degree polynomial: $K(x, x') = (1 + \langle \boldsymbol{x}, \boldsymbol{x}' \rangle)^d$,
- Radial basis: $K(x, x') = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2)$,
- Neural network: $K(x, x') = \tanh(\kappa_1 \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \kappa_2)$.

7.4 SVM for More than two classes

There are two strategies to apply SVM to multi-class classification problems:

- *One-versus-all*: fit K different two class SVM classifiers $\{f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})\}$, each class versus the rest, then classify $\boldsymbol{x}$ to the class for which $f_i(\boldsymbol{x})$ is the largest.
- *One-versus-one*: fit all the $\binom{K}{2}$ pairwise classifiers $f_{ij}(\boldsymbol{x})$. Then classify $\boldsymbol{x}$ to be the class that wins most of the pairwise competitions.

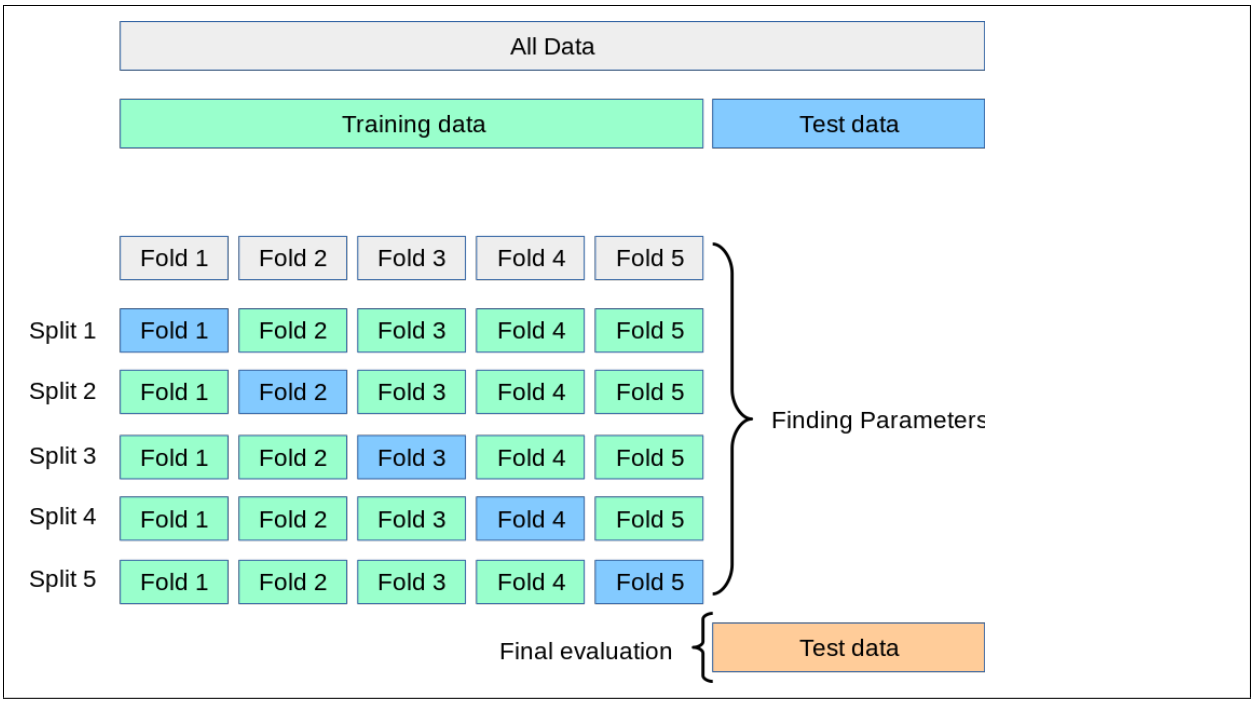


Figure 3: Illustration of cross-validation. Figure from https://scikit-learn.org/stable/modules/cross_validation.html.

8 Model Evaluation

8.1 Cross-Validation

Cross-validation is a way for estimating the test MSE or *any test metric* (like accuracy, AUC, mean absolute error etc.) without sacrificing too much data in the training set. In k-fold cross validation, the training set is randomly divided into k folds. For each fold i, we fit the model on all other folds, and then compute the metric $score_i$ on fold i. We do the same for all k folds, and our final output is the average of all $score_i$'s:

$$score = \frac{1}{k}\sum_{i=1}^{k} score_i.$$

See Fig. 3 for illustration. We may have these *two purposes* for performing cross-validation:

(1) in some situations, we want to know how well a given statistical learning procedure can be expected to perform on independent data; in this case, the actual estimate of the metric is of interest;

(2) but at other times we are interested only in the location of the minimum point in the estimated test metric curve. This is because we might be performing cross-validation

on a number of statistical learning methods, or on a single method using different levels of flexibility, in order to identify the method that results in the lowest test error (e.g. when we are seeking the optimal hyper-parameters). For this purpose, the location of the minimum point in the estimated test metric curve is important (e.g. the hyper-parameter), but the actual value of the estimated test metric curve is not.

There is a bias-variance trade-off for cross-validation:

- Since each time we use less data to train the model, the model can perform slightly poorly on each fold, so our estimate of the test metric may be biased upward (e.g. we may estimate a large test error while the true test error may not be as large).
- This bias is minimized when $k = n$, i.e. the *leave-one-out (LOO)* cross validation. However, LOO often results in high variance as an estimator for the test error. Intuitively, since $n - 1$ of the n samples are used to build each model, models constructed from folds are virtually identical to each other and to the model built from the entire training set. Also, LOO is computationally far more expensive (however in the case of linear or polynomial regression, there is a direct formula for cross-validated *MSE*:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{y_i - \hat{y}_i}{1 - h_i}\right]^2$$

where $\hat{y}_i$ is the prediction for y_i from the OLS fit on all the data, and h_i is the ith diagonal element in the projection matrix $\boldsymbol{H}$. This is like the classical MSE, but the ith residual is divided by $1 - h_i$.)

As a general rule, 5-fold or 10-fold cross-validation should be preferred.

There are also many variants of k-fold cross validation. For example, StratifiedKFold is often used for classification problems. It returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set. More variants can be viewed at `https://scikit-learn.org/stable/modules/cross_validation.html`.

8.2 ROC curve

The receiver operating characteristic (ROC) curve is commonly used in machine learning to measure the performance of a binary classifier. We assume that each label is represented as a number in $\{0, 1\}$, where 0 is the "negative" label and 1 is the "positive" label. We use the data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ to learn a function $s : \mathbb{R}^p \to [0, 1]$ that assigns probability scores to its inputs. The classification rule is

$$C_\tau(\boldsymbol{x}) = \begin{cases} 1 & s(\boldsymbol{x}) > \tau, \\ 0 & s(\boldsymbol{x}) \leq \tau. \end{cases}$$

There are four cases:

- If $y_i = C_\tau(\boldsymbol{x}_i) = 1$ then the sample $(\boldsymbol{x}_i, y_i)$ is a *true positive (TP)*.
- If $y_i = C_\tau(\boldsymbol{x}_i) = 0$ then the sample $(\boldsymbol{x}_i, y_i)$ is a *true negative (TN)*.
- If $C_\tau(\boldsymbol{x}_i) = 1$ but $y_i = 0$ then the sample $(\boldsymbol{x}_i, y_i)$ is a *false positive (FP)*.
- If $C_\tau(\boldsymbol{x}_i) = 0$ but $y_i = 1$ then the sample $(\boldsymbol{x}_i, y_i)$ is a *false negative (FN)*.

The ROC curve is a plot of the ***true positive rate (TPR)*** against the ***false positive rate (FPR)*** as τ varies from 1 to 0. The *true positive rate* is how much the classifier gets it *right* among the positive samples. It is defined as the size of the true positives divided by the size of all positive samples, which consist of both true positives and false negatives:

$$TPR = \frac{\#\text{True Positive Samples}}{\#\text{Positive Samples}} = \frac{|TP|}{|TP| + |FN|}.$$

The *false positive rate* is how much the classifier gets it *wrong* among the negative samples. It is defined as the size of the false positives divided by the size of all negative samples, which consist of both false positives and true negatives:

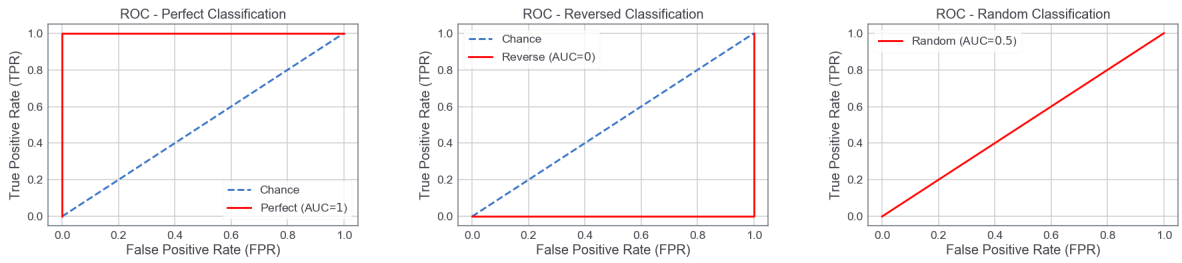
$$FPR = \frac{\#\text{False Positive Samples}}{\#\text{Negative Samples}} = \frac{|FP|}{|FP| + |TN|}.$$

At $\tau = 1$, we classify every sample as being negative ($C_1(x) \equiv 0$). No positive sample is correctly classified, so true positive rate is 0. Also in this situation we are able to correctly predict the labels of all negative samples, so the false positive rate is also 0. Thus the curve passes the point $(0, 0) \in \mathbb{R}^2$.

At $\tau = 0$, we classify every sample as positive ($C_0(x) \equiv 1$). In particular we classify every positive sample as positive, so the true positive rate is 1. At the same time we get every negative sample wrong, so the false positive rate is also 1. Thus the curve passes the point $(1, 1) \in \mathbb{R}^2$.

In general, the closer the ROC curve is to the upper left corner, the better. If the ROC curve passes point $(0, 1) \in \mathbb{R}^2$, as in Fig. 4a, it means that the classifier is able to achieve 0% false positive rate and 100% true positive rate, which implies that the model does perfect classification: it classifies every positive sample as positive and every negative sample as negative. If, on the other hand, the ROC curve is like Fig. 4b, then the model classifies every positive sample as negative and every negative sample as positive. But this is actually not a bad thing: we just need to reverse the prediction for every sample and then we can get perfect classification. If the ROC curve is close to the 45 degree line (Fig. 4c), then it means that the model is not much better than random classification.

A natural metric for comparing ROC curve is the *area under the ROC curve (AUC)*. $AUC = 1$ corresponds to the perfect classification case, and $AUC = 0$ corresponds to the



(a) $AUC = 1$. (b) $AUC = 0$. (c) $AUC = 0.5$.

Figure 4: ROC curve for three extreme cases.

reversed classification case. The closer the AUC is to 1, the better. AUC score also has a probabilistic interpretation: *It is the probability that the classifier ranks a randomly chosen positive sample higher than a randomly chosen negative sample.* We can justify this assertion as follows: let the random variable $s(X^{pos}) \in [0,1]$ denote the output score for a random positive sample X^{pos} and let f^{pos} denote its density function. Similarly let $s(X^{neg}) \in [0,1]$ denote the output score for a random negative sample X^{neg} and let f^{neg} denote its density. The true positive rate as a function of the threshold τ is

$$T(\tau) := \mathbb{P}(s(X^{pos}) > \tau) = \mathbb{E}[\mathbb{I}(s(X^{pos}) > \tau)] = \int_{\tau}^{1} f^{pos}.$$

Similarly the false positive rate is

$$F(\tau) := \mathbb{P}(s(X^{neg}) > \tau) = \mathbb{E}[\mathbb{I}(s(X^{neg}) > \tau)] = \int_{\tau}^{1} f^{neg}.$$

We have

$$\begin{aligned} AUC &= \int_{1}^{0} T(\tau) dF(\tau) = \int_{1}^{0} T(\tau) d\left(\int_{\tau}^{1} f^{neg}\right) \\ &= \int_{0}^{1} T(\tau) f^{neg}(\tau) d\tau = \int_{0}^{1} \left(\int_{\tau}^{1} f^{pos}\right) f^{neg}(\tau) d\tau \\ &= \int_{0}^{1} \mathbb{E}[\mathbb{I}(s(X^{pos}) > \tau)] \cdot f^{neg}(\tau) d\tau = \mathbb{E}[\mathbb{I}(s(X^{pos}) > s(X^{neg}))] \\ &= \mathbb{P}\left(s(X^{pos}) > s(X^{neg})\right). \end{aligned}$$

References

Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine". *The Annals of Statistics* 29.5, pp. 1189–1232 (cit. on pp. 25, 32).

James, G. et al. (2014). *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York (cit. on p. 4).

Platt, J. (1998). "Sequential minimal optimization: A fast algorithm for training support vector machines" (cit. on p. 35).