XIU SHENGJIE

A GUIDE TO ML

1
Overview of Supervised Learning

This chapter is developed on Chapter 2: Overview of Supervised Learning of ESLII, the note by Prof. Jing, and Chapter 3: Linear Basis Function Models of PRML.

To consider a supervised learning problem from a probabilistic view, we assume that the target variable y is given by a deterministic function $f(\mathbf{x})$ with additive Gaussian noise

$$y = f(\mathbf{x}) + \epsilon,$$

where $f(\mathbf{x}) = \mathrm{E}(y|\mathbf{x})$. Our objective is to find $f(\cdot)$ given training data $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

A model selection problem can be usually formulated as follows

$$\beta^* = \arg\min_{\beta} \{\mathrm{Loss}(y, f(\mathbf{x}; \beta)) + \lambda \cdot \mathrm{Penalty}(\beta)\}.$$

1.1 Loss Function

- Regression: L_2-loss, L_1-loss, robust loss $\rho(y - f(\mathbf{x}; \beta))$.
- Classification (binary): misclassification error, entropy, deviance, L_2-loss, hinge loss (SVM).

To be noticed, different loss functions lead to different estimates. For example, the optimal estimate $f(\cdot)$ is $\mathrm{E}(y|\mathbf{x})$ under L_2-loss, while is $\mathrm{med}(y|\mathbf{x})$ under L_1-loss.

1.2 Prediction Function

For high-dimensional data, it would be difficult to estimate $f(\cdot)$ because of the curse of dimensionality. If we can impose some structures on $f(\cdot)$, it would be more possible. Popular structures include:

- Linear regression model:
$$f(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\beta}$$
- Generalized linear model (GLIM):
$$g(f(\mathbf{x})) = \mathbf{x}^T \boldsymbol{\beta}, \qquad (g \text{ known}),$$
e.g. Logistic regression model: $\mathrm{logit}(p(\mathbf{x}))$.
- Kernel linear regression model:
$$f(\mathbf{x}) = h(\mathbf{x})^T \boldsymbol{\beta}, \qquad (h \text{ known})$$

- Additive model:

$$f(\mathbf{x}) = h_1(x_1) + \cdots + h_p(x_p),\ \mathbf{x} \in R^p \qquad (h_i\text{'s unknown})$$

- Generalized additive model (GAM):

$$g\left(f(\mathbf{x})\right) = h_1(x_1) + \cdots + h_p(x_p), \qquad (g \text{ known}, h_i\text{'s unknown})$$

- Single index model:

$$f(\mathbf{x}) = h(\mathbf{x}^T\boldsymbol{\beta}), \qquad (h \text{ unknown})$$

- Neural networks:

$$f(\mathbf{x}) = h_1(\mathbf{x}^T\boldsymbol{\beta}_1) + \cdots + h_k(\mathbf{x}^T\boldsymbol{\beta}_k), \qquad (h_i\text{'s unknown})$$

2
Linear Models for Regression

This chapter is developed on Chapter 3: Linear Methods for Regression of ESLII, and the note by Prof. Jing.

2.1 Ordinary Least Squares

The OLS has the objective:

$$\begin{aligned}\hat{\boldsymbol{\beta}}^{\text{ols}} &= \arg\min \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},\end{aligned} \tag{2.1}$$

where each row of $\mathbf{X}$ is one observation $\mathbf{x}_i$.

2.2 Shrinkage Methods

Problems with OLS:

- If $\mathbf{X}^T\mathbf{X}$ is not of full rank, i.e., the columns of $\mathbf{X}$ are highly correlated ($\mathbf{X} \in R^{N\times p}, N \geq p$), the estimate of $\boldsymbol{\beta}$ becomes unstable.
- Lower bias, higher variance, and hard to interpret when more features are considered.

By shrinkage, we can solve the problems above. There are two groups of shrinkage methods: subset and shrinking coefficients. We will focus on shrinking coefficients by adding regularization terms.

2.2.1 Ridge Regression

$$\text{Ridge Regression} = \text{OLS} + L_2 \text{ penalty}$$

Now the solution is different to $\hat{\boldsymbol{\beta}}^{\text{ols}}$

$$\begin{aligned}\hat{\boldsymbol{\beta}}^{\text{ridge}} &= \arg\min \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2 \\ &= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}\end{aligned} \tag{2.2}$$

by making the problem nonsingular, even if $\mathbf{X}^T\mathbf{X}$ is not of full rank.

Another view of ridge regression is from the SVD of $\mathbf{X}$: $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. For OLS we have

$$\begin{aligned}\mathbf{X}\hat{\boldsymbol{\beta}}^{\text{ols}} &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{U}\mathbf{U}^T\mathbf{y}\end{aligned},$$

after some simplification[1], while for ridge regression we have

$$\begin{aligned}\mathbf{X}\hat{\boldsymbol{\beta}}^{\text{ridge}} &= (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{U}\mathbf{D}(\mathbf{D}^2+\lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{y} \\ &= \sum_{j=1}^{p}\frac{d_j^2}{d_j^2+\lambda}\mathbf{u}_j\mathbf{u}_j^T\mathbf{y},\end{aligned}$$

which means that basis vectors with smaller singular values are shrunk more.

[1] Recall that $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices, which $\mathbf{D} \in R^{N\times p}$.

2.2.2 *Lasso*

$$\text{Lasso} = \text{OLS} + L_1 \text{ penalty}$$

Lasso is defined as

$$\hat{\boldsymbol{\beta}}^{\text{lasso}} = \arg\min \|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1, \tag{2.3}$$

which usually doesn't has a closed-form solution.

2.2.3 *Comparison of Shrinkage Methods*

(1) *Shrinking patterns*

In the case of orthonormal features, the closed-form solutions of best subset, ridge regression, and the lasso are summarized in Table 2.1 and Figure 2.1.

Estimator	Formula
Best subset (size M)	$\hat{\boldsymbol{\beta}}_j^{\text{ols}} \cdot \mathbb{I}(\lvert\hat{\boldsymbol{\beta}}_j^{\text{ols}}\rvert \geq \lvert\hat{\boldsymbol{\beta}}_{(M)}^{\text{ols}}\rvert)$
Ridge regression	$\hat{\boldsymbol{\beta}}_j^{\text{ols}}/(1+\lambda)$
Lasso	$\text{sign}(\hat{\boldsymbol{\beta}}_j^{\text{ols}})(\lvert\hat{\boldsymbol{\beta}}_j^{\text{ols}}\rvert - \lambda)_+$

Table 2.1: Closed-form solutions in the case of orthonormal features.

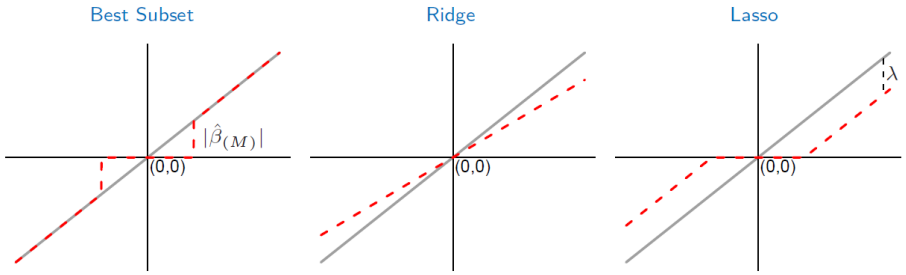

Figure 2.1: Comparison of shrinkage methods.

(2) *Grouping effect*

Ridge regression has the grouping effect: the regression coefficients of a group of highly correlated variables tend to be equal Tibshirani [1996], Zou and Hastie [2005]. The grouping effect is a feature of the L_2 penalty, also appearing in the elastic net. It is a tendency to make the coefficients equal to minimize their squared norm.

In contrast, the lasso doesn't have the grouping effect, i.e., two highly correlated features may have quite different coefficients.

(3) *Sparsity*

For (2.2)(2.3), they have equivalent forms that add a L_1 or L_2 constraint to the original OLS problem (2.1). A 2-D example is shown in Figure 2.2, where the dark zones are hard constraints. The optimal point of the lasso is highly likely to fall on the corner, which means the lasso tends to produce a sparse model. In contrast, ridge regression doesn't have this sparse representation.

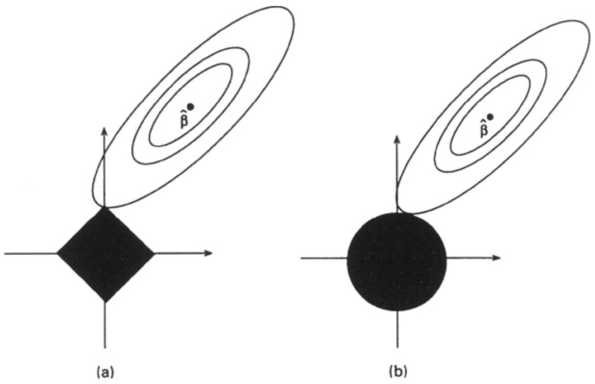

Figure 2.2: Illustration of sparsity for (a) Lasso and (b) Ridge regression.

2.2.4 *Elastic Net*

$$\text{Elastic Net} = \text{OLS} + L_1 \text{ penalty} + L_2 \text{ penalty}$$

Elastic net is a combination of the lasso and ridge regression. A naive elastic net is defined as

$$\hat{\boldsymbol{\beta}}^{\text{elastic}} = \arg\min \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda_1\|\boldsymbol{\beta}\|_1 + \lambda_2\|\boldsymbol{\beta}\|_2,$$

which produces sparse solutions and has grouping effect. It has a similar soft-thresholding effect as the lasso with a smaller slope (Figure 2.1); and a constraint lies between L_1 and L_2. Surprisingly, bridge regression $L_q, (1 < q < 2)$ doesn't produce sparse solutions Zou and Hastie [2005].

2.2.5 *Group Lasso*

The group lasso divides the p features into L groups and has the following form

$$\hat{\boldsymbol{\beta}}^{\text{group}} = \arg\min \|\mathbf{y} - \sum_{l=1}^{L} \mathbf{X}_l\boldsymbol{\beta}_l\|_2^2 + \lambda \sum_{l=1}^{L} \sqrt{p_l}\|\boldsymbol{\beta}_l\|_2.$$

ESLII claims that the group lasso encourages sparsity at the group and individual levels. The group lasso indeed prefers fewer active groups so that fewer "weights" $\sqrt{p_l}$'s are counted. However, Friedman et al. [2010] claim that it doesn't yield sparsity within a group unless an additional penalty $\lambda_2\|\boldsymbol{\beta}\|_1$ is added.

3
Linear Models for Classification

This chapter is developed on Chapter 3: Linear Models for Classification of PRML.

Linear models for classification: Their decision surfaces are linear functions of the input vector $\mathbf{x}$ and hence are defined by $(D-1)$-dimensional hyperplanes within the D-dimensional input space. We focus on finding this hyperplane (for two classes) or these hyperplanes (for multiple classes).

There are three main kinds of approaches for classifications:

1. Discriminant functions: least squares, perception, and FDA
2. Probabilistic generative models: LDA and QDA
3. Probabilistic discriminative models: MLE$\rightarrow$ iterative reweighed least squares

Simply speaking,

- 1 don't consider probability. They control the decision boundary directly via $\mathbf{w}$.
- 2 focus on the modeling of class-conditional densities $p(\mathbf{x} \mid C_k)$ and prior class probabilities $p(C_k)$, so that we can make decision based on the calculated posterior probabilities $p(C_k \mid \mathbf{x})$ (the decision boundary is the ratio of $p(C_k \mid \mathbf{x})$). They control the decision boundary indirectly via model parameters.
- 3 focus on the probabilistic view of the decision boundary (inspired by 2) and control the decision boundary directly via $\mathbf{w}$.

In *PRML*, 1 focus on the simplest form $y(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x} + w_0$ and 3 consider *generalized linear models* $y(\mathbf{x}) = f\left(\mathbf{w}^\intercal \mathbf{x} + w_0\right) \rightarrow \sigma(\mathbf{w}^\intercal \boldsymbol{\phi})$.[1] 2 look different because they focus on tuning prior class probabilities $p(C_k)$ and class-conditional densities $p(\mathbf{x} \mid C_k)$, and then computing posterior probabilities $p(C_k \mid \mathbf{x})$, without direct touch of hyperplanes $y(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x} + w_0$.

[1] $f(\cdot)$ is known as an *activation function* that is often nonlinear, however the decision surfaces $\mathbf{w}^\intercal \mathbf{x} + w_0 = \text{const}$ are always linear. Probabilistic discriminative models in PRML make two specifications: 1) First make a fixed nonlinear transformation of the input $\mathbf{x}$ to produce $\boldsymbol{\phi}$ (change feature space); 2) $f(\cdot)$ is a sigmoid or softmax function.

3.1 *Discriminant Functions*

Both discriminant functions and probabilistic discriminative models are based on the discriminative models. Consider a multiclass problem. We identify the relationship between discriminant functions and probabilistic discriminative models.

Both models produce a "score" for each class (a score vector). 1 apply $y_k(\mathbf{x}) = \mathbf{w}_k^\intercal \mathbf{x} + w_{k0}$ [2] and 3 apply $\mathbf{w}_k^\intercal \boldsymbol{\phi}$ [3]. They have the same meaning. Both can be viewed as the distance from the hyperplane, i.e., a higher score of Class K means the deeper inside the K region.

[2] a compact form $y_k(\mathbf{x}) = \widetilde{\mathbf{w}}^\intercal \widetilde{\mathbf{x}}$

[3] $\phi_0(\mathbf{x}) = 1$ so the corresponding parameter w_0 plays the role of a bias

The scores cannot be interpreted as probabilities[4]. This being the case, 1 can only assign the class label with the highest score to the input vector $\mathbf{x}$, denoted C_k. 3 further apply a *logistic sigmoid* function (to two classes) or a *softmax* function (to multiple classes) on these scores to produce the probabilities. So 3 outputs each class's probability and allows us to make a decision, although usually we prefer the class with the highest probability.

[4] Equation (4.18-4.19) in PRML

Discriminant functions are irrelevant to probability. We can use least squares, Fisher's linear discriminant, and the perception algorithm instead of MLE to learn the models.

3.1.1 *Making Decision*

A linear discriminant function for two classes can be written as

$$y(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x} + w_0,$$

where $\mathbf{x}$ is an input vector, $\mathbf{w}$ is a weight vector that control the orientation of the hyperplane, and w_0 is a bias. The input is assigned to class C_1 if $y(\mathbf{x}) \geq 0$. It has a more compact form

$$y(\mathbf{x}) = \widetilde{\mathbf{w}}^\mathsf{T}\widetilde{\mathbf{x}},$$

where we introduce a dummy input $x_0 = 1$ and then define $\widetilde{\mathbf{w}} = (w_0, \mathbf{w})$ and $\widetilde{\mathbf{x}} = (x_0, \mathbf{x})$.

For multiple classes, we will consider a single K-class discriminant comprising K linear functions of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^\mathsf{T}\mathbf{x} + w_{k0}$$

and then assigning a point $\mathbf{x}$ to class C_k with the biggest $y_k(\mathbf{x})$.

3.1.2 *Learning Algorithms*

The key to learning algorithms is how we construct the optimization objective functions.

1. Least squares considers the distance between prediction and ground truth. It uses 1-of-k binary coding scheme to represent ground truth and compute the L2-norm of the error between output score vectors and ground truth.

2. Perception algorithm is designed for binary classification problems. It also considers the distance between prediction and ground truth. Instead of using the score directly, perception puts it into a step function, a nonlinear activation function. Then we can compare the binary prediction with the binary ground truth.

3. Fisher's linear discriminant transforms the linear discriminant result into a scalar Fisher criterion. This criterion is the ratio of a between-class covariance matrix and a within-class covariance matrix.

The **least squares** has a closed-form solution. However, it lacks robustness to outliers because it corresponds to maximum likelihood under the assumption of a Gaussian conditional distribution, whereas binary target vectors have a distribution far from Gaussian[5].

[5] Recall that Student's t-distribution is more robust to outliers compared with Gaussian distribution.

The **perception algorithm** doesn't have a closed-form solution because it introduces a nonlinear activation function. It uses a stochastic gradient descent algorithm

that has other problems: 1) It cannot guarantee the error decreases at each step; 2) If training data is not linearly separable, it never converge; 3) If training data is linearly separable, it may produce many solutions.

Fisher's linear discriminant (FDA) regards a linear classification model as dimensionality reduction. Consider a binary classification problem, the Fisher criterion is

$$\max J(\mathbf{w}) = \frac{\mathbf{w}^\intercal \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\intercal \mathbf{S}_W \mathbf{w}},$$

where $\mathbf{S}_B$ is the between-class covariance defined by[6]

$$\mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\intercal,$$

[6] $\cdot,\cdot$ distance of two projected center is $\|\mathbf{w}^\intercal \boldsymbol{\mu}_1 - \mathbf{w}^\intercal \boldsymbol{\mu}_2\|_2^2$.

and $\mathbf{S}_W$ is the within-class covariance defined by

$$\mathbf{S}_W = \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^\intercal + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^\intercal.$$

In a binary classification problem, Fisher's linear discriminant is the same as least squares.

Consider a K-class problem with the equation,

$$\mathbf{y} = \mathbf{W}^\intercal \mathbf{x},$$

where $\mathbf{W}$ is a $D \times K$ weight matrix whose columns are the weight vectors $\{\mathbf{w}_k\}$, and D is the dimension of the input space ($D \geq K$). One generalization of Fisher criterion is,

$$J(\mathbf{W}) = \operatorname{Tr}\left\{ \left(\mathbf{W}^\intercal \mathbf{S}_W \mathbf{W}\right)^{-1} \left(\mathbf{W}^\intercal \mathbf{S}_B \mathbf{W}\right) \right\},$$

where

$$\mathbf{S}_B = \sum_{k=1}^{K} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^\intercal.$$

It measures the convergence of within-class data and the separation of between-class data.

3.2 *Probabilistic Generative Models*

The workflow of constructing probabilistic generative models is: 1) model the class-conditional densities $p(\mathbf{x} \mid C_k)$; 2) model the class probabilities $p(C_k)$; 3) compute the posterior probabilities $p(C_k \mid \mathbf{x})$.

3.2.1 *Making Decision*

For the binary classification case, the posterior probability for class C_1 can be written as

$$\begin{aligned} p(C_1 \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid C_1) p(C_1)}{p(\mathbf{x} \mid C_1) p(C_1) + p(\mathbf{x} \mid C_2) p(C_2)} \\ &= \frac{1}{1 + \exp(-a)} = \sigma(a), \end{aligned}$$

where we define

$$a = \ln \frac{p(\mathbf{x} \mid C_1) p(C_1)}{p(\mathbf{x} \mid C_2) p(C_2)}, \tag{3.1}$$

and $\sigma(a)$ is the *logistic sigmoid* function. The message is that the posterior probability can be written in a sigmoid form with a in (3.1). a is also the **decision boundary** of C_1 and C_2.

For the multiclass case, we have

$$p(C_k \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid C_k)p(C_k)}{\sum_j p(\mathbf{x} \mid C_j)p(C_j)} = \frac{\exp(a_k)}{\sum_j \exp(a_j)},$$

where we define

$$a_k = \ln\left(p(\mathbf{x} \mid C_k)p(C_k)\right). \tag{3.2}$$

The message is that a the posterior probability can be written in a *softmax* form with a_k in (3.2), also regarded as a multiclass generalization of the logistic sigmoid.

For continuous inputs (feature space), a in (3.1) and a_k in (3.2) can be written in a linear function of $\mathbf{x}$

$$a = \mathbf{w}^\intercal \mathbf{x} + w_0, \tag{3.3}$$

$$a_k = \mathbf{w}_k^\intercal \mathbf{x} + w_{k0}, \tag{3.4}$$

which are the same for discrete inputs.[7] In fact, if we assume the class-conditional densities are members of the exponential family of distributions, the posterior class probabilities are given by generalized linear models with logistic sigmoid or softmax activation functions.

[7] Section 4.2.1 of PRML

Note: We may not compute $\mathbf{w}$ and w_0 explicitly because they are "secondary" parameters (i.e. a complicated combination of "primary" parameters in $p(\mathbf{x} \mid C_k)$ and $p(C_k)$). But they inspire the design of probabilistic discriminative models that regards $\mathbf{w}$ and w_0 as "primary" parameters without interests in $p(\mathbf{x} \mid C_k)$ and $p(C_k)$.

Specifically, **linear discriminant analysis (LDA)** and **quadratic discriminant analysis (QDA)** use multivariate Gaussian densities $p(\mathbf{x} \mid C_k)$. LDA assume that the classes have a common covariance matrix $\mathbf{\Sigma}_k = \mathbf{\Sigma}, \forall k$. It is called "linear" because its decision boundary

4.3 Linear Discriminant Analysis of ESLII

$$\begin{aligned}\ln \frac{p(\mathbf{x} \mid C_k)p(C_k)}{p(\mathbf{x} \mid C_j)p(C_j)} &= \ln \frac{p(\mathbf{x} \mid C_k)}{p(\mathbf{x} \mid C_j)} + \ln \frac{p(C_k)}{p(C_j)} \\ &= \ln \frac{p(C_k)}{p(C_j)} - \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_j)^\intercal \Sigma^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_j) + \mathbf{x}^\intercal \Sigma^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_j)\end{aligned}$$

is linear in $\mathbf{x}$. QDA doesn't assume $\mathbf{\Sigma}_k$'s are equal, resulting in a quadratic decision boundary.

3.2.2 *Learning Algorithms*

To find the LDA of the binary classification, we can write the posterior probability of the labels $\mathbf{t}$ denoted as

$$p(\mathbf{t} \mid \mathbf{x}; \pi_1, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \mathbf{\Sigma}) = \cdots,$$

and then find the parameters via MLE. This is a general method for all probabilistic generative models.

3.3 *Probabilistic Discriminative Models*

The advantages of probabilistic discriminative models compared with probabilistic generative models include 1) Fewer adaptive parameters to be determined; 2) Improve predictive performance when true class-conditional density is hard to be

approximated. Probabilistic discriminative models are a combination of the linear function of $\mathbf{x}$ in discriminant functions and the probabilistic views in probabilistic generative models.

3.3.1 *Making Decision*

All algorithms in this chapter are equally applicable if we first make a fixed nonlinear transformation of the inputs using a vector of basis functions $\boldsymbol{\phi}(x)$. The resulting decision boundaries will be linear in the feature space $\boldsymbol{\phi}$, and these correspond to nonlinear decision boundaries in the original $\mathbf{x}$ space. Typically, the first basis function is set to a constant, like $\phi_0(\mathbf{x}) = 1$, so the corresponding parameter w_0 plays the role of a bias.

Inspired by (3.3), for binary classification, the approach has the form

$$p(C_1 \mid \boldsymbol{\phi}) = \sigma(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}),$$

named as **logistic regression**.

In fact, if the activation function is a step function instead of a logistic sigmoid function, the classifier becomes a perception ($p(C_1 \mid \boldsymbol{\phi}) = \{0, 1\}$) that is difficult for optimization.

One benefit of the logistic regression is that it is more robust to outliers. Once correctly classified, the outlier has a limited contribution, as shown in Figure 3.1, compared with the least squares.

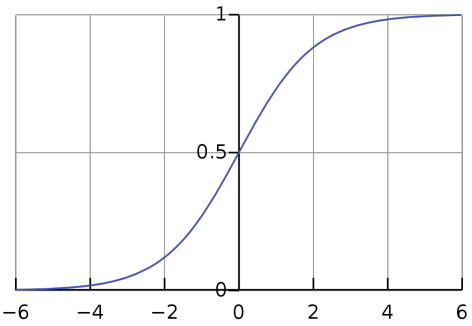

Figure 3.1: Logistic sigmoid function. [Wikipedia]

One minor problem is that if training data is linearly separable, the logistic regression may not converge. The reason is that the logistic regression continues maximizing the likelihood by shaping the logistic curve into a perception curve. If training data is not separable, the logistic regression can converge, opposite to the perception algorithm.

Inspired by (3.4), for multiclass problem, there is a **multiclass logistic regression (softmax)** as

$$p(C_k \mid \boldsymbol{\phi}) = \frac{\exp(\mathbf{w}_k^\mathsf{T}\boldsymbol{\phi})}{\sum_j \exp(\mathbf{w}_j^\mathsf{T}\boldsymbol{\phi})}.$$

Notice: If the class-conditional densities aren't members of the exponential family, like Gaussian mixtures, we do not have logistic (or softmax) transformation acting on a linear function of the feature variables like (3.1)(3.2), and we need to consider other types of models such as *probit regression*.[8]

[8] for binary classification; use Cumulative Distribution Function (CDF) as the activation function; Section 4.3.5 of PRML

3.3.2 *Learning Algorithms*

Like the learning algorithm of probabilistic generative models, we can also write the conditional probability, but conditioned on $\mathbf{w}$ this time,

$$p(\mathbf{t} \mid \mathbf{x}; \mathbf{w}) = \cdots .$$

Our objective is to find the optimal $\mathbf{w}$ that achieves the maximum likelihood. The negative log-likelihood is also called logistic error function. See Section 4.2.1 for details.

However, because of the nonlinearity of the logistic sigmoid function, there is no closed-form solution. Instead, we can use *Newton's method*. Because it has an update formula that takes the form of a set of normal equations for a weighted least-squares problem, this method is also called **iterative reweighed least squares.**

4
Sparse Kernel Machines

In PRML, the author names the maximal margin classifier, the support vector classifier, and the support vector machine as "maximum margin classifiers". Here we follow the definitions from ISLR to avoid confusion.

4.1 Maximum Margin Classifiers

They focus on the linear separable classification problem. Let's return to the binary classification problem using a hyperplane of the form

$$f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}) + b = 0$$

where $\boldsymbol{\phi}(\mathbf{x})$ denotes a fixed feature-space transformation, and we have made the bias parameter b explicit. The training data set comprises N input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$, with corresponding target values $y_1, \ldots, y_N$ where $y_n \in \{-1, 1\}$.

It can be easily shown that the perpendicular distance of a point $\mathbf{x}$ from the hyperplane is

$$\frac{|f(\mathbf{x})|}{\|\mathbf{w}\|} = \frac{y_n\left(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n) + b\right)}{\|\mathbf{w}\|}.$$

Then the maximum margin classifiers is given by

$$\arg\max_{\mathbf{w},b} \quad \left\{\frac{1}{\|\mathbf{w}\|}\min_n\left[y_n\left(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n) + b\right)\right]\right\},$$

where the **margin** is defined as the minimal distance from the observations to the hyperplane.[1]

This can be further simplified as the following quadratic programming problem,

$$\begin{aligned} \arg\min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_n\left(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n) + b\right) \geq 1,\ n = 1, \ldots, N, \end{aligned} \tag{4.1}$$

by rescaling $\mathbf{w}$ and b to $\kappa\mathbf{w}$ and κb to set[2]

$$y_n\left(\mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x}_n) + b\right) = 1$$

for the point that is closest to the surface without changing the real distances, or in a more general form:

$$\min \quad \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{n=1}^{N} E_\infty(y_n f(\mathbf{x}_n) - 1),$$

where E_∞ is a hard margin constraint.

[1] Later in support vector classifiers, we allow some observations to fall into the margin. The margin is no longer the minimal distance, and the margin borders are virtual lines parallel to the hyperplane.

[2] Scaling the direction vector doesn't change the absolute position of the hyperplane, but puts the absolute distance of each point to the hyperplane in a new scale.

The complementary slackness of (4.1) is

$$a_n \{y_n f(\mathbf{x}_n) - 1\} = 0.$$

Any data points for which $a_n = 0$ will not plays role in the classifiers. The remaining data points are called **support vectors** that satisfy $y_n f(\mathbf{x}_n) = 1$.

4.2 *Support Vector Classifiers*[3]

[3] Here we follow the definition in ISLR.

The generalization of the maximal margin classifier to the non-separable case is known as the support vector classifier. It has the advantages:

- Greater robustness to individual observations, and
- Better classification of most of the training observations.

The solution is to change the hard margin constraint to a soft one. We continue with the optimization problem (4.1), and introduce a slack variable $\xi_n \geq 0$ for each training data point $\mathbf{x}_n$, a linear penalty for the distance from the margin boundary. It is defined as

$$\xi_n = [1 - f(\mathbf{x}_n) y_n]_+ ,$$

which is a *hinge loss.* It has the following cases:

- $\xi_n = 0$: on or inside the correct margin boundary;
- $0 < \xi_n \leq 1$: inside the margin, but on the correct side;
- $\xi_n = 1$: on the decision boundary (hyperplane);
- $\xi_n > 1$: misclassified, on the wrong side of the hyperplane

Now the optimization problem (4.1) becomes:

$$\begin{aligned} \arg\min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \lambda \sum_{n=1}^{N} \xi_n \\ \text{s.t.} \quad & y_n f(\mathbf{x}_n) \geq 1 - \xi_n \\ & \xi_n \geq 0,\ n = 1, \ldots, N, \end{aligned}$$

with a soft margin and a trade-off hyperparameter λ. Support Vector Classifiers depends directly on the observations that either lie on the margin or violate the margin (**support vectors**), but not on the other observations that lie strictly on the correct side of the margin. A smaller "cost" λ brings a wider margin and more support vectors.

4.2.1 *Relation to Logistic Regression*

The support vector classifier's decision rule is based on a potentially small subset of the training observations (the support vectors), which means that it is quite robust to the observations far away from the hyperplane. The support vector classifier and logistic regression are closely related, as shown in Figure 4.1. Both methods have very low sensitivity to observations far from the decision boundary. In contrast, LDA introduced in Section 3.2 is sensitive because it depends on all the observations. This is the concern of the bias-variance trade-off in Section 7.1.

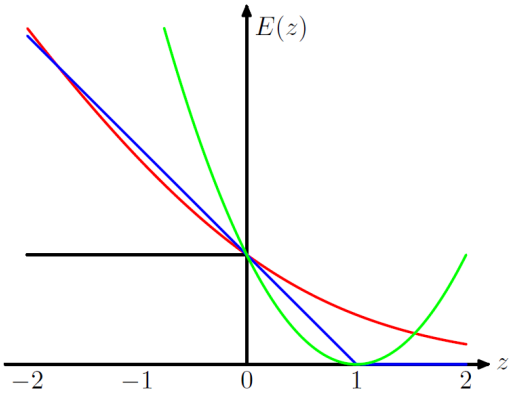

Figure 4.1: [black] misclassification error; [blue] hinge loss; [red] logistic error function (neg loglik); [green] squared error

Let's interpret Figure 4.1 in detail. For a binary classification case $y \in \{-1, 1\}$, z represents $y_n f(\mathbf{x}_n)$. SVM use the hinge loss defined as

$$\xi_n = [1 - f(\mathbf{x}_n) y_n]_+ = [1 - z]_+ .$$

For the logistic regression, there are two definitions of the loss function depending on the "negative" class is denoted as $y_n = 0$ or $y_n = -1$. For the "positive" class $y_n = 1$, we have

$$\begin{aligned} p(y_n = 1 \mid \mathbf{x}_n) &= \sigma(f(\mathbf{x}_n)) \\ &= \frac{1}{1 + \exp(-f(\mathbf{x}_n))} \\ &= \frac{\exp(f(\mathbf{x}_n))}{1 + \exp(f(\mathbf{x}_n))}, \end{aligned}$$

and for the "negative" class, we have

$$\begin{aligned} p(y_n = 0/-1 \mid \mathbf{x}) &= 1 - \sigma(f(\mathbf{x}_n)) \\ &= \frac{1}{1 + \exp(f(\mathbf{x}_n))}. \end{aligned}$$

So the likelihood function has two forms,

$$p(y_n \mid \mathbf{x}) = \begin{cases} \frac{\exp(y_n f(\mathbf{x}_n))}{1+\exp(f(\mathbf{x}_n))} & y_n = \{0, 1\} \\ \frac{1}{1+\exp(-y_n f(\mathbf{x}_n))} & y_n = \{-1, 1\}. \end{cases}$$

Learning the logistic regression is maximizing the data likelihood, which equals minimizing the negative log-likelihood. The negative log-likelihood is also called the **logistic error function**,

$$l(y_n \mid \mathbf{x}) = \begin{cases} y_n f(\mathbf{x}_n) - \log(1 + \exp(f(\mathbf{x}_n))) & y_n = \{0, 1\} \\ \log(1 + \exp(-y_n f(\mathbf{x}_n))) & y_n = \{-1, 1\}, \end{cases}$$

and the second definition is the red curve.

4.3 *Support Vector Machines*

The support vector machine (SVM) extends the support vector classifier that enlarges the feature space using kernels.

Relevant to kernel methods.

5
Basis Expansions

This chapter is developed on Chapter 5: Basis Expansions and Regularization of ESLII and the note by Prof. Jing. The key idea is that we will focus on non-linear models instead of linear models.

Broadly speaking, there are two kinds of non-linear models:

1. Basis functions / kernel method:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m h_m(\mathbf{x}),$$

   where $h_m(\mathbf{x}) : \mathbb{R}^p \mapsto \mathbb{R}$ is the new basis function to project inputs $\mathbf{x} \in \mathbb{R}^p$ to a new scalar. This is a linear basis expansion in X, and is widely used when the amount of data is small.

2. Hierarchical models:

$$f(\mathbf{x}) = f_M(f_{M-1}(\cdots f_0(\mathbf{x}))),$$

   which is the idea of deep learning. It requires big data.

In this chapter, we will focus on the first method. Different kinds of basis lead to different types of expansion. Consider x is one-dimensional,

- Polynomial regression:

$$h_m(x) = x^m$$

- Splines = piecewise polynomials:

$$f(x)\mathbb{I}(x \in R_m) = \left(\beta_{m0} + \beta_{m1}x + \beta_{m2}x^2 + \cdots + \beta_{mr}x^r\right)\mathbb{I}(x \in R_m),$$

  where $\mathbb{I}(\cdot)$ is an indicator function, and R_m is a sub region.

- Local (polynomial) regression:

$$E\left[y|x_i \in (x-\varepsilon, x+\varepsilon)\right], \tag{5.1}$$

  which regress on the neighbors of input x and do the prediction.

- Reproducing kernel Hilbert space:

- Orthogonal basis:

  - Fourier series;
  - Wavelet

5.1 Splines

Cubic splines and natural cubic splines need to select knots and become linear regression problems. In contrast, smoothing splines don't need to select knots and become ridge regression problems.

5.1.1 Cubic Splines

A cubic spline is a spline of piecewise cubic polynomials between K knots and twice continuously differentiable at all K knots. A cubic spline has a degree of freedom (d.f.) $= 4(K+1) - 3K = K+4$. A piecewise cubic polynomial has four degree of freedoms $(\beta_{m0}, \beta_{m1}, \beta_{m2}, \beta_{m2})$ if there is no constraint. Three constraints (continuous, continuous first derivative, and continuous second derivative) on K nodes will reduce the d.f. by $3K$.

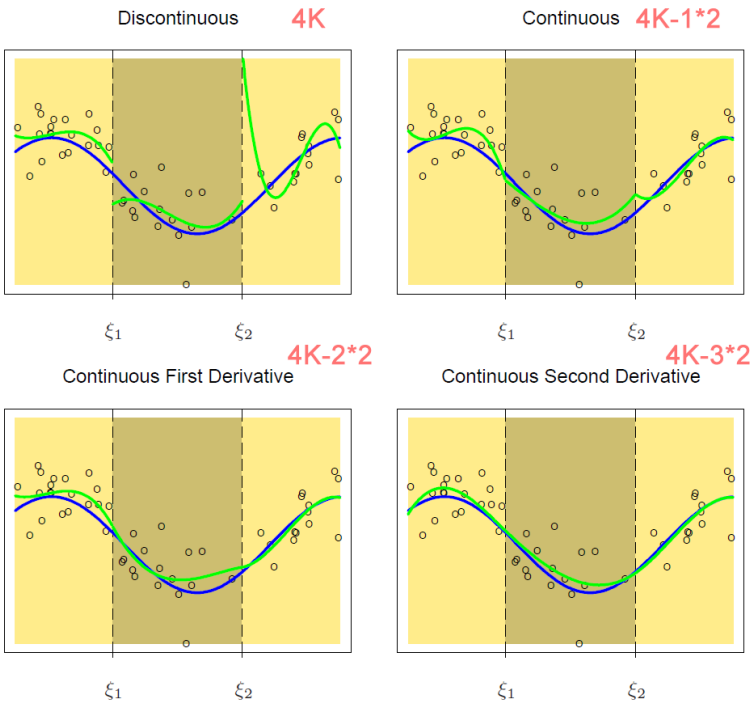

Figure 5.1: Evolution of degree of freedom. [ESLII]

The evolution of d.f. is shown in Figure 5.1. The cubic spline has the following six basis:

$$h_1(x) = 1, \quad h_3(x) = x^2, \quad h_5(x) = (x - \xi_1)_+^3$$
$$h_2(x) = x, \quad h_4(x) = x^3, \quad h_6(x) = (x - \xi_2)_+^3$$

5.1.2 Natural Cubic Splines

The major problem of cubic splines is that there is often little data in the first and last intervals, leading to overfitting. Natural cubic splines cope with this problem with further requirements: the polynomials in the first and last intervals are linear. Consequently, its d.f. is reduced from $K+4$ to K.

5.1.3 *Smoothing Splines*

Cubic splines and natural cubic splines all need to select a set of knots. In contrast, smoothing splines will use a maximal set of knots, all input values, and control the complexity by regularization. It minimizes the penalized residual sum of squares:

$$\text{RSS}(f, \lambda) = \sum_{i=1}^{N} \{y_i - f(x_i)\}^2 + \lambda \int \{f''(t)\}^2 dt.$$

The penalty term ensures that the curvature doesn't change a lot, shrinking some piecewise cubic polynomials to linear fits.

5.2 *Reproducing kernel Hilbert space*

6
Kernel Smoothing Methods

This section is developed on Chapter 6: Kernel Smoothing Methods of ESLII, and notes by Prof. Jing. In this chapter, kernels are used as a device for localization, continuing the idea of local regression (5.1). We should distinguish this from the kernel methods that computes an inner product in a high-dimensional (implicit) feature space, being used for regularized nonlinear modeling.

6.1 Nonparametric Regression Estimate

6.1.1 kNN

The first and simplest method is kNN, which takes k nearest neighbors and average them. This can be done with the nearest-neighbor kernel shown in Figure 6.1.

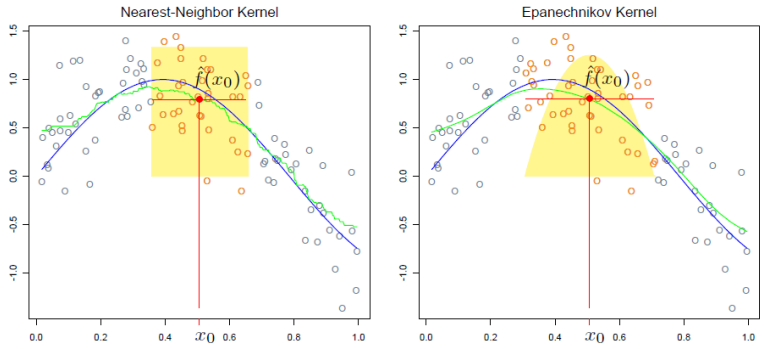

Figure 6.1: Regression kernels. [ESLII]

6.1.2 Local Constant Regression

Consider the Taylor expansion at $x \approx x_0$,

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \cdots . \tag{6.1}$$

If we want to estimate $f(x)$ near x_0 by a constant c, then our objective function is

$$\min_{c(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i) \left[y_i - c(x_0)\right]^2 ,$$

with

$$K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right),$$

where λ is the window size, and the estimate is $\hat{f}(x_0) = c(x_0)$. It has a closed-form solution called Nadaraya–Watson kernel-weighted average

$$\hat{f}(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}.$$

We can assign weights to the observations closer to the target point, using the Epanechnikov quadratic kernel shown in Figure 6.1

$$D(t) = \begin{cases} \frac{3}{4}(1 - t^2) & \text{if } |t| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$$

6.1.3 *Local Linear Regression*

From (6.1), we can derive the objective function for local linear regression

$$\min_{\alpha(x_0), \beta(x_0)} \sum_{i=1}^{N} K_\lambda(x_0, x_i) \left[y_i - \alpha(x_0) - \beta(x_0) x_i \right]^2,$$

and the estimate is $\hat{f}(x_0) = \hat{\alpha}(x_0) + \hat{\beta}(x_0) x_0$.

Higher degree polynomial regression can be conducted.

- Local linear fits can help reduce bias dramatically at the boundaries at a modest cost in variance. Local quadratic fits do little at the boundaries for bias, but increase the variance a lot.
- Local quadratic fits are more helpful in reducing bias due to curvature in the interior of the domain.

6.2 *Nonparametric Density Estimation*

It is similar to nonparametric regression estimate. It has the formula

$$\hat{f}(x_0) = \frac{1}{N\lambda} \sum_{i=1}^{N} K_\lambda(x_0, x_i).$$

The kernel $K_\lambda(x_0, x_i) = K(t)$ has the following properties:

1. $K(t)$ is a proper p.d.f. symmetrical around 0.
2. $\int K(t)dt = 1$, $\int tK(t)dt = 0$, $\int t^2 K(t)dt > 0$.

Examples of $K(t)$:

1. Uniform (Rectangle): $K(t) = \frac{1}{2}\mathbb{I}(|t| < 1)$
2. Normal (Gaussian): $K(t) = \frac{1}{2\pi} e^{-t^2/2}$

7
Model Assessment and Selection

This chapter is developed on Chapter 7: Model Assessment and Selection of ESLII. Model selection means we want to evaluate the performance of different models to choose the best one. Model assessment means after choosing one, we estimate its prediction error on new data.

If we are in a data-rich situation, the best approach for both problems is to divide the data set into a training set, a validation set, and a test set. However, if there is insufficient data, we will use the methods introduced in this chapter. They approximate the validation step analytically (AIC, BIC, MDL, SRM) or reusing samples (cross-validation and bootstrap).

7.1 The Bias-Variance Decomposition[1]

[1] From my point of perspective, this section in PRML isn't clear for readers. To introduce it, I will use the idea from ESL with the notations from PRML.

We assume that $y = f(\mathbf{x}) + \epsilon$ where $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma_\epsilon^2$. $\mathbf{y}$ is the realization, and $f(\mathbf{x})$ is the truth. We can derive an expression for the expected prediction error of a regression fit $\hat{f}(\cdot)$ at an input point $\mathbf{x} = \mathbf{x}_0$, using squared-error loss:

$$\begin{aligned}
\text{Err}(\mathbf{x}_0) &= E\left[\left(y - \hat{f}(\mathbf{x}_0)\right)^2 | \mathbf{x} = \mathbf{x}_0\right] \\
&= E\left[\left(\underbrace{(y - f(\mathbf{x}_0))}_{a} + \underbrace{\left(f(\mathbf{x}_0) - E[\hat{f}(\mathbf{x}_0)]\right)}_{b} + \underbrace{\left(E[\hat{f}(\mathbf{x}_0)] - \hat{f}(\mathbf{x}_0)\right)}_{c}\right)^2\right] \\
&= E\left[(y - f(\mathbf{x}_0))^2\right] + E\left[\left(f(\mathbf{x}_0) - E[\hat{f}(\mathbf{x}_0)]\right)^2\right] + E\left[\left(E[\hat{f}(\mathbf{x}_0)] - \hat{f}(\mathbf{x}_0)\right)^2\right] \\
&= \sigma_\epsilon^2 + (\text{bias})^2 + \text{variance}
\end{aligned} \tag{7.1}$$

$\hat{f}(\mathbf{x}_0)$ is not a single value but looks like a random variable (y is also a random variable). The reason is that we consider many fitted models trained on different data set $\mathcal{D}$'s, and use $E[\hat{f}(\mathbf{x}_0)]$ to simplify the notation $E_{\mathcal{D}}[\hat{f}(\mathbf{x}_0|\mathcal{D})]$. $(\text{bias})^2$ is the amount by which the average of our estimate differs from the true mean; variance is the instability of the estimations when taking different training sets; noise is the variance of the target around its true mean, and cannot be avoided no matter how well we estimate $f(\mathbf{x})$, unless $\sigma_\epsilon^2 = 0$.

(7.1) omits the cross products because they are all zeros. One example is that

$$\begin{aligned} E\left[(y - f(\mathbf{x}_0))\left(E[\hat{f}(\mathbf{x}_0)] - \hat{f}(\mathbf{x}_0)\right)\right] &= E\left[\epsilon\left(E[\hat{f}(\mathbf{x}_0)] - \hat{f}(\mathbf{x}_0)\right)\right] \\ &= E[\epsilon]E[\hat{f}(\mathbf{x}_0)] - E[\epsilon\hat{f}(\mathbf{x}_0)] \\ &= 0, \qquad \text{(observe that } \epsilon \text{ and } \hat{f}(\mathbf{x}_0) \text{ are independent)} \end{aligned}$$

Figure 7.1 illustrates the bias-variance decomposition heuristically, and their relative positions are the training errors. The blue-shaded region indicates the error σ_ϵ centered at the truth of all training samples, and one realization (a set of realizations) is shown. The model space is the set of all linear predictions from p inputs (training samples), larger than the restricted model space that regularizes the coefficients. The model space has a better closest fit in population (average performance on different training sets); however, its closest fit to one specific set is far from the closest fit in population, which shows lower bias but higher variance.

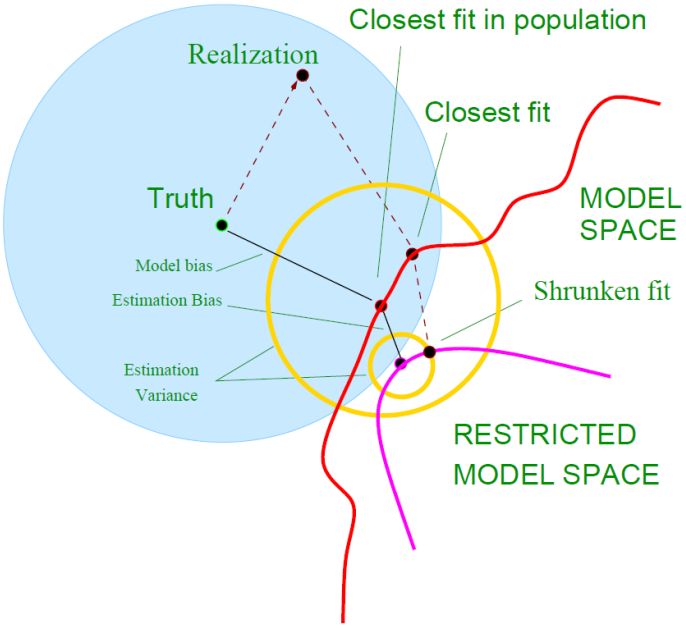

Figure 7.1: Illustration of bias-variance decomposition. [ESLII]

7.2 *Optimism*

Giving a training set

$$\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\},$$

and new observations are denoted as (X^0, Y^0).

Here are some important concepts:

1. Test error / generalization error: a fixed training set $\mathcal{T} \to$ a random test sample (X^0, Y^0)
$$\text{Err}_\mathcal{T} = \text{E}_{X^0,Y^0}\left[L(Y^0, \hat{f}(X^0))|\mathcal{T}\right]$$
2. Expected test (prediction) error: a random training set $\mathcal{T} \to$ a random test sample (X^0, Y^0)
$$\text{Err} = \text{E}_\mathcal{T}\text{E}_{X^0,Y^0}\left[L(Y^0, \hat{f}(X^0))|\mathcal{T}\right]$$

3. Training error: sample $\mathbf{x} \to$ sample $\mathbf{y}$

$$\overline{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(\mathbf{x}_i))$$

4. In-sample error: sample $\mathbf{x} \to$ a set of new Y^0 [2]

$$\text{Err}_{\text{in}} = \frac{1}{N}\sum_{i=1}^{N} \text{E}_{Y^0}\left[L(Y_i^0, \hat{f}(\mathbf{x}_i))|\mathcal{T}\right]$$

[2] It is also a kind of prediction error.

To evaluate the future performance of a model, we are interested in the test error. But for comparison between models, the in-sample error is more convenient and often leads to effective model selection. The reason is that the relative (rather than absolute) size of the error is what matters.

However, in practice, we can only calculate training error easily that is usually biased downward as an estimate of in-sample error. The difference between in-sample error and training error is called *optimism*.

1. Optimism: the difference between Err_{in} and the training error $\overline{\text{err}}$; for a fixed set of new $\mathbf{Y}^0$

$$\text{op} \equiv \text{Err}_{\text{in}} - \overline{\text{err}}$$

2. Average optimism: a random set of new $\mathbf{Y}^0$

$$\omega \equiv \text{E}_{\mathbf{y}}(\text{op})$$

Theorem 1. *For squared error, 0-1, and other loss functions, one can show quite generally that*

$$\omega = \frac{2}{N}\sum_{i=1}^{N} Cov(\hat{y}_i, y_i).$$

The harder we fit the data, the greater $Cov(\hat{y}_i, y_i)$ will be, thereby increasing the optimism. So the message is don't work too hard.

If $\hat{y}_i$ is obtained by a linear fit with d inputs or basis functions, for example the additive error model $y = f(\mathbf{x}) + \epsilon$, we have

$$\begin{aligned} \text{E}_{\mathbf{y}}(\text{Err}_{\text{in}}) &= \text{E}_{\mathbf{y}}(\overline{\text{err}}) + \omega \\ &= \text{E}_{\mathbf{y}}(\overline{\text{err}}) + 2 \cdot \frac{d}{N}\sigma_\epsilon^2, \end{aligned} \tag{7.2}$$

where many linear estimators get their ideas of estimating in-sample prediction error.

7.3 Estimates of In-Sample Prediction Error

The general form of the in-sample estimates is

$$\widehat{\text{Err}_{\text{in}}} = \overline{\text{err}} + \hat{\omega},$$

where $\hat{\omega}$ is an estimate of the average optimism.

- For linear estimators: C_p, AIC, BIC, EBIC, MDL, SRM, and SURE.
- More general-purpose methods: cross-validation and bootstrap methods.

7.3.1 *Mallow's C_p*

Orientated from (7.2),

$$C_p = \overline{\text{err}} + 2 \cdot \frac{d}{N} \hat{\sigma}_\varepsilon^2,$$

where $\hat{\sigma}_\varepsilon^2$ is a variance estimate of the noise.

7.3.2 *AIC*

AIC calculates Err_{in} by a **log-likelihood loss function**

$$\begin{aligned}
\text{Err}_{\text{in}} &= \frac{1}{N} \sum_{i=1}^{N} \text{E}_{Y^0} \left[L(Y_i^0, \hat{f}(x_i)) | \mathcal{T} \right] \\
&= -2\text{E} \left[\log p_{\hat{\theta}}(Y_i^0) \right] \\
&\approx -\frac{2}{N} \sum_{i=1}^{N} \log p_{\hat{\theta}}(y_i) + 2 \cdot \frac{d}{N} \\
&= -\frac{2}{N} \cdot \text{loglik} + 2 \cdot \frac{d}{N} \\
\text{AIC} &= -2 \cdot \text{loglik} + 2 \cdot d,
\end{aligned}$$

where $\text{loglik} = \sum_{i=1}^{N} \log p_{\hat{\theta}}(y_i)$. The key idea is that the log-likelihood of in-sample Y^0 is similar to the log-likelihood of training y_i that hold asymptotically as $N \to \infty$, showing in the third line.

AIC has the following features:

- It is asymptotically efficient (in minimizing the MSE) because it tends to find the best model without prior knowledge;
- It is not asymptotically consistent (in selecting p), i.e., it cannot find the optimal p;
- It is not good for small N and large d problem;
- It should be used for prediction.

7.3.3 *BIC*

BIC is defined as

$$\text{BIC} = -2 \cdot \text{loglik} + (\log N) \cdot d,$$

where $\text{loglik} = \log p(\{x_i, y_i\}_1^N | \hat{\theta}_m, \mathcal{M}_m)$, suppose we have a set of candidate models $\mathcal{M}_m, m = 1, \ldots, M$ and corresponding model parameters θ_m.

Compared with AIC, BIC is also maximization of log-likelihood. Still, it arises in the Bayesian approach, and choosing the model with minimum BIC is equivalent to choosing the model with the largest (approximate) posterior probability.

BIC replaces the factor 2 in AIC with $\log N$. When $N > e^2 \approx 7.4$, it tends to penalize complex models more heavily, giving preference to simpler models in selection.

BIC has the following features:

- It is not asymptotically efficient (in minimizing the MSE) because its objective is the posterior probability;
- It is asymptotically consistent (in selecting p);

- It tends to choose simpler models due to its heavier penalty. It has additional complexity from the prior, comparing with the frequentist analyses in AIC.
- It should be used for building descriptive models.

7.3.4 Cross-Validation

Cross-validation directly estimates the expected extra-sample error

$$\text{Err} = \text{E}\left[L(Y, \hat{f}(X))\right]$$

8
Mixture Models and EM

This chapter focuses more on the explanation of the EM instead of the mixture models, and thus I change the order of the sections.

8.1 The EM Algorithm in General

Section 9.4

Consider a probabilistic model in which we collectively denote all of the observed variables by $\mathbf{X}$ and all of the hidden variables by $\mathbf{Z}$. The joint distribution $p(\mathbf{X}|\boldsymbol{\theta})$ is governed by a set of parameters denoted $\boldsymbol{\theta}$. Our goal is to maximize the likelihood function that is given by

$$p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) \tag{8.1}$$

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \ln \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}), \tag{8.2}$$

which will apply equally well to continuous latent variables by replacing the sum with an integral.

A key observation is that even if the joint distribution $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ belongs to the exponential family, $\ln \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ is not simple because of its log-sum form. In comparison, optimization of the complete-data likelihood function $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ is significantly easier, which leads to the problem that how we can call out $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ without knowing the latent variables $\mathbf{Z}$.[1]

[1] We consider instead its expected value under the posterior distribution of the latent variable $\mathbb{E}_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta})}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})]$, which corresponds (as we shall see) to the E step of the EM algorithm. We will show how to derive this soon.

Here comes an important decomposition by introducing any distribution $q(\mathbf{Z})$ defined over the latent variables,

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q\|p), \tag{8.3}$$

where

$$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})}{q(\mathbf{Z})} \right\} = \mathbb{E}_{q(\mathbf{Z})}\left[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})\right] + \mathrm{Entropy}[q(\mathbf{Z})]$$

$$\mathrm{KL}(q\|p) = -\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \right\}.$$

Note that $\mathcal{L}(q, \boldsymbol{\theta})$ is a functional of the distribution $q(\mathbf{Z})$, and a function of the parameters $\boldsymbol{\theta}$. The decomposition (8.3) tells that no matter what $q(\mathbf{Z})$ we choose, $\mathcal{L}(q, \boldsymbol{\theta})$ is always the lower bound of $\ln p(\mathbf{X}|\boldsymbol{\theta})$ because $\mathrm{KL}(q\|p) \geq 0$.

E-step: The lower bound $\mathcal{L}(q, \boldsymbol{\theta})$ is maximized with respect to $q(\mathbf{Z})$ while holding $\boldsymbol{\theta}^{\mathrm{old}}$ fixed. The optimum will occur when the KL divergence vanishes, in other words when $q(\mathbf{Z})$ is equal to the posterior distribution $p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\mathrm{old}})$.

$$\text{E-step: } q(\mathbf{Z}) = \arg\max_{q} \mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{old}}) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{\mathrm{old}}).$$

It can be visualized as Figure 8.1, in which we want to maximize the blue curve $\mathcal{L}(q, \boldsymbol{\theta})$ so that it can tangential contact with the objective $\ln p(\mathbf{X}|\boldsymbol{\theta})$ at $\boldsymbol{\theta}^{\text{old}}$.[2] After E-step, our objective (8.3) can be simplified as:

[2] $\because \frac{\partial}{\partial \boldsymbol{\theta}^{\text{old}}} \text{KL}(q\|p) = 0$

$$\text{E-step's result: } \ln p(\mathbf{X}|\boldsymbol{\theta}) = \mathbb{E}_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{\text{old}})}\left[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})\right] + \text{Entropy}[q(\mathbf{Z})].$$

Because this step brings in the expectation of the complete-data log likelihood $\mathbb{E}_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta})}\left[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})\right]$, it gets its name E-step.

M-step: Since the entropy term is irrelevant with $\boldsymbol{\theta}$, the quantity being maximized is the expectation term:

$$\text{M-step: } \boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{\text{old}})}\left[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})\right].$$

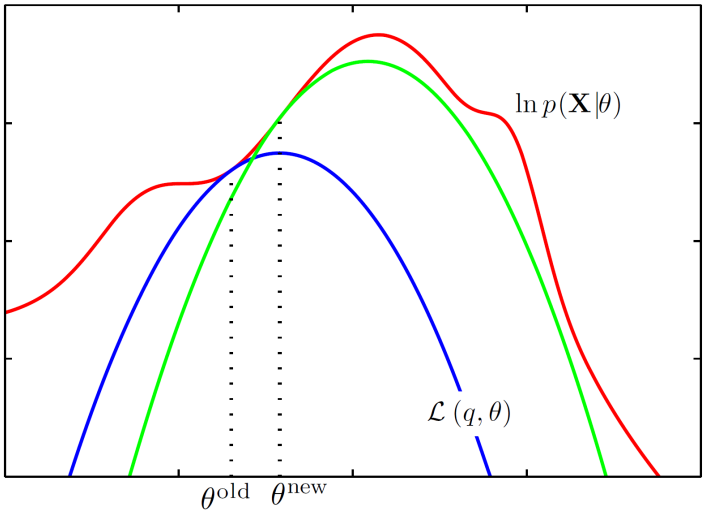

Figure 8.1: Illustion of EM algorithm

8.2 *Gaussian Mixtures Revisited*

In this section, we won't give a detailed explanation of clustering based on the Gaussian mixture model. Instead, we want to repeat the derivation of the general EM algorithm to inspire the the EM algorithm for Gaussian mixtures.

Like (8.1)(8.2), we first write the likelihood function. For a single observation $\mathbf{x}$:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$\ln p(\mathbf{x}|\boldsymbol{\theta}) = \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

where π_k is the prior probability of class k. For a data set of observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$:

$$p(\mathbf{X}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{8.4}$$

$$\ln p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{8.5}$$

This brings the same problem that maximizing (8.5) is difficult.

Now we introduce a K-dimensional binary random variable $\mathbf{z}$ having a 1-of-K representation in which a particular element z_k is equal to 1 and all other elements are equal to 0. $\mathbf{z}_n$ is the 1-of-K representation of the true label of $\mathbf{x}_n$ that we never know. We can write $p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$ and $p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$. By introducing the latent variable $\mathbf{z}$, we can write the complete-data likelihood function,

$$p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}$$

$$\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left\{ \ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}, \tag{8.6}$$

where z_{nk} denotes the k^{th} component of $\mathbf{z}_n$.

You might think we have not gained much by introducing the equivalent form, however, dealing with (8.6) instead of (8.5) is much easier.

You might also be confused by whether $\mathbf{z}_n$ is a latent variable or a parameter. Here we need to clarify what a latent variable is. A latent variable $\mathbf{z}_n$ is a random variable whose value depends on the corresponding observation $\mathbf{x}_n$. In contrast, a parameter like π_k is a constant value across different observations. $p(z_k = 1) = \pi_k$ or more specifically $p(z_{nk} = 1) = \pi_k$ are the (prior) statistics of the random variable, while the posterior probability

$$\begin{aligned} \gamma(z_{nk}) \equiv p(z_{nk} = 1|\mathbf{x}_n, \boldsymbol{\theta}) &= \frac{p(z_{nk} = 1) p(\mathbf{x}_n|z_{nk} = 1)}{\sum_{j=1}^{K} p(z_{nj} = 1) p(\mathbf{x}_n|z_{nj} = 1)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \end{aligned} \tag{8.7}$$

is a different story. Although we don't know $\mathbf{z}_n$, we have its posterior probability $p(z_{nk} = 1|\mathbf{x}_n, \boldsymbol{\theta})$.

E-step: (8.7) is what we need.

M-step:

$$\begin{aligned} \mathbb{E}_{\mathbf{Z}} \left[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) \right] &= \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) z_{nk} \left\{ \ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left\{ \ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}. \end{aligned}$$

9
Tree-based model

This section is developed based on Chapter 8: Tree-Based Methods of ISLR, and the notes by Prof. Jing.

9.1 Regression Tree

What is a regression tree:

1. Partition the feature space R into a set of J high-dimensional rectangles $R_1, \ldots, R_J$.
2. For every observation that falls into the region R_j, its prediction is simply the mean of the response values for the training observations in R_j.

Pros:

- Easy to implement
- Often easy to interpret

Cons:

- Rectangles: too restrictive
- Lack of accuracy

9.1.1 Recursive Binary Splitting

The optimal goal of our regression tree is to minimize the residual sum of squares (RSS) given by

$$\sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \bar{y}_{R_m})^2 = \sum_{m=1}^{|T|} \operatorname{var}(R_m),$$

where $|T| = J$ indicates the number of terminal nodes of the tree T.

However, it is computationally expensive to consider all possible partitions. *Recursive binary splitting* is a greedy approach that only focuses on finding the largest reduction in RSS at the current step. It has the following steps:

1. It slides along each subspace dimension to look for the best cutpoint that can minimize the RSS within each of the resulting regions.
2. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

9.1.2 *Tree Pruning*

A good tree should be:

- Low bias: as simple as possible, but not simpler;
- Low variance: as homogeneous (leaf nodes) as possible,

which inspires that a smaller tree with fewer splits may be better. There are two methods to prune a tree: forward search and backward search.

Forward search: Before splitting, make sure its RSS reduction exceeds a high threshold. The problem is that it is short-sighted because some ignored splits may have big gain later on.

Backward search: Grow a very large tree, and then prune it to obtain a subtree. Within this method, there are two approaches:

1. Test error rate: Validate each subtree and select the one with the lowest test error rate. This method is computationally expensive.
2. Model complexity: *Cost complexity pruning*, also known as *weakest link pruning*: Its objective is
$$T^* = \arg\min_T \sum_{m=1}^{|T|} \text{var}(R_m) + \alpha|T|,$$
which is similar to the lasso. α is a hyperparameter to control a trade-off between the subtree's complexity and its fit to the training data, which can be chosen by K-fold cross-validation.

9.2 *Classification Tree*

The first difference is that RSS cannot be used as the criterion. Consider a binary classification problem, in each region, the probabilities (# appearance) of the two classes are p_1 and p_2. For each region, we can use one of the three main measures:

1. Entropy: $p_1 \log_2 p_1^{-1} + p_2 \log_2 p_2^{-1}$.
2. Gini impurity index: $1 - p_1^2 - p_2^2$. Its intuition is that if we randomly pick two values from the region , the probability of having different classes.
3. Misclassification error: $1 - \max(p_1, p_2)$.

The total measure of all the regions is the weighted sum of each region's measure. The corresponding weight is $N(R_m)/N$. The recursive binary splitting requires that each split achieves the largest reduction in the total measure.

The three measures have different usages in practice. Entropy and Gini index are used in training (growing a tree), while misclassification error is used in testing and pruning.

10 Ensemble learning

This section is developed based on Chapter 8: Ensemble learning of Watermelon book, Chapter 8: Tree-Based Methods of ISLR, Chapter 10: Boosting and Additive Tree, Section 7.11 Bootstrap Methods of ESLII, and the notes by Prof. Jing.

Figure 10.1 shows two types of ensemble learning: sequential learning and parallel learning. Sequential learning is based on boosting; AdaBoost method will be introduced. Parallel learning is based on bootstrapping; Bagging and Random Forest will be introduced.

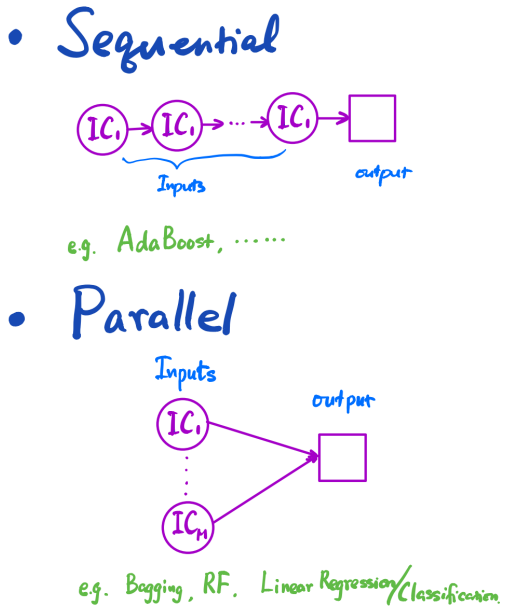

Figure 10.1: Two types of ensemble learning. [SML]

Boosting focuses more on lowing bias and thus can ensemble from individual learners with poor generalization ability. Bagging and Random Forest focuses more on lowing variance, and therefore it's efficient in learners sensitive to sample selection, such as decision trees without pruning and neural networks.

10.1 Boosting

Boosting is an idea of combining the outputs of many "weak" classifiers to produce a powerful "committee". It is also called residual learning. It has the following models:

- AdaBoost

- GBM/GBR
- Extensions:
  - SGB
  - XGBoost
  - lightGBM

Here we will introduce the most popular AdaBoost model.

10.1.1 *AdaBoost*[1]

[1] Chapter 10 Boosting Methods, ESLII

Consider a binary classifier. AdaBoost can be viewed as an addictive model:

$$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right],$$

where $G_m(x) \in \{-1, 1\}$ is the individual classifier; α_m is a weight of the contribution of $G_m(x)$, i.e., the quality of the classifier.

The intuition of AdaBoost is that at step m, those observations that were misclassified by the classifier $G_{m-1}(x)$ induced at the previous step have their weights increased. In contrast, the weights are decreased for those that were classified correctly.[2] Therefore, the successive classifier focuses more on the difficult observations, which is why it is called residual learning or boosting. Based on the error rates, we can also evaluate the quality of each classifier.[3]

[2] Algorithm 1 (b)(d)

[3] Algorithm 1 (c)

Algorithm 1 shows the details of the AdaBoost.M1 algorithm. Figure 10.2 visualize an AdaBoost process which shows how observations are reweighed.

Algorithm 1: AdaBoost.M1

1: Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$
2: **for** $m = 1, \ldots, M$ **do**
3: (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
4: (b) Compute $\text{err}_m = \sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i)) / \sum_{i=1}^{N} w_i$.
5: (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
6: (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
7: **end for**
8: Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

AdaBoost is robust to overfitting. When the number of weak learners[4] increases, the bias converges exponentially fast while the variance increases by geometrically diminishing magnitudes, showing a much slower overfitting speed than most of the other methods. However, AdaBoost is sensitive to noise, and thus the regularized forms (RegBoost, AdaBoostReg, LPBoost, QPBoost) are preferable in this case.

[4] e.g. a "stump", a two terminal- node tree.

Application: Boosting for Trees. Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly. Boosting doesn't involve bootstrap sampling; instead each tree fits on a modified version of the original data set and grown sequentially.

Section 8.2.3 Boosting, ISLR

10.2 *Bootstrap*

Bootstrap is a resampling method to create multiple subsets of a data set. Given a data set of N samples, each subset is sampled from the data set without taking

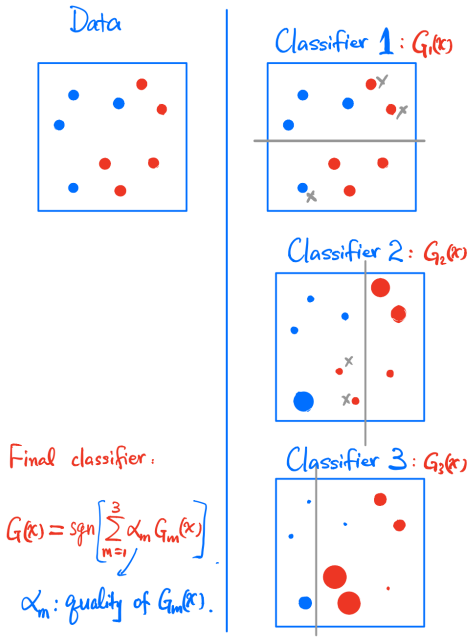

Figure 10.2: Visualization of an AdaBoost process.

the samples out, i.e., it enables replicate sampling. Figure 10.3 shows the bootstrap process.

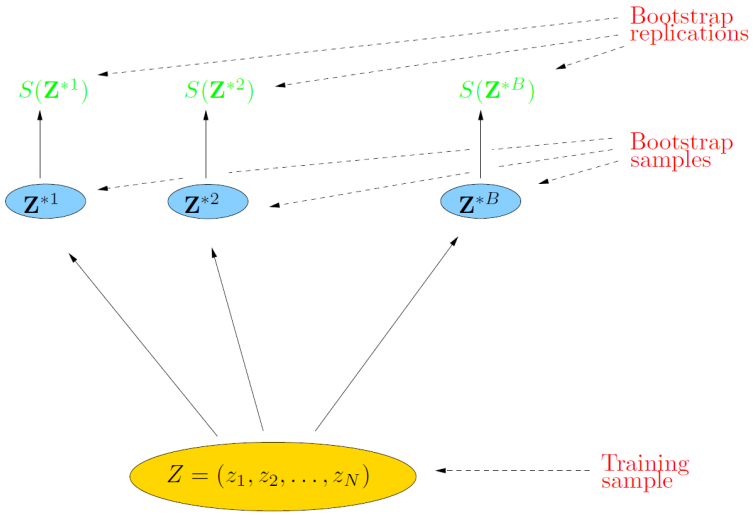

Figure 10.3: Schematic of the bootstrap process. [ESLII]

The probability that an observation is in a bootstrap subset is roughly

$$\begin{aligned} p(\text{observation } i \in \text{bootstrap sample } b) &= 1 - \left(1 - \frac{1}{N}\right)^N \\ &\approx 1 - e^{-1} \\ &= 0.632, \end{aligned}$$

where N is the number of samples in each subset. We will use the whole subset for training, and the observations not in the subset, about 36.8% of the data set, for testing. The result is called *out-of-bag estimate*.

The bootstrap is very useful when the data set is small since it is tough to separate the data set into a training and a test set. Besides, the bootstrap benefits ensemble

learning by creating multiple subsets.

10.2.1 *Bagging*

Bagging stands for Bootstrap aggregation, and it is based on bootstrap sampling. It is a general-purpose procedure for reducing the variance of a statistical learning method because averaging a set of observations reduces variance. Its idea is that although we do not have access to multiple training sets, we can generate bootstrap samples, train our method on these subsets, and average (vote) all the predictions.

Application: Bagging for Trees. These trees are grown deep and are not pruned. Hence each tree has high variance but low bias. Averaging these trees reduces the variance.

Section 8.2.1 Boosting, ISLR

10.2.2 *Random Forest*

Random forests is an improved version of bagged trees by a tweak to decorrelate the trees. One problem of bagged trees is that when strong predictors are in the data set, they will be used as top splits in most bagged trees, leading to highly correlated learners. In this case, ensemble learning cannot reduce the variance significantly.

Random forests overcome this problem by forcing only a small subset k of the d predictors can be considered in each split. A recommended choice is $k = \log_2 d$. Compared with the bagging, random forests further increase the diversity by the augmentation of parameter selection.

11
Bibliography

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A note on the group lasso and a sparse group lasso. *preprint*, 2010.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.