*Chapter 4*

# *BASIC PRACTICE AND MODEL ASSESSMENT IN MACHINE LEARNING*

**CONTENTS**

## 4.1 Feature Engineering

The problem of transforming raw data into a usable dataset, which contains clear and concretely describable components of data points, is called feature engineering. For most practical problems, feature engineering is a labor-intensive process that demands from the data analyst a lot of creativity and, preferably, domain knowledge, and expert advice.

**Example 4.1.1.** In IT industry, to transform the logs of users' interaction with a computer system, one could create features that contain information about the user and various statistics extracted from the logs. The features maybe:

1. price of the subscription; and
2. frequency of connections daily, weekly, or yearly; and
3. average session duration in seconds or the average response time for one request; and
4. the positions and movements of the mouse cursor on the screen.

The role of the data analyst is to create informative features, which improve the prediction power of the model to be built. Highly informative features are also called features with high predictive power.

### 4.1.1 One-hot Encoding (a single high (1) bit, hot, with all else are low (0))

Some learning algorithms only work with numerical feature vectors, but some features in the dataset are categorical by nature. We can transform the categorical feature into several binary ones.

**Example 4.1.2.** Consider a categorical feature "colors" with three possible values: "red", "yellow", and "green". We can transform this feature into a three new numerical features named:

"red", "yellow", and "green" .

1. if "colors" = red, then feature "red" = 1, feature "yellow" = 0, and feature "green" = 0; and
2. if "colors" = yellow, then feature "red" = 0, feature "yellow" = 1, and feature "green" = 0; and
3. if "colors" = green, then feature "red" = 0, feature "yellow" = 0, and feature "green" = 1.

**Remark 4.1.1.** In Example 4.1.2, we should not transform the "color" feature into a single feature labeled with identity number, *i.e.* red into 1, yellow into 2, and green into 3. Because that would imply there is an order among the values in this category, any this kind of specific order is important for decision making, and this will mislead them that results in overfitting. However, in dealing with ordinal categorical features, one can transform the ordinal categorical feature into a single feature labeled with several ordinal values to emphasize the importance of the order.

### 4.1.2 Binning

As opposed to one-hot encoding, is when you have a numerical feature but the real precise value of the feature does not matter, one may convert it into a categorical feature. **Binning** (or bucketing) is the process of converting a continuous feature into multiple binary features, called bins or buckets, typically depending on value range. Common binning methods are splitting by equal intervals of the value and splitting by quantile of the dataset. Minimum Description Length Principle is another binning method which based on information entropy (See Subsection **??**).

**Example 4.1.3.** In questionnaire, instead of representing age as a single real-valued feature, the analyst could chop ranges of age into discrete bins. For example, one could set:

1. **Bin 1**: all aged between 0 and 17 years-old; and
2. **Bin 2**: all aged between 18 and 64 years-old; and
3. **Bin 3**: all aged 65 years-old or above.

Assume that feature $d = 4$ represents age. By applying binning, we replace this feature with the corresponding bins. Let the three new bins be:

"age_bin1", "age_bin2", and "age_bin3",

added with indexes $d = 123$, $d = 124$, and $d = 125$, respectively[1]. For example,

1. If $x_n^{(4)} = 7$, then we set $x_n^{(123)} = 1$, $x_n^{(124)} = 0$, and $x_n^{(125)} = 0$.
2. If $x_n^{(4)} = 54$, then we set $x_n^{(123)} = 0$, $x_n^{(124)} = 1$, and $x_n^{(125)} = 0$.

In actual implementation, we shall drop both feature 4 and feature 123 to avoid the problem of multicollinearity as $x_n^{(123)} + x_n^{(124)} + x_n^{(125)} = 1$.

**Remark 4.1.2. Multicollinearity Problem**: $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(D)})$ is said to be multicollinear if some of its feature variables $\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(D)}$ are linearly dependent, *i.e.* there are high linear correlations among the feature variables. For instance, in Example 4.1.3, clearly we have $x_n^{(123)} + x_n^{(124)} + x_n^{(125)} = 1$, so if $x_n^{(124)} = 0$ and $x_n^{(125)} = 1$, we immediately know that $x_n^{(123)} = 1 - x_n^{(124)} - x_n^{(125)} = 0$. Linear regression models suffer from multicollinearity, because the least squares estimator $\hat{\boldsymbol{\omega}}_{\text{LSE}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}$ relies heavily on inverting the matrix $\boldsymbol{X}^\top \boldsymbol{X}$. Especially if $\boldsymbol{X}^\top \boldsymbol{X}$ is closed to ill-posed, *i.e.* with a very small $|\det(\boldsymbol{X}^\top \boldsymbol{X})|$,

[1]The large numbering of features is to ensure that the newly created features may not mix up with previously assigned features.

the variance of the estimated least square coefficients

$$\mathrm{Var}(\hat{\boldsymbol{\omega}}_{\mathrm{LSE}}) = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\mathrm{Var}(\boldsymbol{Y})\{(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\}^\top = \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$$

will be very large, indicating that the estimated least square coefficients are unstable. Let say if $\boldsymbol{x}^{(d)}$ is the cause of near-multicollinearity, a common practice is to remove this feature variable, but this will suffer from information loss. To remedy this shortcoming, one approach is to use Pinciple Component Analysis (PCA) to remove the linearity among feature variables, and then apply Principle Component Regression (PCR), which leads to Section 5.3.

### 4.1.3 Normalization and Standardization

#### 4.1.3.1 Normalization

Normalization is a process of converting an actual range of values taken by a numerical feature, into a standard range of values, typically in the interval $[-1, 1]$ or $[0, 1]$. The normalization formula for a $d$-th feature variable is:

$$\tilde{\boldsymbol{x}}^{(d)} = \frac{\boldsymbol{x}^{(d)} - \min\limits_{n=1,\cdots,N}\{x_n^{(d)}\}}{\max\limits_{n=1,\cdots,N}\{x_n^{(d)}\} - \min\limits_{n=1,\cdots,N}\{x_n^{(d)}\}}\,.$$

Normalizing the data is not necessary, but it is good to have normalization because:

1. **Avoid Number Overflow**: Ensure the inputs are roughly in the same relatively small range to avoid problems computers calculation working with very small or very large numbers.

2. **Increase Speed of Learning**: Imagine the model has a two-dimensional feature vector $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})$ with mean squared error (MSE) as a cost function, for simplicity, we omit the intercept term $b$, *i.e.*

$$\mathrm{MSE} = \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \omega_1 x_n^{(1)} - \omega_2 x_n^{(2)}\right)^2 \propto (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega}) =: L\,.$$

The gradient with respect to $\boldsymbol{\omega}$ is:

$$\nabla_{\boldsymbol{\omega}} L = -2\boldsymbol{X}^\top\boldsymbol{y} + 2\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\omega} = -2(\boldsymbol{X}^\top\boldsymbol{y} - \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\omega})\,.$$

From big O and small o notations in Appendix 4.A.2, consider the dataset containing only $N = 2$ samples, $x_n^{(1)} = O(1)$ and $x_n^{(2)} = o(1)$, for $n = 1, 2$, *i.e.* $x_n^{(1)}$ is likely not large and $x_n^{(2)}$ is likely small, we also assume that $y_1$ and $y_2$ are of similar magnitude. Then,

$$-\frac{1}{2}\nabla_{\boldsymbol{\omega}} L = \boldsymbol{X}^\top\boldsymbol{y} - \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\omega} = \begin{pmatrix} O(1) & O(1) \\ o(1) & o(1) \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} - \begin{pmatrix} O(1) & O(1) \\ o(1) & o(1) \end{pmatrix}\begin{pmatrix} O(1) & o(1) \\ O(1) & o(1) \end{pmatrix}\begin{pmatrix} \omega_1 \\ \omega_2 \end{pmatrix}$$

$$= \begin{pmatrix} O(1) \\ o(1) \end{pmatrix} - \begin{pmatrix} O(1) & o(1) \\ o(1) & o(1) \end{pmatrix}\begin{pmatrix} \omega_1 \\ \omega_2 \end{pmatrix} = \begin{pmatrix} O(1) - O(1) - o(1) \\ o(1) - o(1) - o(1) \end{pmatrix} = \begin{pmatrix} O(1) \\ o(1) \end{pmatrix},$$

where we suppose in advance that $\omega$ and $\omega_2$ are bounded values, and so the partial derivative with respect to $\omega_1$ dominates the update. Most converging numerical schemes rely on iterative refinement on the parameters, a typical example is the Gauss-Newton algorithm (See Subsection 6.1.2). For the present scheme, the modification in each iteration is given by $\Delta\omega_d := \eta \cdot \frac{\partial L}{\partial \omega_d}$, where $\eta$ is a hyperparameter learning rate (See Subsection 6.2.1), for the present $\omega_d$, $d = 1, 2$, so the change $\Delta\omega_1$ for $\omega_1$ is of relatively much larger order than that $\Delta\omega_2$ for $\omega_2$. $L$ is very sensitive to changes in $\omega_1$ than

that of $\omega_2$. Therefore, we need an even smaller learning rate $\eta$, and so more iterations are required for $\omega_2$ to approach its optimal value, see Figure 4.1.1 for an illustration. The blue line represents the update where $\Delta\omega_1^{(t)}$ has relatively much larger order than $\Delta\omega_2^{(t)}$ for each iteration $t$, and it takes 5 iterations for the algorithm to arrive at the optimal solution; while the red line represents the update where a smaller learning rate is used such that 4 more steps are required to reach the optimal solution.

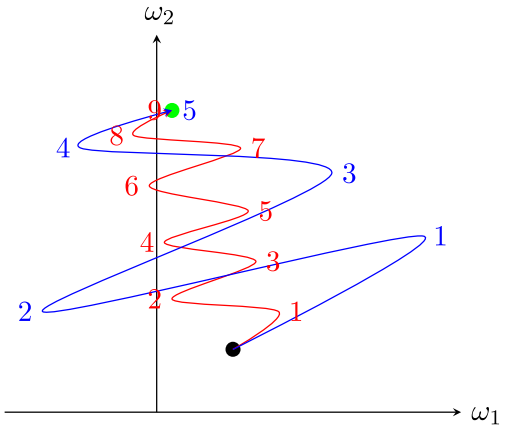



Figure 4.1.1: Parameter updates.

#### 4.1.3.2 Standardization

Standardization (or $z$-score normalization) is the procedure during which the feature values are rescaled so that they share the properties similar to a standard normal distribution with $\mu = 0$ and $\sigma = 1$. The standardization formula on feature $d$ is given by;

$$\tilde{\boldsymbol{x}}^{(d)} = \frac{\boldsymbol{x}^{(d)} - \mu^{(d)}}{\sigma^{(d)}}, \qquad \text{where} \quad \mu^{(d)} = \frac{1}{N}\sum_{n=1}^{N} x_n^{(d)} \quad \text{and} \quad \sigma = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}\left(x_n^{(d)} - \mu^{(d)}\right)^2}. \tag{4.1.1}$$

The reason for having a standardization is the same as that for normalization. Usually, if your dataset is not too big and you have time, you can try both and see which one performs better for your task. Yet as a rule of thumb, standardization is more preferable than normalization when:

1. using unsupervised learning algorithms; and
2. if the distribution of the feature is close to a normal distribution; and
3. if the feature contains extreme values (outliers): Normalization will "squeeze" the normal values into a relatively smaller range.

in all other cases, normalization is more preferable.

### 4.1.4 Dealing with Missing Features

In some datasets, entry values of some features variables can be missing. The typical approaches of dealing with missing values for a feature include:

1. Remove the examples with missing features from the dataset if your dataset is big enough so you can sacrifice some training examples, especially if the missing features is missing completely at random

(MCAR).

2. Use a learning algorithm that can deal with missing feature values, for example, C4.5 in classification tree method in Subsection **??**.

3. Use data imputation technique. Assume that the missing feature is the $d$-th feature and the missing value exists in the $\ell^{th}$ entry, *i.e.* $x_\ell^{(d)}$ is missing. The data imputation techniques are:

   (a) Replace the missing value $x_\ell^{(d)}$ of a feature $d$ by an average value of this feature in the dataset:
   $$\hat{x}_\ell^{(d)} \leftarrow \frac{1}{N}\sum_{n=1}^{N} x_n^{(d)} .$$

   (b) Replace the missing value by a value in the middle of the range, if not the median,
   **Example**: If $\boldsymbol{x}^{(d)} \in [0,1]$, then you set the missing value $x_\ell^{(d)} = 0.5$.
   **Intuition**: The value in the middle of the range will often not significantly affect the prediction, as the data points many symmetrically distributed around the mid-point.

   (c) Use the missing feature $\boldsymbol{x}^{(d)}$ as the target variable and regress against the remaining features $(\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(d-1)}, \boldsymbol{x}^{(d+1)}, \cdots, \boldsymbol{x}^{(D)})$. The model becomes:
   $$x_n^{(d)} = \omega_0 + \omega_1 x_n^{(1)} + \cdots + \omega_{d-1} x_n^{(d-1)} + \omega_{d+1} x_n^{(d+1)} + \cdots + \omega_D x_n^{(D)} ,$$
   where the training data here is the subset of feature vectors $\{\boldsymbol{x}_n\}_{n=1,n\neq\ell}^{N}$ which contains no missing values in the missing feature $d$, *i.e.* $\tilde{\boldsymbol{X}}_{(-\ell)} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{\ell-1}, \boldsymbol{x}_{\ell+1}, \cdots, \boldsymbol{x}_N)$. After the model is built, *i.e.* after the optimal solution $\boldsymbol{\omega}^*$ is found, we set the missing value by:
   $$x_\ell^{(d)} \leftarrow \omega_1^* x_\ell^{(1)} + \cdots + \omega_{d-1}^* x_\ell^{(d-1)} + \omega_{d+1}^* x_\ell^{(d+1)} + \cdots + \omega_D^* x_\ell^{(D)} .$$

   (d) (**Warning**) This approach is applicable only if you have a significantly large dataset with just a few features with missing values.
   Replace the missing value by any number, mid-point of just some common reference point, and add an additional binary feature for each missing feature, *i.e.*
   $$x_n^{(D+1)} = \begin{cases} 1, & \text{if } x_n^{(d)} \text{ is present} \\ 0, & \text{if } x_n^{(d)} \text{ is missing} \end{cases}, \qquad n = 1, \cdots, N .$$
   **Example**: If feature $d$ is the missing feature and the $\ell^{th}$ entry in feature $d$ is missing, then
   $$x_\ell^{(d)} = 0, \qquad x_1^{(D+1)} = \cdots = x_{\ell-1}^{(D+1)} = x_{\ell+1}^{(D+1)} = \cdots = x_N^{(D+1)} = 1, \qquad \text{and} \qquad x_\ell^{(D+1)} = 0 .$$

**Remark 4.1.3.** In dealing with Missing Not At Random (MNAR), the above techniques will fail. It becomes very challenging that requires us a good understanding of the data generating process. Unfortunately, the technique in dealing with MNAR will not be discussed in this book. For instance, an example of MNAR is that it is offensive to ask a lady for her age, we always can find some missing entries on the question asking their age. This absense of information comes from a structual manner, but not from randomness.

**Remark 4.1.4.**

1. **Be Consistent**: Use the same data imputation principle to fill the missing features throughout the whole dataset.

2. **Try All Combinations**: Try several data imputation techniques with several models, compare and select the best one.

## 4.2 Three Sets of a Dataset

In assessing the prediction power of a machine learning algorithm to future examples, we test the model with a test set. However, there are a lot of different models and they commonly have some hyperparameters, this is where the notion of validation set comes in. In practice, the dataset $\mathcal{S}$ is shuffled and then split into three sets:

1. **Training Set**: Build up the model.
2. **Validation Set**:
   (a) Choose the best learning algorithm; and
   (b) find the best values for hyperparameters.

   To this ends, we train a great number of different models with various hyperparameters in (1), and then use validation set to find among these trained models, the one with the smallest possible error.
3. **Test Set**: Test the precision of the selected model.

For a small dataset, *e.g.* a dataset with 10 thousand observations, the rule of thumb is to keep 70% for training, 15% for validation, and 15% for testing. However, in the era of big data, datasets often have millions of examples, it is still reasonable to keep a larger proportion, *e.g.* 90%, for training such that 5% is used for validation and another 5% is used for testing.

In Subsection 4.1.3, we have explained the need of scaling the continuous feature variables before modelling. In practice, the model is only built from the training set, and we shall never leak any information on the validation set and the test set to the model building process. Therefore, we shall first scale the training set and obtain the corresponding scaling parameters. We then use these scaling parameters to scale the validation set and the test set.

## 4.3 Overfitting and Underfitting

In the linear regression model, one might suggest to use a more complex model such as the polynomial regression model. However, introducing the higher degrees might encounter the problem of overfitting, also see Figure 4.3.1 for an illustration.

**Definition 4.3.1.**

1. **Overfitting**: The model predicts the labels of the in-sample (training) data very well but frequently makes errors when applied to the out-sample (test) data.
2. **Underfitting**: The model fails to generalize both the in-sample data and the out-sample data.

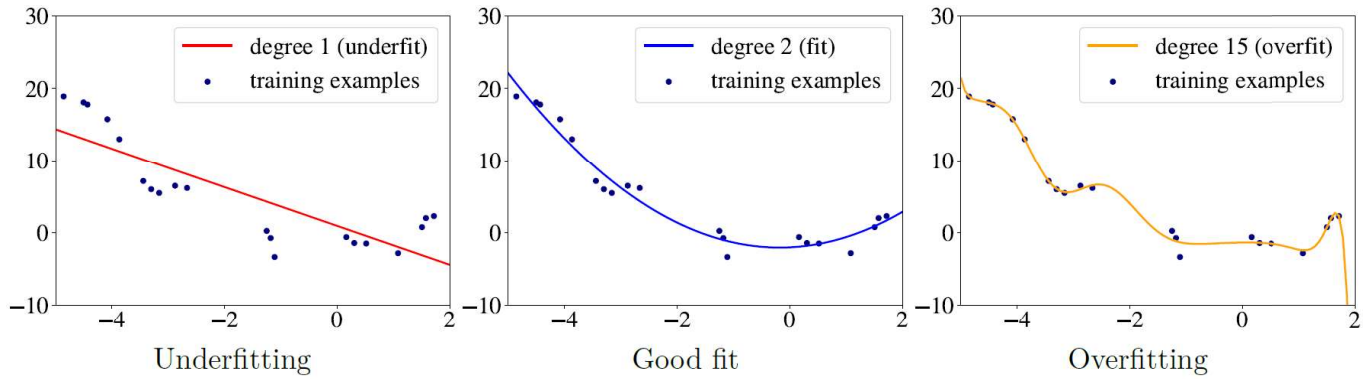



Figure 4.3.1: Examples of underfitting (linear model), good fit (quadratic model), and overfitting (polynomial of degree 15).

From Figure 4.3.1, we observed that:

1. **Underfitting**:
   (a) **Low Variability**: The curve seldom varies.
   (b) **High Bias**: The curve fails to generalize the in-sample data.
2. **Overfitting**:
   (a) **High Variability**: The curve varies a lot.
   (b) **Low Bias**: The curve fits the in-sample data pretty well.

### 4.3.1 Reasons and Solutions to Overfitting and Underfitting

1. **Reasons for having Overfitting Problem**:
   (a) The model is too complex, too many characteristics and parameters, for the data: By trying to perfectly predict labels of all training examples, the model will also learn the idiosyncratic noises among data points in the training set; and
   (b) Too many features of input, but a small number of training examples.
2. **Reasons for having Underfitting Problem**:
   (a) The model is too simple for the data; and
   (b) The features of the input are not informative.
3. **Common Solution to Overfitting**:
   (a) Try a simpler model (linear instead of polynomial regression, or SVM with a linear kernel instead of Radio Basic Function (RBF), a neural network with fewer layers/units).
   (b) Reduce the dimension of features of examples in the dataset, by using PCA, let say.
   (c) Add more training data, if possible.
   (d) Regularize the model, by enforcing coefficients or parameters close to zero to become definite zeros.

### 4.3.2 Bias-Variance Tradeoff

In the true model (or the data generating model), suppose that the output label as follows:

$$y_0 = f_0(x) + \varepsilon_0\,,$$

where $\varepsilon_0$ follows a symmetric distribution with the mean 0 and the variance $\sigma_0^2 = \mathbb{E}(\varepsilon_0^2)$. The trained model $\hat{f}$ is used to predict the label $y_0$ in the true model. Note that

1. ($f_0(x)$ is a deterministic constant): All randomness of $y_0$ comes from $\varepsilon_0$ when $x$ is given. Hence, we have
$$\mathbb{E}_{\varepsilon_0}\left(f_0(x)\right) = f_0(x)\,.$$
2. ($\hat{f}(x)$ is a variable): $\hat{f}$ is the model learnt from the training set $\mathcal{S}$, which are assumed to contain some randomness $\varepsilon_I$ including the random nature of an arbitrarily collected training set $\mathcal{S}$, or measurement or careless input; and so the randomness of $\hat{f}$ is explained by the randomness of the collected training set $\mathcal{S}$, that means for every training set, we have one $\hat{f}$ which is different from one another. Since the trained model and the true model are fundamentally different, and they are generated by separate mechanisms, it is natural to assume the independence of $\varepsilon_I$ and $\varepsilon_0$, *i.e.* $\varepsilon_I \perp\!\!\!\perp \varepsilon_0$ by using a commonly used notation, again for a fixed $x$, $\hat{f}(x)$ is regarded as a function of $\varepsilon_I$, and so $\hat{f}(x) \perp\!\!\!\perp \varepsilon_0$ too.

Then, the expected testing error at a test feature $\tilde{x}$ is given by:

$$\begin{aligned}
&\mathbb{E}\{(y_0 - \hat{f}(\tilde{x}))^2\} = \mathbb{E}\{(f_0(\tilde{x}) + \varepsilon_0 - \hat{f}(\tilde{x}))^2\} \\
&= \mathbb{E}_{\mathcal{S}}\{(f_0(\tilde{x}) - \hat{f}(\tilde{x}))^2\} + 2\mathbb{E}\{\varepsilon_0(\underbrace{f_0(\tilde{x})}_{\text{constant}} - \underbrace{\hat{f}(\tilde{x})}_{\varepsilon_0 \perp\!\!\!\perp \hat{f}(\tilde{x})})\} + \underbrace{\mathbb{E}_{\varepsilon_0}(\varepsilon_0^2)}_{\sigma_0^2} \\
&= \mathbb{E}_{\mathcal{S}}\{(f_0(\tilde{x}) - \mathbb{E}\{\hat{f}(\tilde{x})\} + \mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\} - \hat{f}(\tilde{x}))^2\} + 0 + \sigma_0^2 \\
&= \mathbb{E}_{\mathcal{S}}\{\underbrace{(f_0(\tilde{x}) - \mathbb{E}\{\hat{f}(\tilde{x})\})^2}_{\text{constant}}\} + 2\mathbb{E}_{\mathcal{S}}\{\underbrace{(f_0(\tilde{x}) - \mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\})}_{\text{constant}}(\mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\} - \hat{f}(\tilde{x}))\} + \mathbb{E}_{\mathcal{S}}\{(\mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\} - \hat{f}(\tilde{x}))^2\} + \sigma_0^2 \\
&= (f_0(\tilde{x}) - \mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\})^2 + 2(f_0(\tilde{x}) - \mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\})\underbrace{\mathbb{E}_{\mathcal{S}}\{\mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\} - \hat{f}(\tilde{x})\}}_{0} + \mathbb{E}_{\mathcal{S}}\{(\hat{f}(\tilde{x}) - \mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\})^2\} + \sigma_0^2 \\
&= \underbrace{(f_0(\tilde{x}) - \mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\})^2}_{(\text{Bias})^2} + \underbrace{\mathbb{E}_{\mathcal{S}}\{(\hat{f}(\tilde{x}) - \mathbb{E}_{\mathcal{S}}\{\hat{f}(\tilde{x})\})^2\}}_{\text{Variance}} + \underbrace{\sigma_0^2}_{\text{irreducible error}}\,.
\end{aligned}$$

In addition to Figure 4.3.1, we now consider two extreme scenarios, so as to further investigate the meaning behind overfitting and underfitting:

1. **Bias is small but variance is large**: Consider a trained machine learner $\hat{f}$ based on an algorithm flexible enough that fits so well with any given training set $\mathcal{S}$. By definition, if $\tilde{x}$ is in the random training set, $\hat{f}(\tilde{x})$ should be very close to $f_0(\tilde{x})$; otherwise, $\tilde{x}$ is an out-sample data point, after taking the average among all random datasets not containing $\tilde{x}$, overfitting and underfitting effects balance off, and the resulting partial average is still close to $f_0$ at $\tilde{x}$. Therefore, bias tends to be small, $\hat{f}(\tilde{x})$ on the average it gives a very close identification with the true model $f_0(\tilde{x})$ in predicting the label $y_0$ for data points from the test set; this average is taken over a great number of different independent training sets, it does not mean every time we have a correct label! On the other hand, the large variance means that for the variation of any one single predicted label, based on one dataset, is away from the mean prediction. After a great number of training attempts, the variance is so large under

the test set. For a machine learner $\hat{f}$ that overfits with the training data, any change of one single training datum, the learner $\hat{f}$ will change drastically, this in turn makes a large variance. Therefore, small bias with a high variance gives a sign of overfitting where the prediction power on the test set is weak.

2. **Bias is large but variance is small**: Consider another trained machine learner $\hat{f}_*$ based on an algorithm that fails to find any underlying patterns within the training set $\mathcal{S}$. On average, given a random training set $\tilde{x}$, $\hat{f}_*(\tilde{x})$ is far from $f_0(\tilde{x})$, meaning that $\tilde{x}$ is in general an out-sample data point. Therefore, the bias tends to be large, $\hat{f}_*(\tilde{x})$ on the average is away from the identification with the true model $f_0(\tilde{x})$ in predicting the label $y_0$ for data points from the test set; this average is taken over a great number of different independent training sets, it does not mean every time we have an incorrect label! On the other hand, the small variance means that for the variation of any one single predicted label, based on one dataset, close to the mean prediction. Even after a great number of training attempts, the variance is still small under the test set. For a machine learner $\hat{f}_*$ that underfits with the training data, any change of one single training datum, the learner $\hat{f}_*$ will only change slightly, this in turn makes a small variance. Therefore, high bias with a small variance gives a sign of underfitting where the prediction power on the test set is weak.

Therefore, the ideal case is where both the bias and variance are small, which the trained model $\hat{f}(x)$ is successful in predicting both training set and test set. However, in practice, one could hardly archieve both low bias and low variance. Therefore, it is called the **bias-variance tradeoff** and we shall further discussed the mathematics behind it in Section 5.4.

## 4.4 Model Performance Assessment

For a regression model, another **mean model**, which always predicts the average of the labels in the training set, is generally used to assess the performance of this selected regression model. The procedure of checking is:

1. If the selected regression model provides a better fit than the mean model, we compare the performances of this regression model via the MSEs of the training and the test sets, respectively.
2. If these MSEs are far different, there could be an overfitting; if they are close enough, we shall accept the model.
3. Otherwise, if the regression model performs weaker than the mean one, we select another new regression model and repeat Step 1.

For classification models, things are a bit more complicated. The most widely used metrics and tools to assess the classification models are:

1. Confusion matrix; and
2. Accuracy; and
3. Cost-sensitive accuracy; and
4. Precision / Recall (a.k.a. True Positive Rate (TPR)); and
5. $F_1$ score; and
6. Area under the ROC curve.

In the rest of this section, we shall illustrate these concepts, and for simplicity, only binary classification is used as an illustration. Multiclass classification will be indicated if necessary.

### 4.4.1 Confusion Matrix

The confusion matrix (a.k.a. error matrix, see Stehman (1997)) is a table that summarizes how successful the classification model is at predicting examples belonging to various classes. One axis of the confusion matrix is the label predicted by the model, and the other axis is the actual label. In a binary classification problem, there are two classes. For instance, in a medical diagnostic test, we call the result of the test is positive if the test predicts that the subject has a certain disease, see the confusion matrix in Table 4.4.1 as an example.

| **Test \ Truth** | **Disease** | **No Disease** | **Row Sum** |
|---|---|---|---|
| **Positive** | True Positive (TP) | False Positive (FP) | TP+FP |
| **Negative** | False Negative (FN) | True Negative (TN) | FN+TN |
| **Column Sum** | TP+FN | FP+TN | **Total Sum** |

Table 4.4.1: Binary confusion matrix.

1. **True Positive** (TP): The model correctly predicts the positive class;
2. **True Negative** (TN): The model correctly predicts the negative class;
3. **False Negative** (FN): The model incorrectly predicts the negative class;

4. **False Positive** (FP): The model incorrectly predicts the positive class.

**Example 4.4.1.** In Spam Detection Example (1.2.1), the model predicts two classes: "spam" and "not_spam". The confusion matrix is:

| **Confusion Matrix** | | **Predicted** | | **Total** |
|---|---|---|---|---|
| | | **Spam** | **Not Spam** | |
| **Actual** | **Spam** | 23 (TP) | 1 (FN) | 24 |
| | **Not Spam** | 12 (FP) | 556 (TN) | 568 |
| **Total** | | 35 | 557 | 592 |

Table 4.4.2: The confusion matrix for spam detection.

1. **True Positive** (TP): 23 spam examples correctly predicted as spam by the model;
2. **True Negative** (TN): 556 not spam examples correctly predicted as not spam by the model;
3. **False Negative** (FN): 1 spam example incorrectly predicted as not spam by the model;
4. **False Positive** (FP): 12 not spam examples incorrectly predicted as spam by the model.

The confusion matrix for multiclass classification has many rows and columns since there are different classes. It can help you to determine mistake patterns. For example, a confusion matrix could reveal that a model trained to recognize different species of animals tends to mistakenly predict "cat" instead of "panther", or "mouse" instead of "rat". In this case, you can add more labeled examples of these species ("cat", "panther", "mouse", and "rat") to help the learning algorithm to "see" the difference between them. Alternatively, you can add additional feature variables to help the learning algorithm to better distinguish between these species.

The confusion matrix is useful in calculating other performance metrics such as those below.

### 4.4.2 Precision / Recall

Usually, we put more emphasis on the positive results than the negative results. For example, in Spam Detection, see Table 4.4.2 as an example, a model is built to label "junk" emails as spam automatically; it is not so essential to find non-spam emails. Hence, precision and recall are formulated based on positive results.

**Definition 4.4.1.**

1. **Precision**: The ratio of correct positive predictions to the total number of positive predictions:
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}\,. \tag{4.4.1}$$
2. **Recall**: The ratio of correct positive predictions to the total number of positive examples:
$$\text{Recall (a.k.a. True Positive Rate, TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}\,. \tag{4.4.2}$$

Almost always, in practice, we have to choose between a high precision or a high recall, as it is usually

impossible to have both; to this end, a simple algebra gives

$$\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}} = 1 + \frac{\text{Total sum} - \text{TN}}{\text{TP}}$$

with both TN/Total sum and TP/Total sum unchanged, either Precision or Recall increases at the expense of the decline of another. We can achieve either of the two by various means:

1. by assigning a higher weighting to the examples of a specific class of more interest (the SVM algorithm accepts weightings of classes as inputs);
2. by tuning hyperparameters to maximize precision or recall based on a model on the validation set;
3. by varying the decision threshold for an algorithm that returns probabilities of a certain class of interest; for instance, if we use logistic regression or classification tree for classification problem, to increase precision (at the cost of a lower recall), we can decide that the prediction will be positive only if the probability returned by the model is higher than 0.9, a larger number than 0.5.

**Extension to Multiclass Classification Model**: Just a modification of the formulae of (4.4.1) and (4.4.2):

1. Consider a class for which you want to assess these metrics; then
2. count in all examples of the selected class as positives while the remaining examples of the remaining classes as negatives; then
3. calculate precision and recall accordingly, using (4.4.1) and (4.4.2), respectively.

### 4.4.3 Accuracy

**Definition 4.4.2. Accuracy**: The ratio of correctly classified examples to the total number of examples:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}. \tag{4.4.3}$$

Accuracy is a useful metric when reducing errors in predicting all classes are equally important. However, in spam detection, one may tolerate on false negatives more than false positives, which is when your good friend sends you an email, but the model labels it as a spam email and does not pop up in the "Inbox". Hence, one might use **cost-sensitive accuracy**.

### 4.4.4 Cost-Sensitive Accuracy

Cost-sensitive accuracy is useful in dealing with the situation in which different classes are of different level of importance. To compute a cost-sensitive accuracy:

1. Assign a cost (a positive number) to each type of mistakes: FP and FN, say $c_{\text{FP}}$ and $c_{\text{FN}}$ respectively; then
2. find the counts TP, TN, FP, FN as usual; then
3. multiply the counts FP and FN by the respective costs; then
4. calculate accuracy by using (4.4.3),

therefore,

$$\text{cost-sensitive accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + c_{\text{FP}}\text{FP} + c_{\text{FN}}\text{FN} + \text{TN}} \tag{4.4.4}$$

### 4.4.5 $F_1$ Score

The name of $F_1$ score comes from the definition of the $F$-measure with parameter $\beta \geq 0$ as

$$F_\beta = \frac{(\beta^2 + 1) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}\,.$$

The $F_1$ score can be obtained by setting $\beta = 1$, *i.e.*

$$F_\beta = \frac{(1^2 + 1) \times \text{Precision} \times \text{Recall}}{1^2 \times \text{Precision} + \text{Recall}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\,,$$

this is the harmonic mean of Precision and Recall.

### 4.4.6 Area Under ROC Curve (AUC)

In building a summary picture of the classification performance of a classifier, **Receiver Operating Characteristic** (ROC) curves use:

1. **True Positive Rate** (TPR): The ratio of correct positive predictions to the overall number of positive examples:
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \text{Recall}\,.$$
2. **False Positive Rate** (FNR): The rato of negative examples predicted incorrectly, *i.e.* to the overall number of negative examples:
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}\,.$$

ROC curves is especially useful and can only be for assessing classifiers which return some confidence score (or probability) of prediction. Generally, logistic regression, neural networks, ensemble models based on classification trees, such as random forest with varying voting threshold, can be assessed using ROC curves.

One can draw a ROC curve for a classification model as follows:

1. Discretize the range of the confidence score over $[0, 1]$, for instance, we pick up splitting end points from $I_{10} := \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$.
2. For each discrete value $t \in I_{10}$, a prediction threshold, predict the labels of examples in the dataset $\boldsymbol{x}$ using the classifier with this threshold $t$ so that whenever the prediction score falls below $t$, we assign the data point the label of negative class; otherwise a positive one. Then construct a confusion matrix and calculate TPR and FPR accordingly.
   **Example**: Assume when $f$ is the prediction score of the classifier for a data point $\boldsymbol{x}$. $t = 0.7$ is selected as the prediction threshold. Then,
$$\text{Predicted Class} = \begin{cases} \text{Positive}, & \text{if } f(\boldsymbol{x}) \geq 0.7\,; \\ \text{Negative}, & \text{if } f(\boldsymbol{x}) < 0.7\,, \end{cases}$$
3. Draw a ROC curve (approximately), TPR against FPR, that means plotting the points of (TPR, FRP), by varying the values of $t$ over $I_{10}$. Certainly, this is just an approximation, a broken curve,

for the ROC curve, as one taking more values from the unit interval $[0, 1]$, we get a small continuous curve, by then we get a full version of ROC curve.

Figure 4.4.1 illustrates four ROC curves from four different models. We further have some immediate observations:

1. If the threshold $t$ is 0, then all predictions will be positive, so TPR = FPR = 1 since both FN = TN = 0 (upper right corner).
2. If the threshold $t$ is 1, then all predictions will be negative, so TPR = FPR = 0 since both TP = FP = 0 (lower left corner).

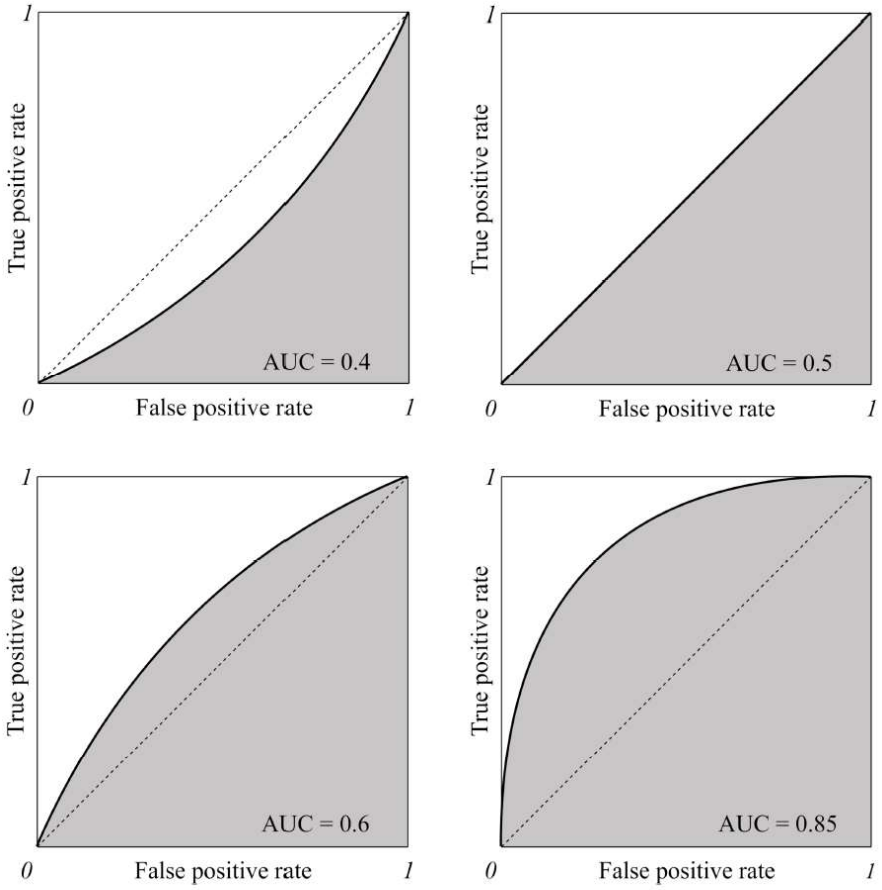



Figure 4.4.1: The area under the ROC curve (shown on gray)

Consider a random classifier that randomly classifies an observation as positive with probability $p$, and negative with probability $1 - p$. Since each positive or negative observation will be classified as positive with probability $p$, one can easily see from Table 4.4.3 that, by the law of large numbers, both ratios TPR and FPR approach to $p$ as $n_1$ (= total number of positive examples) and $n_2$ (= total number of negative examples) tend to infinity. In other words, by the Law of Large Numbers (LLN),

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}/n_1}{\text{TP}/n_1 + \text{FN}/n_1} \to p\,, \qquad \text{as} \quad n_1 \to \infty\,,$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} = \frac{\text{FP}/n_2}{\text{FP}/n_2 + \text{TN}/n_2} \to p\,, \qquad \text{as} \quad n_2 \to \infty\,,$$

indicating that (TPR, FPR) $\to (p, p)$ for all $0 < p < 1$. Thus, the ROC is a diagonal line and the area under curve (AUC) of ROC is $0.5 \times 1 \times 1 = 0.5$.

| **Test \ Truth** | **Disease** | **No Disease** | **Row Sum** |
|---|---|---|---|
| **Positive** | $\approx {}^{2}n_1 p$ | $\approx n_2 p$ | $\approx (n_1 + n_2)p$ |
| **Negative** | $\approx n_1(1-p)$ | $\approx n_2(1-p)$ | $\approx (n_1 + n_2)(1-p)$ |
| **Column Sum** | $n_1$ | $n_2$ | $n_1 + n_2$ |

Table 4.4.3: Classification table for the randomly classifying rule, the values are taken approximately assuming $n_1, n_2$ are sufficiently large.

In general, the higher the area under the ROC curve (AUC), the better the classifier. A classifier with an AUC higher than 0.5, or equivalently the ROC curve of an algorithm lies above the main diagonal of the unit square, is better than a random classifier. If AUC is lower than 0.5, very likely something is wrong with your model. A perfect classifier would have an AUC of 1. Usually, if a chosen machine learning model behaves well, you obtain a good classifier accordingly by selecting the value of the hyperparameter threshold $t$ that gives TPR close to 1 while keeping FPR near 0.

**Example 4.4.2.** Table 4.4.4 shows a training set of a diagnostic test with 20 observations $(x_n^{(1)}, x_n^{(2)})$ and the corresponding label $y_n$. 10 of the observations are positive ($y_n = 1$) and the rest are negative ($y_n = 0$). The last column shows the regressed probability $\pi_n = \mathbb{P}(y_n = 1)$ under a logistic regression. The observations are arranged in descending order with respect to $\pi_n$ in the table.

[2]The symbol $\approx$ here means that the expected value of the current cell is term on the right-hand side, after all it is a random variable, yet if $n_i$ goes to infinity, the ratio of this random variable to the corresponding $n_i$ still coverges to $p$.

| $n$ | $x_n^{(1)}$ | $x_n^{(2)}$ | $y_n$ | $\pi_n$ |
|---|---|---|---|---|
| 1 | 24.43 | 6.95 | 1 | 0.98 |
| 2 | 8.84 | 11.92 | 1 | 0.91 |
| 3 | 18.69 | -1.17 | 1 | 0.86 |
| 4 | 17.37 | -0.07 | 1 | 0.86 |
| 5 | 4.77 | 11.66 | 1 | 0.85 |
| 6 | 0.83 | 10.74 | 0 | 0.73 |
| 7 | 1.57 | 8.51 | 1 | 0.69 |
| 8 | 10.07 | -0.53 | 1 | 0.66 |
| 9 | 0.99 | 6.04 | 1 | 0.58 |
| 10 | 10.73 | -4.88 | 0 | 0.53 |
| 11 | 11.16 | -6.77 | 0 | 0.47 |
| 12 | -11.21 | 14.64 | 0 | 0.46 |
| 13 | -5.67 | 5.05 | 1 | 0.31 |
| 14 | -0.06 | -1.47 | 0 | 0.28 |
| 15 | -9.25 | 6.74 | 0 | 0.26 |
| 16 | 1.05 | -4.86 | 0 | 0.21 |
| 17 | -12.35 | 5.61 | 0 | 0.16 |
| 18 | -6.12 | -2.41 | 1 | 0.12 |
| 19 | -2.17 | -14.40 | 0 | 0.04 |
| 20 | -4.06 | -15.70 | 0 | 0.02 |

Table 4.4.4: Dataset with 20 labeled sample points from a diagnostic test and the regressed probabilities under a logistic regression

As the probability threshold $t$ varies from above 1 down to 0, we can witness the change of TPR and FPR as functions of $t$. Suppose that a decrement in $t$ of 0.05 is used:

1. ($t > 1$): All observations are classified as negative. Hence, no observation would be classified as positive and (TPR, FPR) = (0, 0). The confusion matrix becomes:

| **Test \ Truth** | **Disease** | **No Disease** | **Row Sum** |
|---|---|---|---|
| **Positive** | 0 | 0 | 0 |
| **Negative** | 10 | 10 | 20 |
| **Column Sum** | 10 | 10 | 20 |

Table 4.4.5: Confusion matrix for $t > 1$.

2. ($t > 0.95$): Only one of the 10 positive observations would be correctly classify and so TPR = 0.1. Since there are no false positive, we still have FPR = 0. Hence, (TPR, FPR) = (0.1, 0). The confusion matrix becomes:

| **Test \ Truth** | **Disease** | **No Disease** | **Row Sum** |
|---|---|---|---|
| **Positive** | 1 | 0 | 1 |
| **Negative** | 9 | 10 | 19 |
| **Column Sum** | 10 | 10 | 20 |

Table 4.4.6: Confusion matrix for $t > 0.95$.

3. The same analysis can be carried out for all other values of $t$.
4. ($t = 0$): All observations are classified as positive. Hence, no observation would be classified as negative and (TPR, FPR) = (1, 1). The confusion matrix becomes:

| **Test \ Truth** | **Disease** | **No Disease** | **Row Sum** |
|---|---|---|---|
| **Positive** | 10 | 10 | 20 |
| **Negative** | 0 | 0 | 0 |
| **Column Sum** | 10 | 10 | 20 |

Table 4.4.7: Confusion matrix for $t = 0$.

With all the 20 different values of $t$ of deciding probability. Next, we illustrate the example in **R**:

```
> y_pred <- c(0.98, 0.91, 0.86, 0.86, 0.85, 0.73, 0.69, 0.66, 0.58, 0.53,
+             0.47, 0.46, 0.31, 0.28, 0.26, 0.21, 0.16, 0.12, 0.04, 0.02)
>
> y_true <- c(1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0 , 1, 0, 0)
> threshold <- seq(from=0.05, to=0.95, by=0.05)
> TPR <- c()
> FPR <- c()
> for (t in threshold){
+     y_hat <- y_pred > t
+     conf <- table(y_true, y_hat)
+     #conf: y_hat
+     # y_true FALSE TRUE
+     #       0     TP     FN
+     #       1     FP     TN
+     TPR <- c(TPR, conf[2,2]/sum(conf[2,]))
+     FPR <- c(FPR, conf[1,2]/sum(conf[1,]))
+ }
> TPR <- c(1, TPR, 0)
> FPR <- c(1, FPR, 0)
```

Here the `seq()` generates a sequence of evenly stepped numbers, in which the arguments `from=0.5` and `to=0.95` respectively are the starting number and the ending number of the sequence, while the argument `by=0.05` indicates the step size. The confusion matrix can be generated by the `table()` function, however, one should take extreme care for the position of TP, TN, FP, and TN. Finally, the ROC is plotted with a blue line and a diagonal red line is added to represent the ROC curve for a random classifier.

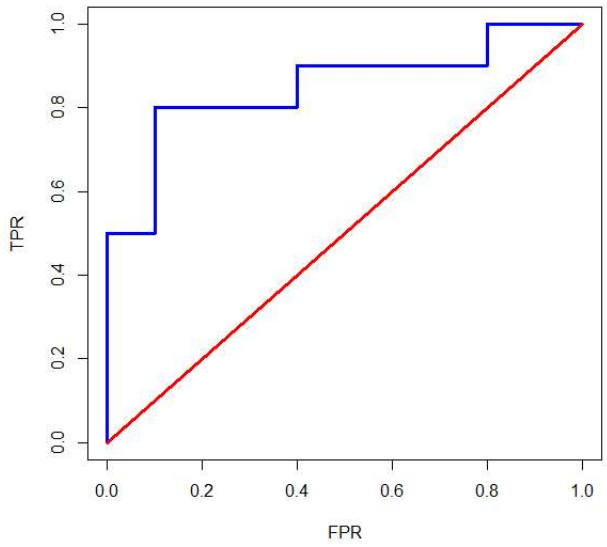



Figure 4.4.2: ROC curve based on Table 4.4.4

The ROC curve tends to be smoother as the size of dataset gets bigger and bigger.

## 4.5 Hyperparameter Tuning

As aforementioned, hyperparameters are not optimized by the learning algorithm itself through the training set. The data analyst has to "tune" hyperparameters by experimentally finding the best combination of values via the validation set.

### 4.5.1 Grid Search

Given enough data for a decent validation set, as long as the range of values taken by hyperparameters is not too large, one can consider using *grid search*. It forms a discrete list (grid) of different combinations of hyperparameters, and we aim to find the best combination by checking one by one with the validation set. Once the best combination of hyperparameters out of the finite list is found, one can try to explore the various values around this best choice. So that an even better model can be founded. Finally, further assessment on the precision of the selected model can be done with the test set.

**Example 4.5.1.** For training an SVM (Readers who have never heard about SVM before may skip this example for the momemt and revisit it after SVM is introduced in detail in Section **??**), there are commonly two hyperparameters: the penalty parameter $C > 0$ and the kernel (either "Linear" or "RBF") $K$ used by considering the objective function in a dual problem:

$$\max_{\lambda_1,\cdots,\lambda_N} C\sum_{n=1}^{N}\lambda_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{\ell=1}^{N}\lambda_n\lambda_\ell y_n y_\ell K(\boldsymbol{x}_n,\boldsymbol{x}_\ell)\,, \qquad \text{subject to} \qquad \sum_{n=1}^{N}\lambda_n y_n = 0 \quad \text{and} \quad \lambda_1,\cdots,\lambda_N \geq 0\,,$$

also (**??**) in Section **??**. Try the penalty parameter $C$ with logarithmic scaling:

$$C \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$$

Together with two kernels, the grid becomes:

| **Kernel** \ $C$ | 0.001 | 0.01 | 0.1 | 1 | 10 | 100 | 1000 |
|---|---|---|---|---|---|---|---|
| Linear (L) | (0.001, L) | (0.01, L) | (0.1, L) | (1, L) | (10, L) | (100, L) | (1000, L) |
| RBF (R) | (0.001, R) | (0.01, R) | (0.1, R) | (1, R) | (10, R) | (100, R) | (1000, R) |

Table 4.5.1: All possible combinations by grid search.

There are $7 \times 2 = 14$ pairs of hyperparameters and one may use each of the couples to build 14 SVMs in total by using the same training set. Then one can assess the performance of each SVM on the validation set by using one of the metrics introduced in Section 4.4 if for classifier, MSE for general machine learners. Finally, the model that performs the best according to the chosen metric will be selected.

Grid search is adopted if both conditions are met: (1) the dataset is small, let say less than 20,000 data points. (2) The grid is small where we have only a few combinations to try.

However, especially for large datasets and/or large number of combinations, trying all combinations of hyperparameters could be time-consuming. Thus, a more efficient method is needed.

### 4.5.2 Random Search

In *random search*, we need to provide a multivariate distribution for all combinations of hyperparameters in a feasible range; for each search attempt, a combination of hyperparameter is randomly sampled, we then assess the model with a score metric and pick the combination that has the highest score metric via a procedure similar to that with the grid search, except that we now have to specify the total number of random combinations to search. Therefore, the underlying distribution plays an important role in choosing the best combination, and its selection depends heavily on one's expertise or knowledge.

**Example 4.5.2.** We follow the same setting as in Example 4.5.1. Recall that two hyperparameters are needed in training an SVM, namely the penalty coefficient parameter $C$ and the choice of kernel ("Linear" or "RBF"). As a folklore, $C$ is often assume to follow the standard exponential distribution with mean 1 and the choice of kernel between "Linear" or "RBF" to follow a Bernoulli distribution with probability $p = 0.7$, respectively. The penalty term is more likely to take a small value while the kernel used is more likely to be RBF. Notationally,

$$C \sim \text{Exp}(1) \qquad \text{and} \qquad \text{KM} \sim \text{Ber}(0.7)\,,$$

where KM = 0 means linear kernel is used, otherwise KM = 1 means RBF kernel is adopted. To perform the random search, we generate $M$ sample combinations of $C$ and KM, from the distributions stated above. Finally, we build $M$ models based on the combinations obtained, and select the model that performs the best according to the chosen metric.

Suppose that we want at least a $p = 95\%$ probability that our selected combinations contain at least one top $1-q = 5\%$ among all possible combinations based on the chosen score metric. Here, $q = 95\%$ represents the quantile of the score metric computed based on all possible hyperparameter combinations. Then, with $M_1$ sample combinations are selected independently, the probability that none of our selected combinations are the top 5% is $q^{M_1} = 0.95^{M_1}$. With at least $p = 95\%$ probability, the probability that there is at least one selected combination is of the top 5% among all possible combinations is

$$1 - q^{M_1} \geq p \qquad \Rightarrow \qquad M_1 \geq \frac{\ln(1-p)}{\ln q} \approx 58.40\,,$$

which recommands that we should use at least $M_1 = 59$ combinations. Therefore, in some occasions where the grid size is not large, *e.g.* the grid size of Example 4.5.1 is only $7 \times 2 = 14$, which is obviously smaller than $M_1 = 59$, random search can be computationally slower than grid search. However, if the grid size is much larger than $M_1 = 59$ (as a rule of thumb suggested that in the literature, taking $M = 60$ instead of $M_1 = 59$), random search should be used.

### 4.5.3 Bayesian Hyperparameter Optimization

Instead of directly optimizing the score metric (as an objective function in hyperparameters), *Bayesian hyperparameter optimization* incorporates the hyperparameter tuning under the Bayesian framework (see Chapter **??**). Briefly speaking, the current evaluation result of a combination of hyperparameters will determine the next combination to be evaluated, each evaluation result is based on a prespecified performance

score that is generally different from the score metric assessing the model for the dataset. Bayesian hyperparameter optimization is more efficient than other hyperparameter tuning methods that need to examine a huge number of combinations of hyperparameters, because the combinations which have a lower performance score will not be evaluated. To this approach, the conditional probability distribution for a score metric given a specific combination is purposely sought, which is further used for obtaining the final performance score based on different choices of negative-loss/utility/acquisition functions. Formally, by the Bayes Theorem, this conditional probability can be understood as:

$$\mathbb{P}(\text{score metric}|\text{combination}) = \frac{\mathbb{P}(\text{combination}|\text{score metric})\mathbb{P}(\text{score metric})}{\mathbb{P}(\text{combination})}. \tag{4.5.1}$$

where $\mathbb{P}(\text{score metric})$ is the prior probability that a specific value of score metric is observed, and $\mathbb{P}(\text{combination})$ is the probability that one certain combination of hyperparameter is observed. Finally, $\mathbb{P}(\text{combination}|\text{score metric})$ is the likelihood, where the randomness comes from the dataset. For instance, referring to Example 4.5.1, given a dataset sampled randomly from the set of all possible datasets, and a combination of hyperparameters for building up one SVM model for this dataset, an accuracy score metric can be calculated for this dataset-hyperparameter pair. In this way, we can obtain a marginal probability density for the score metric at a certain level, let say 90%, whose randomness is inherited from the set of datasets, especially when each dataset carries a non-uniform "density", by summing up all probability densities for those dataset-hyperparameter pairs that attain the 90% score metric. Likewise, we can calculate the corresponding joint probability density for a specific combination of hyperparameters among those dataset-hyperparameter pairs that yield both the 90% score metric and this particular combination. The conditional probability we are looking for can be obtained exactly by dividing this joint probability density by that marginal one.

Using (4.5.1), the prior distribution is updated to the posterior distribution one, which is combined with a negative-loss/utility/acquisition function $u$ so that the corresponding maximizer serves as the performance score. However, the evaluation step for the score metric is usually very time-consuming, especially in the event of training a complicated machine learning model, due to the necessity of training the model before we can find the score metric for a combination of hyperparameters. To remedy the complexity caused by the massive amount of calculations, we can approximate the score metric through the use of *surrogate function*, more details will be discussed in Bayesian Hyperparameter Optimization in Section **??**.

### 4.5.4 Gradient-Based Hyperparameter Optimization

It computes the gradient of the objective function with respect to the hyperparameter, and then use the gradient descent method to find its optimal combination. One of which is called the *Gauss-Newton algorithm* and it will be discussed in Subsection 6.1.2.

### 4.5.5 Evolutionary (Genetic Algorithm) Hyperparameter Optimization

It uses the concepts of mutation and evolution, which aims to find the globally best combination of hyperparameters. We initialize the genetic algorithm by randomly selecting $M$ combination of hyperparameters, which are also known as candidates. In each iteration (generation), two best candidates, in terms of the

performance on a chosen score metric, are selected, which are called parents. These two parents will then generate two offsprings by crossover. Crossover mimics the sexual reproduction of two parents to generate two offsprings through the recombination of genes, in the context of hyperparameter tuning, the inner values of the parents are exchanged to form two new offsprings; this crossover will only happen with a probability $p_{\text{cross}}$; while with $1 - p_{\text{cross}}$ that the offsprings are the parents themselves. After that, these two offsprings will mutate to another candidates. Mutation is done by changing some inner hyperparameter values of the offspring to form one new candidate, and the mutation will only happen with a probability $p_{\text{mut}}$. Finally, the current generation together with the two offsprings will form a pool with $M + 2$ candidates. Among this pool, $M$ candidates are selected again based on the best score metric to form the next generation. The whole process iterates again and again until the difference between the best and the worst evaluated score metric falls below a pre-specified threshold $\varepsilon$ or a certain number of iterations is reached.

In practice, a high crossover rate is used, let say $p_{\text{cross}}$ = 90-95%, with a low mutation rate at around $p_{\text{mut}}$ = 5-10%, correspondingly. Even though mutation helps to maintain gene diversity so as to prevent from premature convergence, it is natural that a low mutation rate is used as we want to avoid the algorithm to be too similar to random search. Meanwhile, to introduce new offsprings different from the current generation, a high crossover rate should be used.

**Example 4.5.3.** Table 4.5.1 in Example 4.5.1 shows all possible combinations (candidates) of two hyperparameters, the penalty parameter $C$ and the kernel function KM. Assume that the current generation $\mathcal{G}$ is the set:

$$\mathcal{G} = \{(0.01, \text{L}), (10, \text{L}), (1, \text{R}), (1000, \text{R})\};$$

and assume that $(10, \text{L})$ and $(1000, \text{R})$ perform the best, *e.g.* having the highest accuracy, among this generation $\mathcal{G}$

1. In crossover, these two combinations, $(10, \text{L})$ and $(1000, \text{R})$, are recombined into two new combinations, $(10, \text{R})$ and $(1000, \text{L})$.
2. In mutation, we randomly select one hyperparameter, $C$ or KM, to mutate/change to a random value. For example, if we select the penalty parameter $C$, the offsprings are respectively mutated to $(100, \text{R})$ and $(100, \text{L})$.

We then compute the accuracies with $(100, \text{R})$ and $(100, \text{L})$. Finally, four best combinations are selected to form the next generation.

In the example above, since we only have two hyperparameters, the crossover can only be accomplished by using one cut-point that seperates these two hyperparameters, which is called the *single-point crossover* (See Figure 4.5.1). In general, if there are more hyperparameters, a greater number of cut-points can be used, let say $k$ of them, which is then called the *k-point crossover* (See Figure 4.5.2 for $k = 2$ with more than 3 hyperparameters). Moreover, we only have a finite set of hyperparameters, the mutation can only happen at some of them.

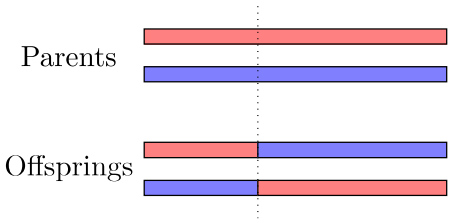



Figure 4.5.1: Single-point crossover.

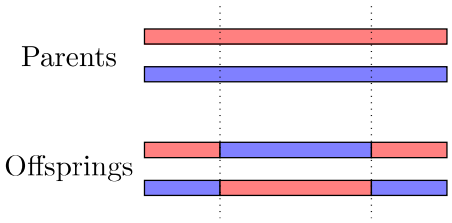



Figure 4.5.2: Two-(cut-)point crossover.

### 4.5.6 $K$-Fold Cross-Validation

When there are only a few examples, it could be prohibitive to have both validation and testing sets; yet, we still need to use more data to train up the model, by then **cross-validation**, a kind of resampling method, may help. To end this, we first fix the values of the hyperparameters, *e.g.* taking various combinations of hyperparameters from a grid search. Then split the training set into several subsets (fold or tranche) of essentially same size.

For example, with $K = 5$, Figure 4.5.3 shows a five-fold cross-validation by first shuffling randomly the dataset, then split the dataset into training and validation sets (80%) and test set (20%, the average box in Figure 4.5.3). After that, the training set is evenly split into five folds, $\{F_1, F_2, F_3, F_4, F_5\}$, each fold $F_i$ contain 20% of the training and validation set (or $20\% \times 80\% = 16\%$ of the entire dataset).

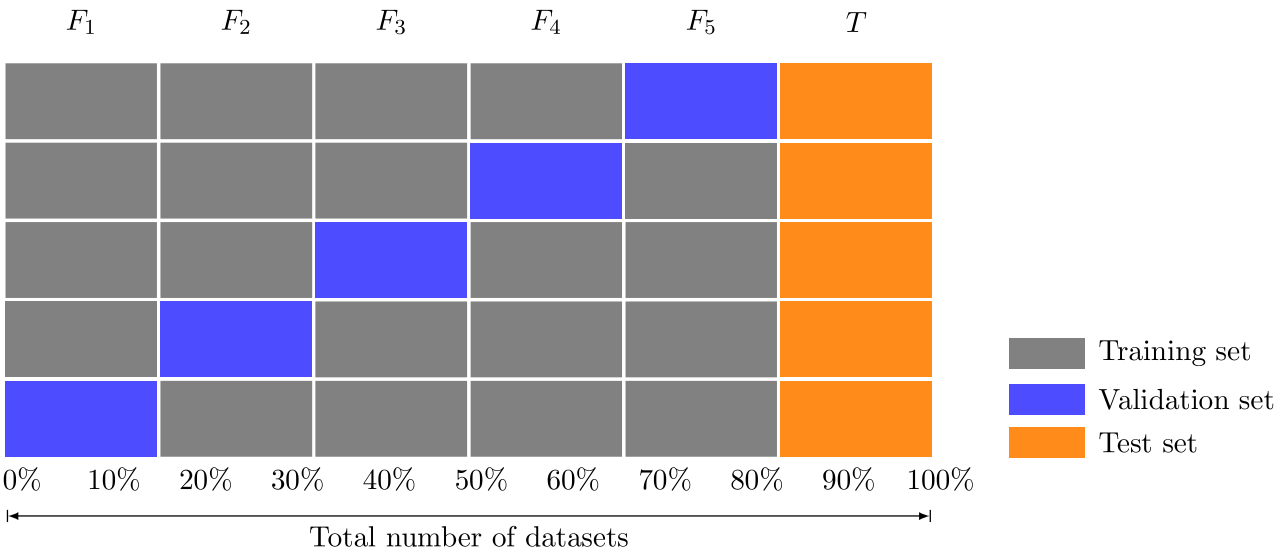



Figure 4.5.3: Five-fold cross-validation

Then, for each set of hyperparameters:

1. Train the first model $f_1$ for this set of hyperparameters with all examples from folds $F_2, F_3, F_4$, and $F_5$ as the training set; then calculate the value of the metric, *e.g.* MSE, for general machine learners, or those introduced in Section 4.4 for classifiers, by treating $F_1$, as the validation set.
2. in general, train the $k$-th model $f_k$ for the same combination of hyperparameters with all examples from folds $F_1, F_2, F_3, F_4, F_5$, except $F_k$ as the training set; then calculate the value of the metric with

$F_k$ as the validation set.

3. Average the five values of the metric obtained above to get the final value as the performance index for this set of hyperparameters.

By repeating the same procedure for all hyperparameters but on the same 5-fold cross validation set arrangements, the best combination of hyperparameters is the one which has the highest final value. Then, one uses the entire training set to build the model with these best values of hyperparameters. Finally, we assess the precision of the model using the testing dataset.

## 4.6 Imbalanced Data

In Section 4.4, we have introduced the use of accuracy and other score metrics to access the performance of a given classifier. However, when dealing with imbalanced data, *i.e.* some of the label (minority) classes have significantly fewer data samples than the other majority classes, *e.g.* a ratio of minority to majority $= 1 : 9$, the accuracy score metric can be misleading. For instance, the HSI dataset in the "`fin-ratio.csv`"[3] consists of 680 HK stocks, in which only 32 of them are HSI constituent stocks, and we aim to build a classifier to determine whether a given HK stock is HSI constituent stock or not. Consider a seemingly "good" classifier with 91.78% accuracy, but when we examine its confusion matrix:

| **Confusion Matrix** | | **Predicted** | | **Total** |
|---|---|---|---|---|
| | | **HSI** | **Not HSI** | |
| **Actual** | **HSI** | 0 (TP) | 32 (FN) | 32 |
| | **Not HSI** | 24 (FP) | 624 (TN) | 648 |
| **Total** | | 24 | 624 | 680 |

Table 4.6.1: The confusion matrix for HSI detection of a seemingly "good" classifier.

Indeed, the accuracy is calculated as $(0 + 624)/680 = 91.78\%$. However, this classifer is not capable of classifying the HSI constituent stock among all HK stocks.

To this end, we have illustrated an example where the imbalanced data can create an accuracy paradox. In dealing with imbalanced data, we can either over-sample our minority class (*e.g.* HSI constituent stocks) or under-sample our majority class (*e.g.* Non-HSI constituent stocks), which can be implemented as described below.

### 4.6.1 Oversampling

One naïve approach of oversampling is to randomly duplicate the minority samples, but this may lead to overfitting. One might regard the random duplication of the minority samples as increasing their corresponding weights, the model can be affected by the minority samples more notably than before, so that the bias decreases but the overfitting merges in a more apparent manner. Figure 4.6.1 illustrates this situation, where all yellow dots with blue circles embracing each of them form the minority class, while the red dots represent the majority class.

[3] See Subsection 5.5.1 for more description of this dataset.

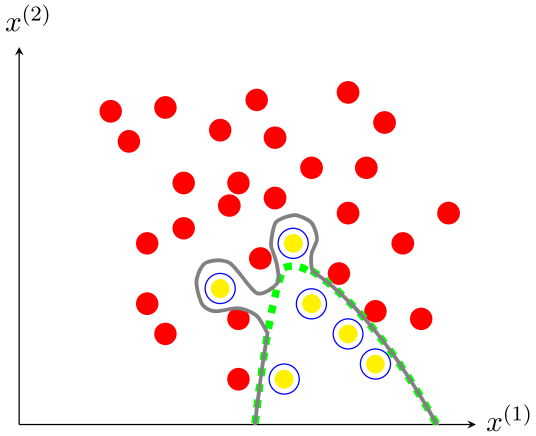



Figure 4.6.1: Overfitting by randomly duplicating samples.

Here the green dashed line represents the primitive decision boundary of a given classifier before oversampling, and the gray solid curve represents the decision boundary after the random oversampling of the minority class with duplicates. Since the weights of those minority samples increase, the model tilts heavily towards those two outward bounding minority samples so as to seperate them from the majority group of red dots. Therefore, the model fits well with a smaller bias, but the decision boundary becomes more complicated that gives a sign of overfitting.

In the following subsection, we shall introduce another oversampling method that provides a less biased and a more rectified decision boundary at the expense of causing mispecification from the majority group, *i.e.* some red dots being misclassified as the minority yellow dots.

#### 4.6.1.1 Synthetic Minority Oversampling Technique (SMOTE)

As one of the commonly used oversampling method, Synthetic Minority Oversampling Technique (SMOTE) proposed in Chawla et al. (2002), generates new minority samples by the linear interpolation between two chosen minority samples. Let $c_{\max}$ denote the label of majority class which has the largest sample size, and $\mathcal{C}_-$ denote the set of all labels $c_m$ that are not $c_{\max}$. Moreover, let $\mathcal{R}_{\max}$ denote the set of all feature vectors of majority class $c_{\max}$, while $\mathcal{R}_m$ denotes the set of all feature vectors of a minority class with a label $c_m \in \mathcal{C}_-$, then a typical SMOTE algorithm runs as follows:

**Algorithm 4.2** SMOTE Algorithm

1: **for** $m : c_m \in \mathcal{C}_-$ **do**
2: Initialize $\mathcal{R}_* = \emptyset$.
3: Randomly select a datum $\boldsymbol{x}_t \in \mathcal{R}_m$.
4: Perform $K$-nearest neighbour with the set $\mathcal{R}_m$, where $K$ is a hyperparameter; denote the $K$-nearest neighbours of $\boldsymbol{x}_t$ as $\boldsymbol{x}_t(1), \cdots, \boldsymbol{x}_t(K)$.
5: Randomly select one neighbour with the label $c_m$, denoted as $\boldsymbol{x}_t(j)$ for some $j = 1, \cdots, K$, the new datum $\boldsymbol{x}_{t,\text{new}}$, called *synthetic sample*, is the linear interpolation:

$$\boldsymbol{x}_{t,\text{new}}(j) = \boldsymbol{x}_t(1 - u_t^{(j)}) + u_t \boldsymbol{x}_t(j)\,, \qquad u_t^{(j)} \sim U[0,1]$$

6: Augment this new datum to $\mathcal{R}_*$, *i.e.* $\mathcal{R}_* \leftarrow \mathcal{R}_* \cup \{\boldsymbol{x}_{t,\text{new}}\}$.
7: Repeat Steps 3, 4, 5, and 6 for $t = 1, 2, \cdots$ until the sum of the number of minority samples with label $c_m$ and the number of synthetic samples equals to the number of majority samples, *i.e.* $|\mathcal{R}_m| + |\mathcal{R}_*| = |\mathcal{R}_{\max}|$.
8: $\mathcal{R}_m \leftarrow \mathcal{R}_m \cup \mathcal{R}_*$ and label each feature vector in $\mathcal{R}_m$ as class $c_m$.
9: **end for**

After the completion of the SMOTE algorithm above, each labeled class will have the same sample size of $|\mathcal{R}_{\max}|$. In the simplified example depicted in Figure 4.6.1, there are only two classes. Under the SMOTE algorithm, Figure 4.6.2 illustrates the first iteration $t = 1$ with $K = 3$, in which the blue star with a yellow cross inside is the corresponding synthetic new datum $\boldsymbol{x}_{1,\text{new}}(j)$.

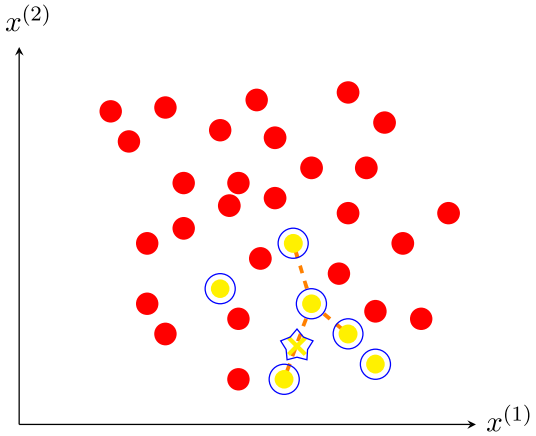



Figure 4.6.2: First iteration of SMOTE algorithm with $K = 3$.

In finding the $K$-nearest neighbours, the Euclidean distance defined in (3.2.1) is used to measure the distance between any two points.

```
> euclidean_distance <- function(x_i, x_j){
+     return (sqrt(sum((x_i - x_j)^2)))
+ }
>
> knear <- function(P, n_clust=1){       # P: Minority data matrix
+     distance <- outer(data.frame(t(P)), data.frame(t(P)),
+                       Vectorize(euclidean_distance))
+     # Avoid picking itself
```

```
+     distance <- distance + diag(rep(max(distance) + 1, ncol(distance)))
+     distance_sort <- apply(distance, 1, sort, index.return=TRUE)
+     distance_sort_ind <- t(as.data.frame(lapply(distance_sort, FUN="[","ix")))
+     return (distance_sort_ind[,1:n_clust])
+ }
```

Programme 4.6.1: Functions for Euclidean distance and $K$-nearest neighbours in **R**.

Here the `outer()` function returns the outer product of two arrays `data.frame(t(P))` and itself with the function `Vectorize(euclidean_distance)`; mathematically, let $\boldsymbol{P} = (x_i^{(d)}) \in \mathbb{R}^{N_- \times D}$, where $N_-$ is the number of samples in the minority class, then this `outer()` function $\mathbb{R}^{D\times N_-} \times \mathbb{R}^{D\times N_-} \ni (\boldsymbol{P}^\top, \boldsymbol{P}^\top) \mapsto (\|\boldsymbol{P}_{i\cdot} - \boldsymbol{P}_{j\cdot}\|_2) \in \mathbb{R}^{N_-\times N_-}$, where $\boldsymbol{P}_{i\cdot} = (x_i^{(1)}, \cdots, x_i^{(D)})^\top$. This `outer()` computes the Euclidean distance between each sample datum and all other sample data, including the same sample datum itself. Thus, we add the maximum value of the distance matrix by `max(distance) + 1` to its diagonal entries *i.e.* the Euclidean distance between the sample datum and itself is set with the maximum value plus one. In light of this, the input only accepts one single matrix but not more, which is one limitation of this method. `index.return=TRUE` is the argument for the `sort()` function, which returns not only the sorted values but also the index position. The index position is stored with an attribute named as `ix`, we retreive only the index position sorted according to their smallest Euclidean distance by using the `lapply()` function, in which `FUN="["` together with the additional argument `"ix"` indicate that the sorted index position of $\boldsymbol{x}_t(1), \cdots, \boldsymbol{x}_t(K), \boldsymbol{x}_t(K+1), \cdots, \boldsymbol{x}_t(N_- - 1)$ is returned, *i.e.* $N_- - 1$ for not including $\boldsymbol{x}_t$ itself. Finally, we return the first $K$=`n_clust` number of sorted index positions of $\boldsymbol{x}_t(1), \cdots, \boldsymbol{x}_t(K)$ for each sample datum $\boldsymbol{x}_t$.

Next, we implement the SMOTE Algorithm 4.2 in **R**.

```
> SMOTE_self <- function(X, y, K=5, Mino_class=NULL, Mino_factor=1,
+                        shuffle=TRUE, seed=NULL){
+     if (!is.null(seed)){ set.seed(seed) }
+     y <- as.vector(y)
+     Majo_class <- names(which.max(table(y)))
+     Majo_index <- which(y == Majo_class)
+     if (is.null(Mino_class)){
+         Mino_class <- unique(y)[unique(y) != Majo_class]
+     }
+
+     feature <- X
+     label <- y
+     for (class_m in Mino_class){
+         Mino_index <- which(y == class_m)
+         Mino_X <- X[Mino_index,]
+         Mino_X <- Mino_X[sample(length(Mino_index)),] # Shuffle the data
+
+         # Step 4: Identifying K nearest neighbour points for each point
+         knear_index <- knear(P=Mino_X, n_clust=K)
```

```
+
+         # Step 7: Find the number of iterations
+         max_it <- (length(Majo_index) - length(Mino_index)) * Mino_factor
+         # Step 3: Randomly select a new datum
+         row_id <- sample(1:length(Mino_index), max_it, replace=TRUE)
+         # Step 5: Generating synthetic new data
+         col_id <- sample(1:K, max_it, replace=TRUE)   # select one neighbour
+         Synt_id <- knear_index[cbind(row_id, col_id)]
+         u <- runif(max_it)
+         Synt_X <- Mino_X[row_id,] * (1 - u) + u * Mino_X[Synt_id,]
+         Synt_y <- rep(class_m, max_it)
+         # Step 8: Augment the synthetic data to the original data
+         feature <- rbind(feature, Synt_X)
+         label <- c(label, Synt_y)
+     }
+
+     if (shuffle){
+         shuf_id <- sample(1:length(label), length(label))
+         feature <- feature[shuf_id,]
+         label <- label[shuf_id]
+     }
+     return (list(X=feature, y=label))
+ }
```

Programme 4.6.2: SMOTE function using Programme 4.6.1 for `euclidean_distance()` and `knear()` in **R**.

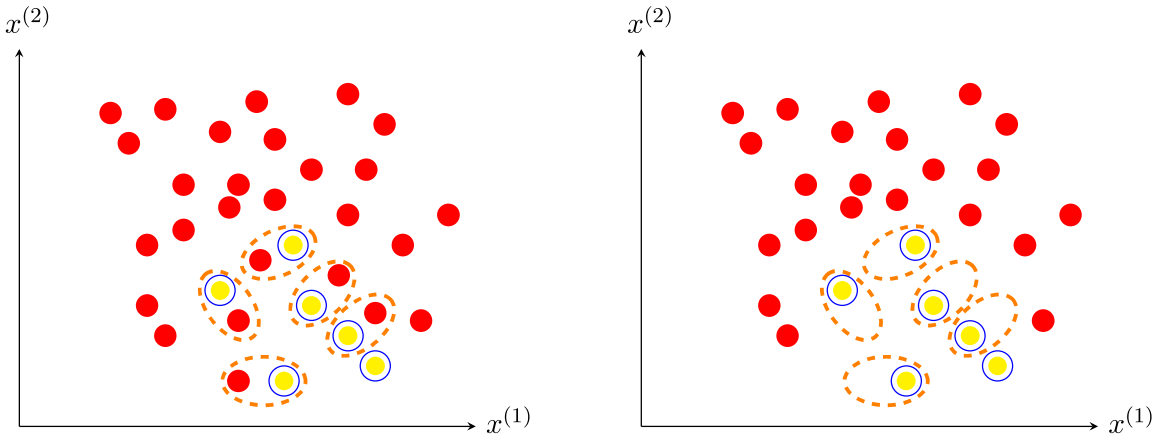
The `SMOTE_self()` function accepts 7 arguments: the parameter `X` takes the feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ as input; `y` accepts the label `numeric`, `array`, or `factor` type object; the parameter `K` is a positive integer indicating the number of neighbours in the $K$-nearest neighbors algorithm, the `Mino_class` parameter indicates which the minority class labels will be oversampled by this function; the `Mino_factor` parameter controls number of synthesized minority samples; the `shuffle` parameter is a boolean parameter indicating that whether the returned dataset is shuffled or not; and the `seed` parameter accepts a numeric value for the seed to kick off the random numbers generation for $u_t^{(j)}$'s. In particular, if `Mino_class=NULL`, then the function will treat all classes which are not labelled as $c_{\max}$ as the minority classes. As afore-mentioned in Programme 4.6.1, the `knear()` function returns an index matrix of shape $|\mathcal{R}_m| \times K$, in which each columns are the indices of the $K$ nearest neigbours for each minority sample. Finally, the function `SMOTE_self()` returns a list object with two items, namely `X` and `y` being respectively the feature vector and the corresponding label after the oversampling with SMOTE.

### 4.6.2 Undersampling

One naïve approach of undersampling is to randomly remove some samples from the classes with a majority in size, but this may leads to information loss. To reduce such loss, there are some commonly adopted undersampling method, such as Tomek links and Edited Nearest Neighbour to be described below.

#### 4.6.2.1 Tomek Links

Tomek links, first proposed in Tomek (1976), are different pairs close enough in "distance" consisting of one from a minority class and another from a majority class: suppose that $\boldsymbol{x}_i \in \mathcal{R}_{\min}$ and $\boldsymbol{x}_j \in \mathcal{R}_{\max}$ are respectively the feature vectors of the minority class and the majority class, then the pair $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a Tomek link if (1) for all $\boldsymbol{x}_\ell (\neq \boldsymbol{x}_j) \in \mathcal{R}_{\max}$, we have $d(\boldsymbol{x}_i, \boldsymbol{x}_j) < d(\boldsymbol{x}_i, \boldsymbol{x}_\ell)$ **and** (2) for all $\boldsymbol{x}_h (\neq \boldsymbol{x}_i) \in \mathcal{R}_{\min}$, $d(\boldsymbol{x}_i, \boldsymbol{x}_j) < d(\boldsymbol{x}_h, \boldsymbol{x}_j)$, where $d$ is a chosen distance metric, recall the notion of various distance in Section 3.2, for instance the Euclidean distance. Then, we downsample the majority group of the dataset by removing point from the majority class in each Tomek link. Unlike SMOTE, Tomek link considers that sets with two classes only, but not one with multi-class. Figure 4.6.3 illustrates an example with five Tomek links enclosed by the orange dashed ovals. Clearly in Figure 4.6.3(b), the minority samples and the majority samples are well seperated after the removal. Note that there is one minority sample point which does not form a Tomek link, because its closest neighbour is also from the minority group but not for the majority group.




(a) Before removing the majority samples. (b) After removing the majority samples.

Figure 4.6.3: Five pairs of Tomek links.

In the following, we shall provide an algorithm for identifying Tomek links and then removing the corresponding sample points from the majority group in **R**.

```
> Tomek_link <- function(X, y, remove_Mino=FALSE){
+     # Only for two classes, but not multi-classes
+     Mino_class <- names(which.min(table(y)))
+     Majo_index <- which(y != Mino_class)
+
+     # looking for the nearest neighbour for each point from the whole dataset
+     knear_index <- knear(P=X, n_clust=1)
+     names(knear_index) <- 1:length(y)
+     link_index <- knear_index[knear_index]
+     # identifying all pairs from majority and minority group
+     Majo_link <- (1:length(y) == link_index) * (y == Mino_class)
+     Majo_link <- Majo_link * (names(Majo_link)%in%Majo_index)
```

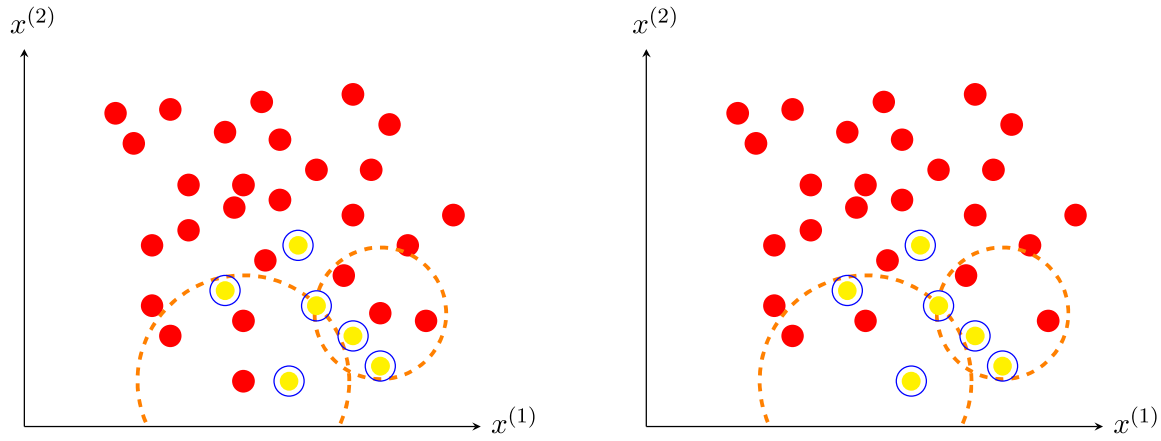
```
+     # identifying the index position of majority group in all Tomek links
+     Tomek_index <- names(which(Majo_link==1))
+
+     if (remove_Mino){ # For SMOTE+Tomek algorithm
+         Mino_index <- which(y == Mino_class)
+         Mino_link <- (1:length(y) == link_index) * (y != Mino_class)
+         Mino_link <- Mino_link * (names(Mino_link)%in%Mino_index)
+         Tomek_index <- c(Tomek_index, names(which(Mino_link==1)))
+     }
+     # Remove the majority samples in all Tomek links
+     Tomek_index <- as.integer(Tomek_index)
+     if (length(Tomek_index) > 0){   # Check if integer(0), i.e. no Tomek link
+         feature <- X[-Tomek_index,]
+         label <- y[-Tomek_index]
+     }else{ # Not to remove
+         feature <- X
+         label <- y
+     }
+     return (list(X=feature, y=label))
+ }
```

Programme 4.6.3: Tomek link using Programme 4.6.1 for `euclidean_distance()` and `knear()` in **R**.

First of all, here the `Tomek_link()` function takes in 3 arguments, where the first two parameters are as those first two for `SMOTE_self()` function in Programme 4.6.2, and the last boolean parameter `remove_Mino` is used for a later purpose such that we shall defer its explaination to Subsection **??**. This function first finds the closest neighbour no matter from the same group or not, through the argument `n_clust=1` using the `knear()` function in Programme 4.6.1. Then it finds the index position of the majority class in all Tomek links and saved it as `Tomek_index`. Note that `Tomek_index` can be empty (`integer(0)` in **R**) if there exists no Tomet link, and we can check `Tomek_index` is not `integer(0)` by the statement `length(Tomek_index) > 0`.

#### 4.6.2.2 Edited Nearest Neighbour (ENN)

Edited Nearest Neighbour, proposed in Wilson (1972), is another commonly adopted undersampling technique, which also uses the $K$-nearest neighbour algorithm and it only considers two classes. Again, $\boldsymbol{x}_j \in \mathcal{R}_{\text{max}}$ denotes the feature vectors of the majority class, then for each $\boldsymbol{x}_j \in \mathcal{R}_{\text{max}}$, we perform a $K$-nearest neighbor. If most of the $K$ neighbours are not the majority samples, then we shall remove $\boldsymbol{x}_j$; indeed, in practice, if the ratio to the whole of the number of majority samples in the $K$ nearest neighbour of $\boldsymbol{x}_j$ is less than a pre-determined threshold (a hyperparameter), then we shall remove $\boldsymbol{x}_j$.




(a) Before removing the majority samples. (b) After removing the majority samples.

Figure 4.6.4: Edited Nearest Neighbour with $K = 5$ and threshold of 0.5.

The following programme is used as the realization of Edited Nearest Neigbour in **R**:

```
> Edited_Nearest_Neighbor <- function(X, y, K=5, threshold=0.5){
+   # Only for two classes, but not multi-classes
+   Mino_class <- names(which.min(table(y)))
+   Majo_index <- which(y != Mino_class)
+
+   # identifying K nearest neighbour points for each point
+   knear_index <- knear(P=X, n_clust=K)
+   rownames(knear_index) <- 1:length(y)
+   Majo_link <- knear_index[Majo_index,]
+   # compute the ratio of the whole of the number of majority samples
+   Majo_ratio <- apply(Majo_link, 1, function(x) sum(x%in%Majo_index)/K)
+   # listing out the index positions for removal
+   ENN_index <- as.integer(names(which(Majo_ratio < threshold)))
+   if (length(ENN_index) > 0){    # Check if integer(0), i.e. no ENN
+     feature <- X[-ENN_index,]
+     label <- y[-ENN_index]
+   }else{ # Not to remove
+     feature <- X
+     label <- y
+   }
+   return (list(X=feature, y=label, Removal=ENN_index))
+ }
```

Programme 4.6.4: Edited Nearest Neighbour in **R**.

Here the `Edited_Nearest_Neighbor()` function takes 4 arguments, where the first three are the same as the first three in `SMOTE_self()` function in Programme 4.6.2, and the last argument is the threshold for the ratio. This function first identifies the $K$ nearest neighboring points of each data point in the dataset, and

then compute the required ratio. If the ratio is less than the pre-determined threshold, then we shall remove the corresponding majority sample point.

### 4.6.3 Examples with Oversampling and Undersampling

In this subsection, we shall illustrate the use and the implication of the oversampling with SMOTE and undersampling with Tomek link in **R**. For simplicity, we use a $D = 2$ dimensional feature vector as an illustration. Firstly, we generate some Gaussian random numbers with `rnorm()` function.

```
> set.seed(4012)
>
> X11 <- rnorm(10)
> X12 <- rnorm(10)
> X21 <- rnorm(30, 0.4, 2)
> X22 <- rnorm(30, 0.4, 2)
> y1 <- rep(0, 10)      # 10 data points for minority group
> y2 <- rep(1, 30)      # 30 data points for majority group
>
> X1 <- cbind(X11, X12)
> X2 <- cbind(X21, X22)
> X <- rbind(X1, X2)
> y <- c(y1, y2)
> idx <- sample(1:length(y), length(y))
> X <- X[idx,]
> y <- y[idx]
> # color index 1 for minority and 3 for majority
> plot(X[,1], X[,2], pch=19, col=2*y+1, asp=1, cex=1.7)
> legend("topright", c("Minority", "Majority"), col=c(1,3), pch=19, cex=1.5)
```

Programme 4.6.5: Generating dataset following normal distribution in **R**.

Here the argument `asp=1` in the `plot()` function indicates that the same scaling is used in the *x-axis* and the *y-axis*, such that it is easier to observe the actual distance between different sample data. The distribution of points are shwon as follows:

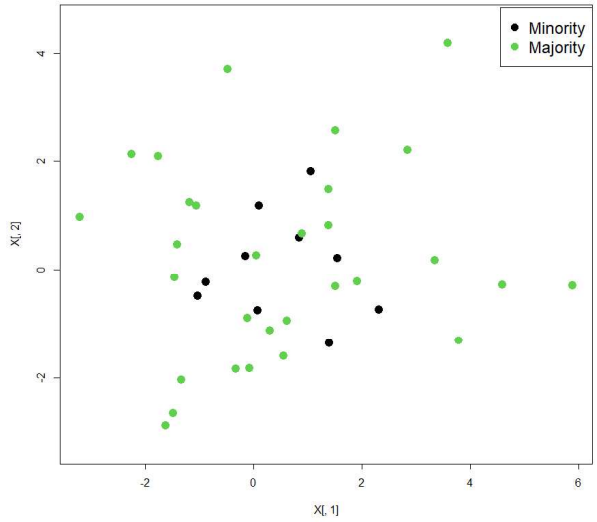



Figure 4.6.5: Distribution of Minority class and Majority class in **R**, generated in Programme 4.6.5.

Here the 10 black dots represent the minority class with both $x^{(1)}$ and $x^{(2)}$ independently following $\mathcal{N}(0,1)$ and the 30 green dots represent the majority class with both $x^{(1)}$ and $x^{(2)}$ independently following $\mathcal{N}(0.4,\sqrt{2})$. The argument `pch=19` in the `plot()` function and the `legend()` function indicates that a solid circle is used in plotting the points.

In dealing with imbalanced data, oversampling and undersampling techniques can be used simultaneously. Let us first undersample the dataset with `Tomek_link()`:

```
> result <- Tomek_link(X, y)
> plot(result$X[,1], result$X[,2], col=2*result$y+1, pch=19, asp=1, cex=1.7)
> legend("topright", c("Minority", "Majority"), col=c(1,3), pch=19, cex=1.5)
```

Programme 4.6.6: Using Tomek link in Programme 4.6.3 to the dataset generated by Programme 4.6.5 in **R**.

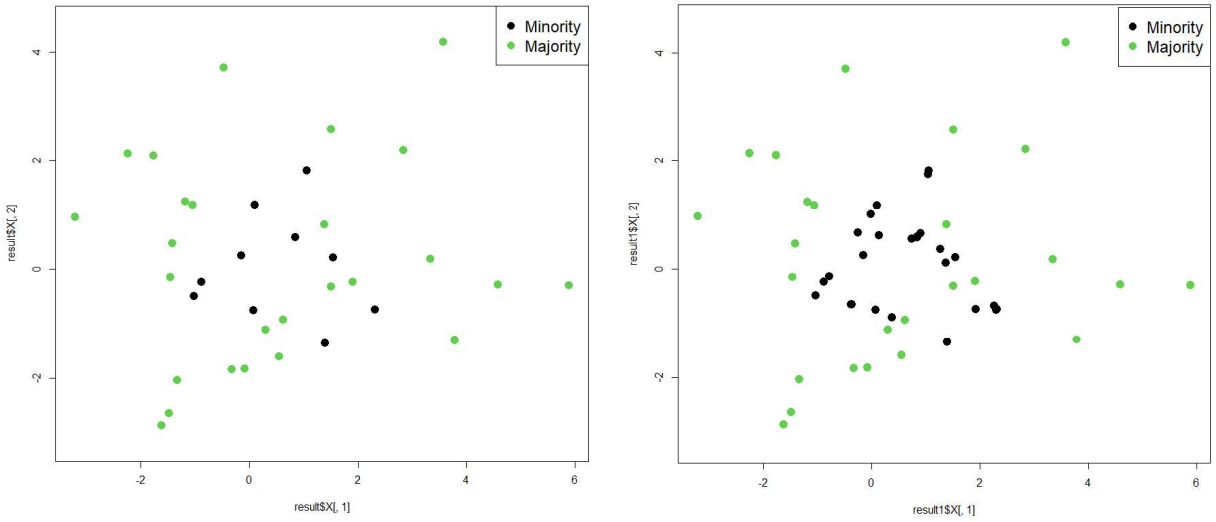
and next we oversample the resulting dataset with `SMOTE()`:

```
> df <- cbind(result$X, result$y)
> result1 <- SMOTE_self(result$X, result$y, K=5)
> plot(result1$X[,1], result1$X[,2], col=2*result1$y+1, pch=19, asp=1, cex=1.7)
> legend("topright", c("Minority", "Majority"), col=c(1,3), pch=19, cex=1.5)
>
> table(result1$y)

 0  1
26 26
```

Programme 4.6.7: Using SMOTE in Programme 4.6.2 to the dataset after Tomek link in Programme 4.6.6

Finally, with the `table(result$y)` function, we observe that the total number of samples become 52, where 4 majority samples were removed, making the majority group size to be 26; and then 16 minority samples were added via SMOTE. The respective plots in order are shown below:




(a) after undersampling with Tomek Link, generated in Programme 4.6.6.

(b) after further oversampling with SMOTE, generated in Programme 4.6.7.

Figure 4.6.6: Distribution of Minority class and Majority class in **R**.

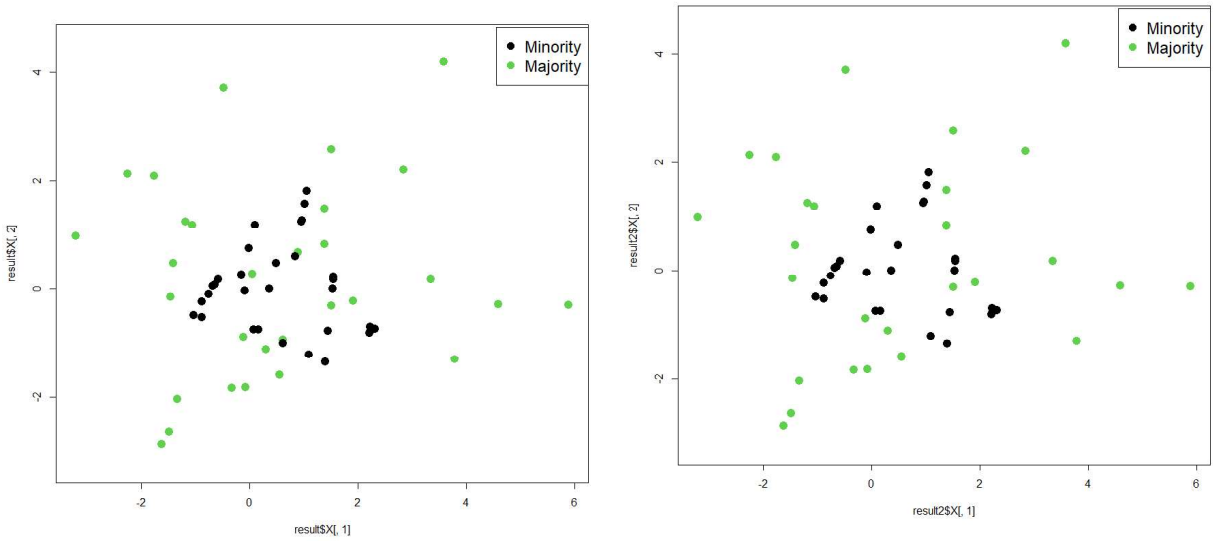
However, in Figure (b) 4.6.6, we observe that there are still some majority class data points lying inside the proximity of synthesized minority class making the resulting decision boundary still complicated. Instead, Batista et al. (2003) proposed the "*SMOTE+Tomek*" algorithm where we first perform the oversampling with SMOTE, and then following the undersampling with Tomek link. However, different from the usual Tomek link in Subsubsection 4.6.2.1, we shall remove both majority class and minority class of all Tomek links, instead of just the usual removal of majority class. Therefore, in Programme 4.6.3, we have an additional boolean argument `remove_Mino` stating that whether we shall also remove the minority classes in all Tomek links:

```
> result <- SMOTE_self(X, y, K=5, seed=4012)
> plot(result$X[,1], result$X[,2], col=2*result$y+1, pch=19, asp=1, cex=1.7)
> legend("topright", c("Minority", "Majority"), col=c(1,3), pch=19, cex=1.5)
>
> result2 <- Tomek_link(result$X, result$y, remove_Mino=TRUE)
> plot(result2$X[,1], result2$X[,2], col=2*result2$y+1, pch=19, asp=1, cex=1.7)
> legend("topright", c("Minority", "Majority"), col=c(1,3), pch=19, cex=1.5)
>
> table(result2$y)

 0  1
27 27
```

Programme 4.6.8: SMOTE+Tomek algorithm using Programmes 4.6.2 and 4.6.3 in **R**.

Here, the results from `table(result$y)` tell us that the total number of samples is 54, where 20 minority samples were first added via SMOTE, making the minority group size of 30; and then the same 3 majority and minority samples are removed, making both the majority and minority group sizes to be 27. The respective plots in order are shown below:




(a) After oversampling with SMOTE.

(b) After further undersampling with Tomek link and `remove_Mino=TRUE`.

Figure 4.6.7: Distribution of Minority class and Majority class in **R**, generated in Programme 4.6.8.

From Figure 4.6.7, we see that the decision boundary is less complicated when comparing with the results in Figure 4.6.6.

A year later, Batista et al. (2004) replaces Tomek link with ENN and becomes an "*SMOTE+ENN*" algorithm. As the same practice of "*SMOTE+Tomek*", we first oversample with SMOTE, but next apply ENN for undersampling. However, one should notice that ENN only removes the majority samples such that the size of the minority group after this algorithm is larger or equal to that of the majority group. Therefore, a common practice is to reduce the number of synthesized minority samples, let say we only generate $0.7 \times (|\mathcal{R}_{\max}| - |\mathcal{R}_{\min}|)$ synthetic minority samples from SMOTE. Then, using the same settings in Programme 4.6.5, we have the following procedures in **R**:

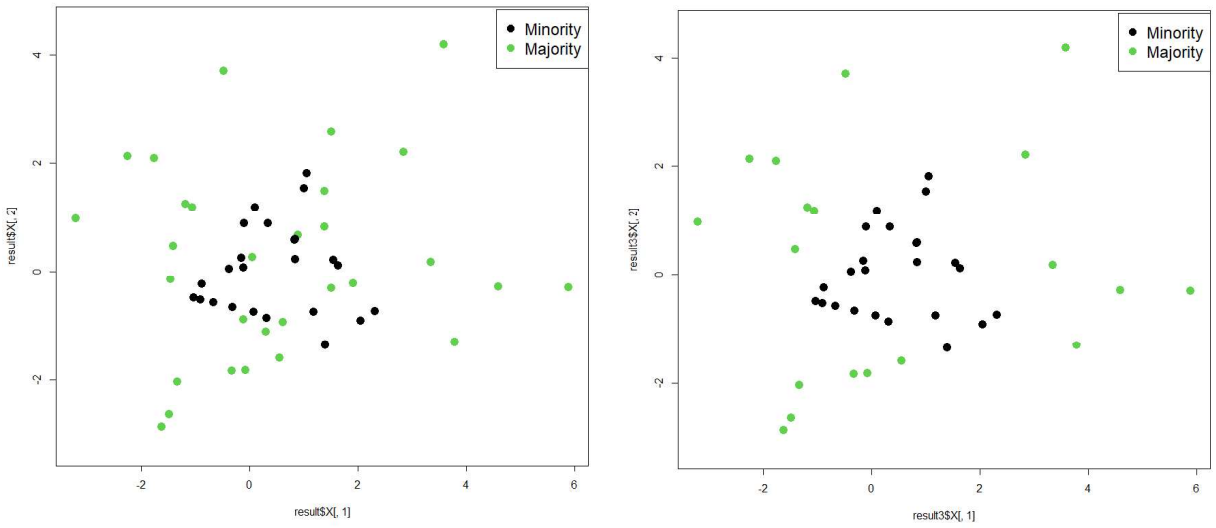
```
> result <- SMOTE_self(X, y, K=5, Mino_factor=0.7, seed=4012)
> plot(result$X[,1], result$X[,2], col=2*result$y+1, pch=19, asp=1, cex=1.7)
> legend("topright", c("Minority", "Majority"), col=c(1,3), pch=19, cex=1.5)
>
> result3 <- Edited_Nearest_Neighbor(result$X, result$y, K=3, threshold=0.5)
> plot(result3$X[,1], result3$X[,2], col=2*result3$y+1, pch=19, asp=1, cex=1.7)
```

```
> legend("topright", c("Minority", "Majority"), col=c(1,3), pch=19, cex=1.5)
> table(result3$y)

 0  1
24 19
```

Programme 4.6.9: SMOTE+ENN algorithm using Programmes 4.6.2 and 4.6.4 in **R**.

Here, since the factor of 0.7 is used in `Mino_factor`, we shall first synthesize $0.7 \cdot (30 - 10) = 14$ samples of minority class such that there are $10 + 14 = 24$ samples belong to the minority group via SMOTE; and then 11 majority samples were removed, making the majority group size to be 19.




(a) after oversampling with SMOTE and `Mino_factor=0.7`.

(b) after further undersampling with ENN.

Figure 4.6.8: Distribution of Minority class and Majority class in **R**, generated in Programme 4.6.9.

Comparing Figures (b) 4.6.8 and (b) 4.6.7, we immediately observe that the decision boundary provided by "*SMOTE+ENN*" is slightly less complicated.

#### 4.6.3.1 Multivariate Normality Test in R

In Subsection 2.5.4, we have introduced several univariate normality tests in **R** and Python. To this end, we may want to test whether the vector $\boldsymbol{x} = (x^{(1)}, \cdots, x^{(D)})^\top$ with a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$ jointly follows a multivariate normal distribution, *i.e.* $\boldsymbol{x}_n \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, for $n = 1, \cdots, N$. We can extend the idea of normal Q-Q plots to test multivariate normality. First, let us recall the square of **Mahalanobis** distance in (3.2.7):

$$d_n^2 = (\boldsymbol{x}_n - \bar{\boldsymbol{x}})^\top \boldsymbol{S}^{-1} (\boldsymbol{x}_n - \bar{\boldsymbol{x}}), \qquad n = 1, \cdots, N, \tag{4.6.1}$$

where $\bar{\boldsymbol{x}}$ and $\boldsymbol{S}$ are the sample unbiased estimates of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ respectively. Note that $d_n$ is a number representing the distance from the observation $\boldsymbol{x}_n$ to the central tendency, and $d_n^2$ is sometimes called the

squared generalized distance of $\boldsymbol{x}_n$. According to distribution theory, if $\boldsymbol{x}_n$ has a multivariate normal distribution, then for large $N$, $d_n^2$ will approximately follow a Chi-square distribution with $D$ degrees of freedom.

Therefore, we can generate a chi-square Q-Q plot as follows:

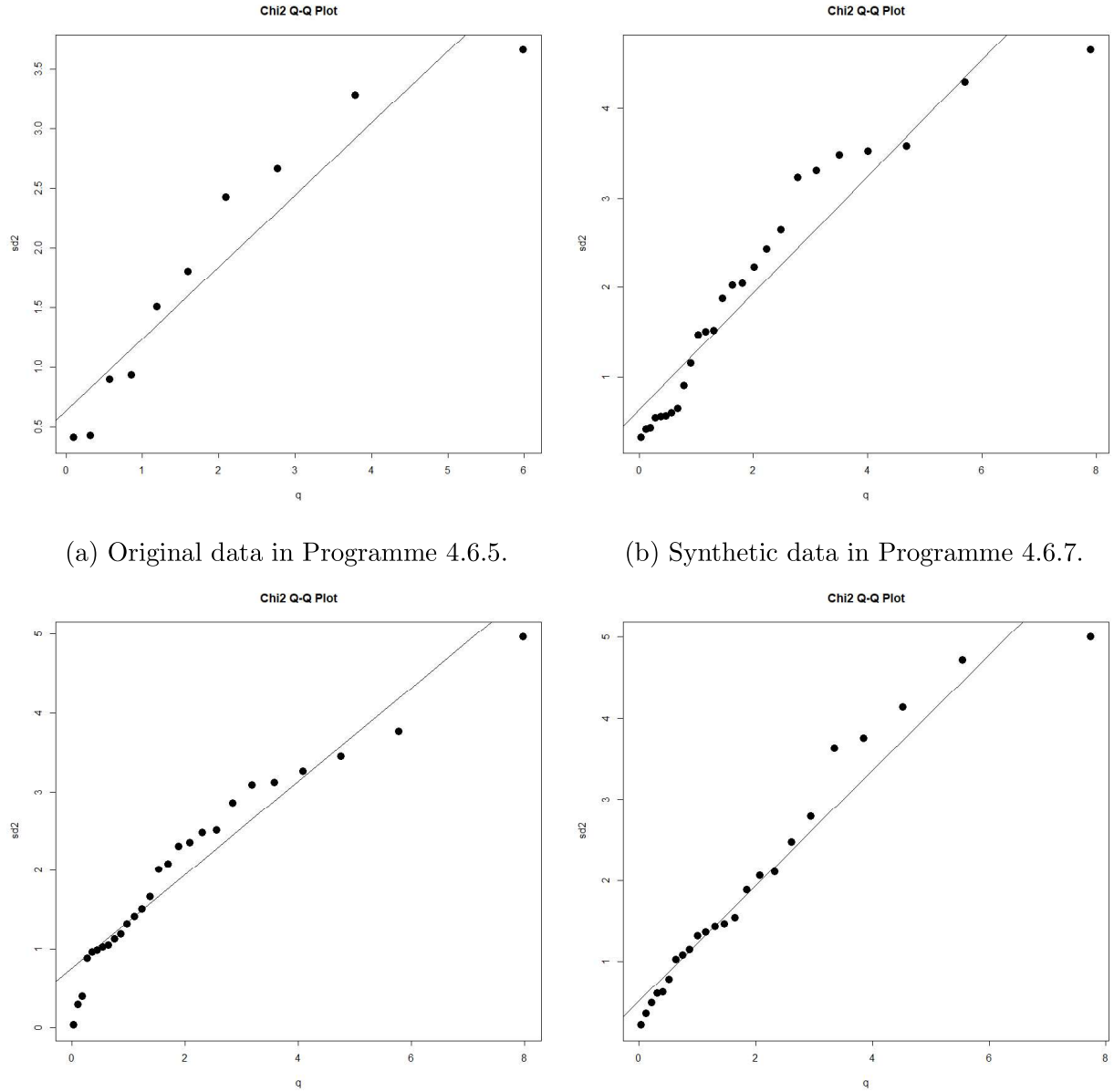
1. Order the $d_n^2$ in ascending order, and denote them by $d_{(n)}^2$ in order.
2. Compute $q_D(n)$, the $n$-th quantile of the chi-square distribution with $D$ degrees of freedom, i.e. $\mathbb{P}(\chi_D^2 < q_D(n)) = (n - 0.5)/N$.
3. Plot $d_{(n)}^2$ against $q_D(n)$. If the points are close to a straight line, the original $\boldsymbol{x}_n$'s follow a multivariate normal distribution.

Let us illustrate this chi-square Q-Q plot by testing whether each set of generated minority data in Subsection 4.6.3 has a bi-variate normal distribution. First, we compute the sample mean vector and the sample variance-covariance matrix for each set of minority data. Then we compute the inverse of $\boldsymbol{S}$, the squared generalized distance and the quantiles of the Chi-square distribution in **R**:

```
> Multivariate_Normality <- function(X){
+     mu <- apply(X, 2, mean)
+     S <- cov(X)
+     z <- sweep(X, 2, mu)
+     d2 <- diag(z%*%solve(S)%*%t(z))
+     sd2 <- sort(d2)                # sort d2 in ascendingly
+     N <- nrow(X)
+     df <- ncol(X)
+     i <- ((1:N)-0.5)/N             # create percentile vector
+     q <- qchisq(i, df)             # compute quantiles
+
+     par(mfrow=c(1,1))
+     qqplot(q, sd2, main="Chi2 Q-Q Plot", pch=19, cex=1.7) # QQ-chisquare plot
+     abline(lsfit(q, sd2))
+     return (d2)                    # squared generalized distance
+ }
>
> Mino_X0 <- X1
> Mino_X1 <- result1$X[result1$y == 0,]  # 0 for minority class
> Mino_X2 <- result2$X[result2$y == 0,]
> Mino_X3 <- result3$X[result3$y == 0,]
>
> d2_0 <- Multivariate_Normality(Mino_X0)
> d2_1 <- Multivariate_Normality(Mino_X1)
> d2_2 <- Multivariate_Normality(Mino_X2)
> d2_3 <- Multivariate_Normality(Mino_X3)
```

Programme 4.6.10: Chi-square QQ-plot in the final minority samples generated in Programmes 4.6.5, 4.6.7, 4.6.8, and 4.6.9 in **R**.

Here the diagonal of the matrix `z %*% solve(S) %*% t(z)` is a vector of $d_n^2$ defined in (3.2.7). This is not a straight forward but a convenient way to compute $d_n^2$ if the underlying dimension is not too large.




(a) Original data in Programme 4.6.5.

(b) Synthetic data in Programme 4.6.7.

(c) "*SMOTE+Tomek*" in Programme 4.6.8.

(d) "*SMOTE+ENN*" in Programme 4.6.9.

Figure 4.6.9: 4 Chi-square QQ-plots in **R**, generated in Programme 4.6.10.

From the plot, the distribution of the minority samples are not significantly different from a multivariate normal distribution (again, except the tail part). We next provide a second evidence using the `ks.test()` function to test whether $d_n^2 \sim \chi_D^2$ in **R**:

```
> ks.test(d2_0, y=pchisq, df=ncol(Mino_X0))

        One-sample Kolmogorov-Smirnov test

data:  d2_0
D = 0.1863, p-value = 0.8176
alternative hypothesis: two-sided

> ks.test(d2_1, y=pchisq, df=ncol(Mino_X1))

        One-sample Kolmogorov-Smirnov test

data:  d2_1
D = 0.15281, p-value = 0.5286
alternative hypothesis: two-sided

> ks.test(d2_2, y=pchisq, df=ncol(Mino_X2))

        One-sample Kolmogorov-Smirnov test

data:  d2_2
D = 0.24535, p-value = 0.06453
alternative hypothesis: two-sided

> ks.test(d2_3, y=pchisq, df=ncol(Mino_X3))

        One-sample Kolmogorov-Smirnov test

data:  d2_3
D = 0.15087, p-value = 0.593
alternative hypothesis: two-sided
```

Programme 4.6.11: KS test of $d_n^2$'s obtained in Programme 4.6.10 in **R**.

Now that the four $p$-values for each set of minority samples are greater than 0.05, we may still accept $H_0$ that all `d2`'s follow a Chi-square distribution with 2 degrees of freedom, or all $\boldsymbol{x} = (x^{(1)}, x^{(2)})^\top$'s have a bivariate normal distribution. However, we clearly observe that the $p$-value for the minority data synthesized from "*SMOTE+Tomek*" is just slightly above the threshold of 0.05, and such method seems to distort the original distribution of the minority data; indeed, we delete both majority and minority samples of all Tomek links, we might even remove the original minority data, which leads to information loss. On the other hand, "SMOTE+ENN" and the method introduced in Programmes 4.6.7 do not remove any samples from minority group, except that "SMOTE+ENN" synthesizes less number of samples. Lastly, the initial number of minority samples is insufficient, there might not provide much information about the distribution of minorty group for sythesizing samples.

# 4.A Appendices

## 4.A.1 Notations of Big O and Small o in Probability

In mathematical proofs, we frequently want to say "the remainder term $\mathrm{x}_n$ is small", or equivalently, $\mathrm{x}_n$ converges to zero or to a boundary. The big $O_p$ and small $o_p$ notations are our desired short-hand notations.

**Definition 4.A.1.** Let $\{\mathrm{x}_n \in \mathbb{R}\}_{n\in\mathbb{N}}$ and $\{\mathrm{a}_n \in \mathbb{R}^+\}$ be sequences of random variables.

1. **Small $\mathbf{o_p}$**: We write $\mathrm{x}_n = o_p(\mathrm{a}_n)$ if
$$\left|\frac{\mathrm{x}_n}{\mathrm{a}_n}\right| \xrightarrow{p} 0\,, \qquad \text{as} \quad n \to \infty\,,$$
that means for each $\varepsilon > 0$,
$$\mathbb{P}\left(\left|\frac{\mathrm{x}_n}{\mathrm{a}_n}\right| > \varepsilon\right) \to 0, \qquad \text{as} \quad n \to \infty\,.$$
2. **Big $\mathbf{O_p}$**: We write $\mathrm{x}_n = O_p(\mathrm{a}_n)$ if: For any $\varepsilon > 0$, there is an $M_\varepsilon \in \mathbb{R}^+$, such that
$$\limsup_{n\to\infty} \mathbb{P}\left(\left|\frac{\mathrm{x}_n}{\mathrm{a}_n}\right| > M_\varepsilon\right) \le \varepsilon\,.$$

**Proposition 4.A.1.** Let $\{\mathrm{x}_n \in \mathbb{R}\}_{n\in\mathbb{N}}$ and $\{\mathrm{y}_n \in \mathbb{R}\}_{n\in\mathbb{N}}$ be two sequences of random variables.

1. If $\mathrm{x}_n = o_p(1)$ and $\mathrm{y}_n = o_p(1)$, then $\mathrm{x}_n + \mathrm{y}_n = o_p(1)$.
2. If $\mathrm{x}_n = o_p(1)$ and $\mathrm{y}_n = O_p(1)$, then $\mathrm{x}_n + \mathrm{y}_n = O_p(1)$.
3. If $\mathrm{x}_n = o_p(1)$ and $\mathrm{y}_n = O_p(1)$, then $\mathrm{x}_n\mathrm{y}_n = o_p(1)$.

**Remark 4.A.1. Differences between Big O and Small o**: Roughly speaking:

1. If we say $\mathrm{x}_n = o_p(1)$, then $|\mathrm{x}_n|$ is likely small.
2. If we say $\mathrm{x}_n = O_p(1)$, then $|\mathrm{x}_n|$ is likely not large.
3. Intuitively, "small" $\Rightarrow$ "not large", but "not large" $\not\Rightarrow$ "small". Mathematically, we have
$$\mathrm{x}_n = o_p(1) \qquad \Rightarrow \qquad \mathrm{x}_n = O_p(1) \qquad \text{and} \qquad \mathrm{x}_n = O_p(1) \not\Rightarrow \qquad \mathrm{x}_n = o_p(1)\,.$$
4. If $\mathrm{x}_n = o_p(\mathrm{a}_n)$, then $\mathrm{x}_n/\mathrm{a}_n = o_p(1)$. So, $|\mathrm{x}_n|$ is much smaller than $\mathrm{a}_n$. Similar interpretation can be made for $\mathrm{x}_n = O_p(\mathrm{a}_n)$.

## 4.A.2 Deterministic Big O and Small o Notations

**Definition 4.A.2.** Let $\{x_n \in \mathbb{R}\}_{n\in\mathbb{N}}$ and $\{a_n \in \mathbb{R}^+\}$ be sequences of real numbers.

1. **Small o**: We write $x_n = o(a_n)$ if
$$\left|\frac{x_n}{a_n}\right| \to 0\,, \qquad \text{as} \quad n \to \infty\,.$$
2. **Big O**: We write $x_n = O(a_n)$ if: $\exists\, 0 \le m \le M < \infty$,
$$m \le \left|\frac{x_n}{a_n}\right| \le M\,, \qquad \text{as} \quad n \to \infty\,.$$

# *BIBLIOGRAPHY*


[1] Stehman, S. V. (1997). Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1), 77-89.

[2] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357.

[3] Tomek I. (1976). Two Modifications of CNN. IEEE Transactions on Systems, Man and Communications, SMC-6, 769-772

[4] Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics, (3), 408-421.

[5] Batista, G. E., Bazzan, A. L., and Monard, M. C. (2003, December). Balancing Training Data for Automated Annotation of Keywords: a Case Study. In WOB (pp. 10-18).

[6] Batista, G. E., Prati, R. C., and Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD explorations newsletter, 6(1), 20-29.

*Chapter 5*

# *REGRESSION LEARNING*

**CONTENTS**



## 5.1 Linear Regression

Linear Regression models the linear relationship between feature variables to response.

$$y = \omega_0 + \boldsymbol{x}\boldsymbol{\omega} + \varepsilon\,,$$

where $\varepsilon$ is the idiosyncratic noise.

With a collection of $N$ examples $\mathcal{S} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$,

### 5.1.1 Solution

In order to make good prediction, the errors $\{\varepsilon_n\}_{n=1}^N$ between the real labels $\{y_n\}_{n=1}^N$ and the predicted label $\{f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n)\}_{n=1}^N$ should be minimized. Note that these errors have both positive and negative values, summing up all the errors will cancel out some errors. Therefore, we need to use some functions to turn all errors to have same sign. Then, the methods are:

1. **Mean Squared Error** (MSE):
$$\frac{1}{N}\sum_{n=1}^{N}\varepsilon_n^2 := \frac{1}{N}\sum_{n=1}^{N}\{f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n) - y_n\}^2 .$$
2. **Mean Absolute Error** (MAE):
$$\frac{1}{N}\sum_{n=1}^{N}|\varepsilon_n| := \frac{1}{N}\sum_{n=1}^{N}|f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n) - y_n| .$$

**Remark 5.1.1. Objective Function**: The expression we minimize or maximize.

1. **Loss / Cost Function**: A measure of penalty for misclassification s.t. we minimize (MSE, MAE).
2. **MSE vs. MAE**: MAE is more useful nowadays in a sense that:
   (a) **MSE Overweights Outliers**: Outliers should be modelled differently, they should not make great influence to our model. Outliers have large errors s.t. $\varepsilon_n^2$ is far more greater than $|\varepsilon_n|$.
   (b) Although MAE does not have a continuous derivative, *i.e.* hard to find a closed form solution, we can use gradient descent method for finding the optimal solution.

## 5.2 Regularization

Regularization is an umbrella-term that encompasses methods that force the learning algorithm to build a less complex model. In practice, that often leads to slightly higher bias but significantly reduces the variance, which is the bias-variance tradeoff discussed in the above section.

The two most widely used types of regularization are called $\mathcal{L}^1$ and $\mathcal{L}^2$ regularization. The idea is: By introducing a penalizing term for complex model, we hope to obtain a simplier model without sacrificing much the model prediction power.

For simplicity, linear regression is used as an example. Reall the MSE of the linear regression is:

$$\min_{\boldsymbol{\omega},b} \frac{1}{N}\sum_{n=1}^{N}\{y_n - f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n)\}^2 .$$

The $\mathcal{L}^1$ and $\mathcal{L}^2$ regularized objective function are given by:

$$\begin{aligned}
\text{(LASSO) } \mathcal{L}^1 : \quad & \min_{\boldsymbol{\omega},b} \frac{1}{N}\sum_{n=1}^{N}\{y_n - f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n)\}^2 + C\sum_{d=1}^{D}|\omega_d| , \\
\text{(Ridge) } \mathcal{L}^2 : \quad & \min_{\boldsymbol{\omega},b} \frac{1}{N}\sum_{n=1}^{N}\{y_n - f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n)\}^2 + C\sum_{d=1}^{D}\{\omega_d\}^2 ,
\end{aligned} \tag{5.2.1}$$

where LASSO stands for Least Absolute Shrinkage and Selection Operator. Note that the intercept term $b$ is not included in the penality term to avoid the problem of multicollinearity. Moreover, similar to the $C$ in SVM (dealing with noise), this $C$ is also a hyperparameter that controls the importance of regularization:

1. If we set $C = 0$, the model becomes a standard non-regularized linear regression model.
2. If we set to $C$ to a high value, the learning algorithm will try to set most $\omega_d$ to a very small value or zero to minimize the objective, the model will become very simple which can lead to underfitting.

The goal is to find the value $C$ such that it doesn't increase the bias too much but reduces the variance to a reasonable level. To illustrate this idea, for simplicity, consider the ridge Regression ($\mathcal{L}^2$) without the intercept term, in practice, we include $b$ as $\omega^{(0)}$ while partitioning a column vector 1 inside $\boldsymbol{x}$

$$L = (\boldsymbol{y} - \boldsymbol{x}\boldsymbol{\omega})^\top(\boldsymbol{y} - \boldsymbol{x}\boldsymbol{\omega}) + \lambda\boldsymbol{\omega}^\top\boldsymbol{\omega}, \qquad \text{where} \quad \lambda = \frac{C}{N} .$$

Then, the derivative of the loss function with respect to $\boldsymbol{\omega}$ is:

$$\nabla_{\boldsymbol{\omega}} L = -2\boldsymbol{x}^\top\boldsymbol{y} + 2\boldsymbol{x}^\top\boldsymbol{x}\boldsymbol{\omega} + 2\lambda\boldsymbol{\omega} = 0 \qquad \Leftrightarrow \qquad \boldsymbol{\omega} = (\mathbf{x}^\top\mathbf{x} + \lambda\mathbf{I}_D)^{-1}\boldsymbol{x}^\top\boldsymbol{y} .$$

Hence, if $\lambda := \frac{C}{N}$ increases, $\boldsymbol{\omega}$ will decrease. $C$ can be regarded as a shrinkage penalty.

**Remark 5.2.1.** In general, we can pick $\lambda = \frac{C}{N^\alpha}$, where $\alpha > 0$. However,

1. if $\alpha$ is small, *i.e.* $\alpha < 1$, then $\lambda$ increases exponentially with $\alpha$. The effect of $\lambda\boldsymbol{I}_D$ to the parameter $\boldsymbol{\omega}$ dominates the effect of the training dataset $\boldsymbol{x}$ to the parameter $\boldsymbol{\omega}$, which becomes useless in generalization.
2. if $\alpha$ is huge, *i.e.* $\alpha > 1$, then $\lambda$ decays exponentially with $\alpha$. If the training dataset $\boldsymbol{x}$ suffers from multicollinearity, then $\boldsymbol{x}^\top\boldsymbol{x}$ is singular such that $(\boldsymbol{x}^\top\boldsymbol{x})^{-1}$ does not exist. Given small $\lambda$, the effect to

the singularity problem is small. However, we can desingularize $\boldsymbol{x}$ by the Pincipal Component Analysis (PCA) (See Section 5.3).

Therefore, $\alpha = 1$ seems to be the most suitable to the choice of $\lambda$, *i.e.* $\lambda = C/N$.

On the other hand, consider the LASSO Regression ($\mathcal{L}^1$) without the intercept term,

$$L = (\boldsymbol{y} - \boldsymbol{x}\boldsymbol{\omega})^\top (\boldsymbol{y} - \boldsymbol{x}\boldsymbol{\omega}) + \lambda \mathbf{1}_D |\boldsymbol{\omega}|, \qquad \text{where} \quad \mathbf{1}_D = (1, \cdots, 1), \quad \text{and} \quad |\boldsymbol{\omega}| = (|\omega_1|, \cdots, |\omega_D|)^\top .$$

However, we cannot solve the equation above through mathematic manipulation, instead, iterative optimization methods or quadratic programming have to be used, one of such method is called the cyclic coordinate descent algorithm.

### 5.2.1 Sparsity of $\mathcal{L}^1$ Regularization

1. $\mathcal{L}^1$ **Regularization**: Produce a sparse model, which is a model with a large hyperparameter $C$ that has most of parameters $\omega_d$ equal to zero, *i.e.* not important.
   Hence, $\mathcal{L}^1$ is good at **feature selection** by deciding which features are essential for prediction and which are not. That can be useful in case you want to increase model explainability.
2. $\mathcal{L}^2$ **Regularization**: Maximize the performance of the model and be differentiable such that gradient descent can be used for optimizing the objective function.

For simplicity, consider a model in Figure 5.2.1 with only two parameters, *i.e.* $\boldsymbol{\omega} = (\omega_1, \omega_2)$.

1. The red ellipse represents the contour of the MSE objective function: The middle black dot represents the non-constrainted least square estimate for $\boldsymbol{\omega}$ s.t. it has the lowest MSE. All the points on the same red line represents the same value of MSE.
2. The blue area represents the constraints regions for $\mathcal{L}^1$ (Left) regularization and $\mathcal{L}^2$ (Right) regularization, where
   (a) In $\mathcal{L}^1$ regularization, the contraint is $|\omega_1| + |\omega_2| \leq t$, *i.e.* the $\mathcal{L}^1$ regularization becomes
   $$\min_{\boldsymbol{\omega}, b} \frac{1}{N} \sum_{n=1}^{N} \{y_n - f_{\boldsymbol{\omega}, b}(\boldsymbol{x}_n)\}^2 , \qquad \text{subject to} \quad \sum_{d=1}^{2} |\omega_d| < t . \tag{5.2.2}$$
   (b) In $\mathcal{L}^2$ regularization, the contraint is $(\omega_1)^2 + (\omega_2)^2 \leq t^2$, *i.e.* the $\mathcal{L}^2$ regularization becomes
   $$\min_{\boldsymbol{\omega}, b} \frac{1}{N} \sum_{n=1}^{N} \{y_n - f_{\boldsymbol{\omega}, b}(\boldsymbol{x}_n)\}^2 , \qquad \text{subject to} \quad \sum_{d=1}^{2} (\omega_d)^2 < t . \tag{5.2.3}$$
   Note that the original $\mathcal{L}^1$ and $\mathcal{L}^2$ problem (in (5.2.1)) can be reconstructed into above constraints problem. The former is called the Tikhonov regularization and the later is called Ivanov regularization. They are equivalent under most scenarios. However, the proof is not provided here, but one might notice that it is similar to the primal and dual problem in SVM.

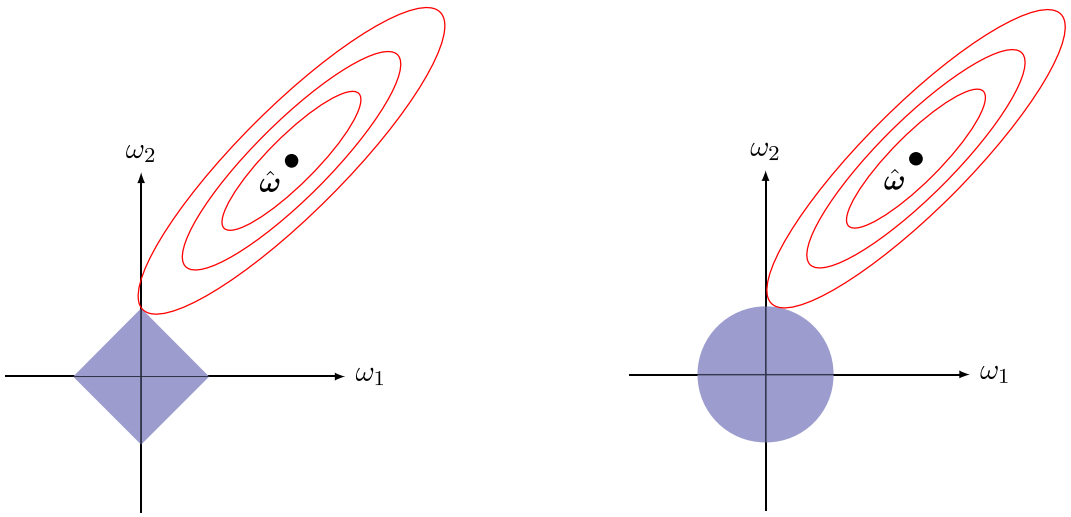



Figure 5.2.1: Left: $\mathcal{L}^1$ regularization; and Right: $\mathcal{L}^2$ regularization

The solutions are the contact point where the red ellipse contour meets the blue area, because the red line which is further from the center black dot has a greater MSE. Note that

1. the blue area for $\mathcal{L}^1$ regularization is a diamond, the solution is more likely to be the four corners of the diamond. In other words, at the four corners, there are infinitely many possible tangent lines, *i.e.* in this figure, we have $\omega_2 > 0$ but $\omega_1 = 0$ which shrinks $\omega_1$ to zero.
2. the blue area for $\mathcal{L}^2$ regularization is a circle, the solution is the point at the boundary of the circle, which is seldom set at which neither $\omega_1$ nor $\omega_2$ is exactly zero.

**Remark 5.2.2.** Some datasets with MSE as loss function may produce a contour which shrinks the parameter $\omega_d$ to zero, even in $\mathcal{L}^2$ regularization. However, Figure 5.2.1 shows the general case.

**Remark 5.2.3. Elastic Net Regularization** combines $\mathcal{L}^1$ and $\mathcal{L}^2$ regularization. For example, in photo detection, $\mathcal{L}^1$ regularization helps to capture the edges and shapes, because $\mathcal{L}^1$ regularization shrinks the unimportant parameters to exact zero. However, the RGB (Red Green Blue) color becomes crucial to the photo detection, in which $\mathcal{L}^1$ regularization fails to capture the gradient of the color. However, $\mathcal{L}^2$ regularization works well, because it seldom shrinks the parameters to exact zero.

### 5.2.2 Cyclic Coordinate Descent Algorithm

The algorithm is as follow: Repeat the following steps until convergence:

**Algorithm 5.3** Cyclic Coordinate Descent Algorithm

**while** Converge **do**
  Fix $\omega_1, \omega_2, \cdots, \omega_D$ and update $b$;
  Fix $b, \omega_2, \cdots, \omega_D$ and update $\omega_1$;

  $\vdots$

  Fix $b, \omega_1, \cdots, \omega_{D-1}$ and update $\omega_D$.
**end while**

Consider the update with $\boldsymbol{\omega}$: Denote $\tilde{\omega}_d$ be the fixed terms and $\omega_\ell$ is for the update. The objective function becomes:

$$\begin{aligned} L &= \sum_{n=1}^{N}\left(y_n - \tilde{\omega}_0 - \sum_{d\neq\ell} x_n^{(d)}\tilde{\omega}_d - x_n^{(\ell)}\omega_\ell\right)^2 + \lambda\sum_{d\neq\ell}|\tilde{\omega}_d| + \lambda|\omega_\ell| \\ &= \sum_{n=1}^{N}\left(r_n^{(\ell)} - x_n^{(\ell)}\omega_\ell\right)^2 + \lambda|\omega_\ell| + C \\ &= \sum_{n=1}^{N}\left(x_n^{(\ell)}\right)^2(\omega_\ell)^2 - 2\sum_{n=1}^{N} r_n^{(\ell)}x_n^{(\ell)}\omega_\ell + \lambda|\omega_\ell| + C' =: a\,(\omega_\ell)^2 - 2b\omega_\ell + \lambda|\omega_\ell| + C', \end{aligned}$$

where $C$, $C'$, and $r_n^{(\ell)}$ does not depend on $\omega_\ell$ and note that:

$$r_n^{(\ell)} = y_n - \tilde{\omega}^{(0)} - \sum_{d\neq\ell} x_n^{(d)}\tilde{\omega}_d, \qquad a = \sum_{n=1}^{N}\left(x_n^{(\ell)}\right)^2 > 0, \qquad \text{and} \qquad b = \sum_{n=1}^{N} r_n^{(\ell)}x_n^{(\ell)}.$$

The problem becomes minimizing the objective function with $\omega_\ell$,

$$\min_{\omega_\ell} a\,(\omega_\ell)^2 - 2b\omega_\ell + \lambda|\omega_\ell| + C' \qquad \Leftrightarrow \qquad \min_{\omega_\ell} (\omega_\ell)^2 - \frac{2b\omega_\ell - \lambda|\omega_\ell|}{a} \qquad (\because a > 0).$$

1. If $\omega_\ell > 0$, then
$$\hat{\omega}_\ell = \arg\min_{\omega_\ell} (\omega_\ell)^2 - \frac{2b\omega_\ell - \lambda\omega_\ell}{a} \quad \Leftrightarrow \quad \hat{\omega}_\ell = \arg\min_{\omega_\ell}\left(\omega_\ell - \frac{2b-\lambda}{2a}\right)^2 = \frac{2b-\lambda}{2a} > 0.$$
Hence, we need extra condition that $2b - \lambda > 0$ as $a > 0$.

2. If $\omega_\ell < 0$, then
$$\hat{\omega}_\ell = \arg\min_{\omega_\ell} (\omega_\ell)^2 - \frac{2b\omega_\ell + \lambda\omega_\ell}{a} \quad \Leftrightarrow \quad \hat{\omega}_\ell = \arg\min_{\omega_\ell}\left(\omega_\ell - \frac{2b+\lambda}{2a}\right)^2 = \frac{2b+\lambda}{2a} < 0.$$
Hence, we need extra condition that $2b + \lambda < 0$ as $a > 0$.

3. If $-\frac{\lambda}{2} \leq b \leq \frac{\lambda}{2}$, then consider the objective function when

   (a) $(\omega_\ell = 0)$:
   $$L = a(0)^2 - 2b \times 0 + \lambda|0| + C' = C' > 0.$$

   (b) $(\omega_\ell > 0)$:
   $$L = \underbrace{a\left(\omega_\ell - \frac{2b-\lambda}{2a}\right)^2 + D}_{>0} + C' > C'.$$

   (c) $(\omega_\ell < 0)$:
   $$L = \underbrace{a\left(\omega_\ell - \frac{2b+\lambda}{2a}\right)^2 + E}_{>0} + C' > C'.$$

   Hence, we conclude that $\omega_\ell = 0$ is the minimizer.

$$\tilde{\omega}_\ell = \begin{cases} \frac{2b-\lambda}{2a}, & \text{if } b > \frac{\lambda}{2} > 0 \\ 0, & \text{if } -\frac{\lambda}{2} \leq b \leq \frac{\lambda}{2} \\ \frac{2b+\lambda}{2a}, & \text{if } b < -\frac{\lambda}{2} < 0 \end{cases}.$$

Consider the update for $b$: The objective function:

$$L = \sum_{n=1}^{N}(y_n - \boldsymbol{x}_n\boldsymbol{\omega} - b)^2 + \lambda\mathbf{1}_D^\top|\boldsymbol{\omega}|,$$

where $\mathbf{1}_D^\top = (1, \cdots, 1) \in \mathbb{R}^{1\times D}$. The partial derivative with respect to $b$ gives:

$$\frac{\partial L}{\partial b} = -2\sum_{n=1}^{N}(y_n - \boldsymbol{x}_n\boldsymbol{\omega} - b) = 0 \qquad \Leftrightarrow \qquad b = \frac{1}{N}\sum_{n=1}^{N}(y_n - \boldsymbol{x}_n\boldsymbol{\omega}) =: \frac{1}{N}\sum_{n=1}^{N} r_n^{(0)}\,.$$

### 5.2.3 Different Contour

Different $\mathcal{L}^q$ will produce different contours,

$$\mathcal{L}^q: \quad \min_{\boldsymbol{\omega},b}\frac{1}{N}\sum_{n=1}^{N}\{y_n - f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n)\}^2 + C\sum_{d=1}^{D}|\omega_d|^q\,.$$

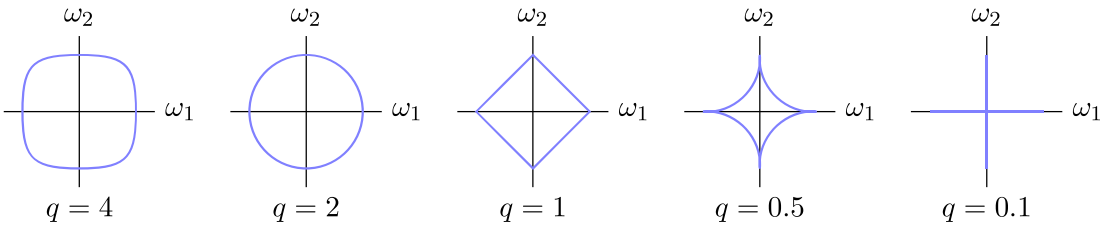



Figure 5.2.2: Contours with different value of $q$

Values of $q \in (1, 2)$ suggest a compromise between the LASSO and ridge regression. In particular, in Remark 5.2.3, the elasticnet has penalty

$$\lambda\sum_{d=1}^{D}\Big\{\alpha\,(\omega_d)^2 + (1-\alpha)|\omega_d|\Big\}, \qquad \alpha \in (0, 1)\,.$$

### 5.2.4 Quadratic Programming in LASSO

In LASSO, from (5.2.2), we have

$$\min_{\boldsymbol{\omega},b}\frac{1}{N}\sum_{n=1}^{N}\{y_n - f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n)\}^2\,, \qquad \text{subject to} \quad \sum_{d=1}^{D}|\omega_d| < t\,.$$

For simplicity, assume it is a linear model, *i.e.* $f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n) = b + \boldsymbol{x}_n\boldsymbol{\omega}$, then

$$\min_{\boldsymbol{\omega},b}\frac{1}{N}\sum_{n=1}^{N}\{y_n - f_{\boldsymbol{\omega},b}(\boldsymbol{x}_n)\}^2 \qquad \Leftrightarrow \qquad \min_{\boldsymbol{\omega},b}\{\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega} - \mathbf{1}_N \times b\}^\top\{\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega} - \mathbf{1}_N \times b\}\,.$$

For simplicity, we augment $b$ into $\boldsymbol{\omega}$ and $\mathbf{1}_N$ into $\boldsymbol{X}$ such that the problem simplifies into:

$$\min_{\boldsymbol{\omega}}\{\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega}\}^\top\{\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\omega}\} \qquad \Leftrightarrow \qquad \min_{\boldsymbol{\omega}}\boldsymbol{\omega}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\omega} - 2\boldsymbol{y}^\top\boldsymbol{X}\boldsymbol{\omega}\,.$$

Further dividing by 2, the problem becomes

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\boldsymbol{\omega}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\omega} - \boldsymbol{y}^\top\boldsymbol{X}\boldsymbol{\omega}\,,\\ &\text{subject to} && \mathbf{1}_D^\top|\boldsymbol{\omega}| \le t\,, \end{aligned}$$

where $\mathbf{1}_D^\top = (1, \cdots, 1) \in \mathbb{R}^{1\times D}$. However, the absolute constraint is troublesome. We can memic the strategy in solving the problem of nonconvex domain in Quadratic Programming in (3.5.6) by introducing $\boldsymbol{\omega}^+$ and $\boldsymbol{\omega}^-$:

$$\omega_d^+ = \frac{|\omega_d| + \omega_d}{2} \ge 0 \qquad \text{and} \qquad \omega_d^- = \frac{|\omega_d| - \omega_d}{2} \ge 0\,,$$

such that $\omega_d^+ \cdot \omega_d^- = 0$. Therefore, in matrix form, we obtain

$$\boldsymbol{\omega} := \boldsymbol{\omega}^+ - \boldsymbol{\omega}^- \qquad \text{and} \qquad |\boldsymbol{\omega}| := \boldsymbol{\omega}^+ + \boldsymbol{\omega}^-, \qquad \boldsymbol{\omega}^+ \succeq \mathbf{0}, \quad \boldsymbol{\omega}^- \succeq \mathbf{0}, \quad \boldsymbol{\omega}^+ \odot \boldsymbol{\omega}^- = \mathbf{0}.$$

The problem becomes

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}(\boldsymbol{\omega}^+ - \boldsymbol{\omega}^-)^\top \boldsymbol{X}^\top \boldsymbol{X}(\boldsymbol{\omega}^+ - \boldsymbol{\omega}^-) - \boldsymbol{y}^\top \boldsymbol{X}(\boldsymbol{\omega}^+ - \boldsymbol{\omega}^-), \\ \text{subject to} \quad & \text{i)} \quad \mathbf{1}_D^\top(\boldsymbol{\omega}^+ + \boldsymbol{\omega}^-) \leq t; \\ & \text{ii)} \quad \boldsymbol{\omega}^+ \succeq \mathbf{0}; \\ & \text{iii)} \quad \boldsymbol{\omega}^- \succeq \mathbf{0}; \\ & \text{iv)} \quad \boldsymbol{\omega}^+ \odot \boldsymbol{\omega}^- = \mathbf{0}. \end{aligned}$$

Or equivalently,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\begin{pmatrix} \boldsymbol{\omega}^+ \\ \boldsymbol{\omega}^- \end{pmatrix}^\top \begin{pmatrix} \boldsymbol{X}^\top\boldsymbol{X} & -\boldsymbol{X}^\top\boldsymbol{X} \\ -\boldsymbol{X}^\top\boldsymbol{X} & \boldsymbol{X}^\top\boldsymbol{X} \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega}^+ \\ \boldsymbol{\omega}^- \end{pmatrix} - \begin{pmatrix} \boldsymbol{y}^\top\boldsymbol{X}, & -\boldsymbol{y}^\top\boldsymbol{X} \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega}^+ \\ \boldsymbol{\omega}^- \end{pmatrix}, \\ \text{subject to} \quad & \text{i)} \quad \begin{pmatrix} \mathbf{1}_D^\top, & \mathbf{1}_D^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega}^+ \\ \boldsymbol{\omega}^- \end{pmatrix} \leq t; \\ & \text{ii)} \quad \boldsymbol{\omega}^+ \succeq \mathbf{0}; \\ & \text{iii)} \quad \boldsymbol{\omega}^- \succeq \mathbf{0}; \\ & \text{iv)} \quad \boldsymbol{\omega}^+ \odot \boldsymbol{\omega}^- = \mathbf{0}. \end{aligned}$$

However, we need to ensure $\boldsymbol{P} = \boldsymbol{X}^\top\boldsymbol{X}$ is positive definite and such problem is still not in the standard form of Quadratic Programming problem. As mentioned in Example 3.5.2, if $\boldsymbol{\omega}$ is of small dimension, then we shall construct $2^D$ number of Quadratic Programming problems. Moreover, in a simple case where $\boldsymbol{\omega} = (\omega_1, \omega_2)^\top$, the constraint $\mathbf{1}_D^\top|\boldsymbol{\omega}| \leq t$ with $K = 2$ can be written as:

$$\boldsymbol{G}|\hat{\boldsymbol{\omega}}| = \begin{pmatrix} -1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \end{pmatrix} \leq \begin{pmatrix} t \\ t \\ t \\ t \end{pmatrix}.$$

such that we do no need to decompose $\boldsymbol{\omega}$ into $\boldsymbol{\omega}^+$ and $\boldsymbol{\omega}^-$.

## 5.3 Principal Component Regression (a.k.a Spectral Regression)

As a key step in the whole precedure of Principle Component Regression (PCR), Principal Component Analysis (PCA) (See Section 3.4) is a classical multivariate statistical technique for reducing the linear effective dimension of a collection of feature variables, that means to find linear combinations of original variables so that the information contained in the original data is "essentially" preserved after the dimensional reduction.

Given a dataset $\mathcal{S} = \{(\boldsymbol{x}_n, y_n) : \boldsymbol{x}_n \in \mathbb{R}^D \text{ and } y_n \in \mathbb{R}\}_{n=1}^N$, Principal Component Regression (PCR) is to predict $y_n$ while we simultaneously carry out dimension reduction on the collection of input covariates $(x_{n1}, x_{n2}, \cdots, x_{nD})$. As a direct consequence, one advantage of using PCR is to remove the multicollinearity in the set of input covariates, which arises when two or more of these $x_{nd}$'s are close to being collinear. Generally, rather than just considering the linear structure, we suppose that there is a functional relation among $y$ depending on the components of $\boldsymbol{x}$ as shown below:

$$y_n = f(\boldsymbol{x}_n) + \tilde{\varepsilon}_n, \qquad n = 1, 2, \cdots, N\,,$$

where $\boldsymbol{f} : \mathbb{R}^D \mapsto \mathbb{R}$ is an unknown function to be estimated [1] and $\varepsilon_n \in \mathbb{R}$'s are iid random noise terms; and we further assume the statistical independence of $\boldsymbol{x}_n$ and $\varepsilon_n$ for the simplicity of derivation. To carry on, we choose a set of basis functions $\{\tilde{f}_m(\cdot)\}_{m=1}^M$, where each $\tilde{f}_m$ is regarded as a relatively more abstract feature element, and we aim to find a linear combination of these feature elements to approximate the unknown function $f$ in a pointwise manner. Hence, we take a step back by allowing some minor additional errors, so that all of these combine with $\tilde{\varepsilon}_n$ to form $\varepsilon_n$ and now try to regress $y_n$ in the following form:

$$y_n = \sum_{m=1}^{M} \alpha_m \tilde{f}_m(\boldsymbol{x}_n) + \varepsilon_n, \qquad n = 1, \cdots, N\,, \tag{5.3.1}$$

where $\alpha_m$, to be determined in the following discussion, is the coefficient serving as weights assigned to $\tilde{f}_m(\cdot)$. Here, we regard $\tilde{\boldsymbol{f}}(\boldsymbol{x}) = (\tilde{f}_1(\boldsymbol{x}), \cdots, \tilde{f}_M(\boldsymbol{x}))$ as a $K$-dimensional random vector. Since $M$ is mostly large, we next use a PCA approach to these $\tilde{f}_m(\boldsymbol{x}_n)$ to reduce the feature element dimension, and as a byproduct to also remove the multicollinearity, if any. With the sample of size $N$ of $\boldsymbol{x}_n$'s, we consider the following data matrix:

$$\tilde{\boldsymbol{D}} = \begin{pmatrix} \tilde{\boldsymbol{f}}(\boldsymbol{x}_1) & \cdots & \tilde{\boldsymbol{f}}(\boldsymbol{x}_N) \end{pmatrix}, \qquad \text{where} \quad \tilde{\boldsymbol{f}}(\boldsymbol{x}_n) := \begin{pmatrix} \tilde{f}_1(\boldsymbol{x}_n) & \cdots & \tilde{f}_M(\boldsymbol{x}_n) \end{pmatrix}^\top,$$

with which the corresponding $M \times M$ sample covariance matrix $\tilde{\boldsymbol{S}}$ is constructed. PCR means to apply PCA to extract principal components for this $\tilde{\boldsymbol{S}}$, among these $\tilde{f}_m(\boldsymbol{x})$'s, as the new covariates to construct a regression model for $y$. In particular, with the spectral decomposition of $\tilde{\boldsymbol{S}} = \boldsymbol{H}\boldsymbol{D}\boldsymbol{H}^\top$ such that $\boldsymbol{H} = (\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_M)$, where $\boldsymbol{h}_j = (h_{1j}, h_{2j}, \cdots, h_{Mj})^\top$ is the unit eigenvector corresponding to the $j$-th largest eigenvalue $\lambda_j$ of $\tilde{\boldsymbol{S}}$, and the corresponding $j$-th principal component of $\boldsymbol{x}_n$, *i.e.* $P_m(\boldsymbol{x}_n)$ for $m = 1, 2, \cdots, M$

[1] Usually $f$ is highly nontrivial, and one of the most useful methods to determine which class of function $f$ belongs to is to get a first feeling from the rough sketch.

can be computed by:

$$\begin{pmatrix} P_1(\boldsymbol{x}_n) \\ P_2(\boldsymbol{x}_n) \\ \vdots \\ P_M(\boldsymbol{x}_n) \end{pmatrix} = \boldsymbol{H}^\top \begin{pmatrix} \tilde{f}_1(\boldsymbol{x}_n) \\ \tilde{f}_2(\boldsymbol{x}_n) \\ \vdots \\ \tilde{f}_M(\boldsymbol{x}_n) \end{pmatrix} = \begin{pmatrix} \boldsymbol{h}_1^\top \\ \boldsymbol{h}_2^\top \\ \vdots \\ \boldsymbol{h}_M^\top \end{pmatrix} \begin{pmatrix} \tilde{f}_1(\boldsymbol{x}_n) \\ \tilde{f}_2(\boldsymbol{x}_n) \\ \vdots \\ \tilde{f}_M(\boldsymbol{x}_n) \end{pmatrix},$$

which implies that $P_j(\boldsymbol{x}_n) = \sum_{m=1}^M h_{mj}\tilde{f}_m(\boldsymbol{x}_n)$. By inverting $\boldsymbol{H}^\top$, we can write $\tilde{f}_j(\boldsymbol{x}_n) = \sum_{m=1}^M h_{jm}P_m(\boldsymbol{x}_n)$, therefore

$$\begin{aligned} y_n &= \sum_{m=1}^M \alpha_m \tilde{f}_m(\boldsymbol{x}_n) + \varepsilon_n = \sum_{m=1}^M \alpha_m \sum_{j=1}^M h_{mj}P_j(\boldsymbol{x}_n) + \varepsilon_n \\ &= \sum_{m=1}^M \sum_{j=1}^M \alpha_m h_{mj} P_j(\boldsymbol{x}_n) + \varepsilon_n = \sum_{j=1}^M \beta_j P_j(\boldsymbol{x}_n) + \varepsilon_n \,, \end{aligned}$$

where $\beta_j := \sum_{m=1}^M \alpha_m h_{mj}$, more neatly $\boldsymbol{\alpha}^\top \boldsymbol{H} = (\beta_1, \cdots, \beta_M) =: \boldsymbol{\beta}^\top$. To achieve the dimension reduction of the problem, we suppose that the first $K$ ($\ll M$) principal components explain nearly all information, let say 99 percent of total $\boldsymbol{\lambda}$, *i.e.* $\sum_{k=1}^K \lambda_k \geq (0.99)\sum_{m=1}^M \lambda_m$, then we believe that these first $K$ number of PCs are crucial while the dependence of the remaining $M-K$ ones is more negligible, that means the unknown coefficients $\beta_j$ for $j = K+1, \cdots, M$ are all $|\beta_j| \leq \varepsilon$ for some sufficiently small $\varepsilon > 0$. Now,

$$\alpha_m = \sum_{j=1}^M h_{mj}\beta_j = \sum_{j=1}^K h_{mj}\beta_j + \sum_{j=K+1}^M h_{mj}\beta_j \,,$$

where the last summation is negligible since

$$\left| \sum_{j=K+1}^M h_{mj}\beta_j \right| \leq \max_{M \geq j \geq K+1} |\beta_j| \cdot \|\boldsymbol{h}_{m\cdot}\|_2 \leq \varepsilon \,,$$

since $\|\boldsymbol{h}_{m\cdot}\|_2 = 1$ by the orthogonality of $\boldsymbol{H}$. Therefore, we can simply approximate

$$\alpha_m \approx \sum_{k=1}^K h_{mk}\beta_k =: \alpha_m^{(K)} \,.$$

Alternatively, if one cannot warrant the smallness of $|\beta_j|$ for $j \geq K+1$, yet we still can sense the uniform boundedness of $|\beta_m|$ for all $m$, let say $\sup_m |\beta_m| \leq \mathcal{M}$, for a nice enough model; since

$$\sum_{m=1}^M \alpha_m^{(K)} \tilde{f}_m(\boldsymbol{x}_n) = \sum_{m=1}^M \sum_{k=1}^K h_{mk}\beta_k \tilde{f}_m(\boldsymbol{x}_n) = \sum_{k=1}^K \beta_k \sum_{m=1}^M h_{mk}\tilde{f}_m(\boldsymbol{x}_n) = \sum_{k=1}^K \beta_k P_k(\boldsymbol{x}_n) \,,$$

then the (remaining) variance of the residual term by using only the truncated coefficients $\alpha_m^{(K)}$'s is:

$$\begin{aligned} \mathrm{Var}\left( \hat{y}_n - \sum_{m=1}^M \alpha_m^{(K)} \tilde{f}_m(\boldsymbol{x}_n) \right) &= \mathrm{Var}\left( \hat{y}_n - \sum_{k=1}^K \beta_k P_k(\boldsymbol{x}_n) \right) = \mathrm{Var}\left( \sum_{m=K+1}^M \beta_m P_m(\boldsymbol{x}_n) \right) \\ &= \sum_{m=K+1}^M \beta_m^2 \mathrm{Var}(P_j(\boldsymbol{x}_n)) \leq \mathcal{M}^2 \sum_{m=K+1}^M \mathrm{Var}\left(P_m(\boldsymbol{x}_n)\right) \leq \mathcal{M}^2 \delta \sum_{m=1}^M \lambda_m \,, \end{aligned}$$

where $\delta$ is chosen to be sufficiently small, say $0.01/\mathcal{M}^2$, based on which we adjust the number $K$ of PCs to be taken.

Finally, we can work backward as follows:

1. Firstly, regress $y_n$ against $P_1, \cdots, P_K$ functions of $\boldsymbol{x}_n$ by establishing the least square estimator for

the corresponding $\boldsymbol{\beta}$ coefficients with the new $M \times K$ data matrix $\boldsymbol{P}^{(K)} = (P_1(\boldsymbol{x}_n), \cdots, P_K(\boldsymbol{x}_n))$, *i.e.*

$$\hat{\boldsymbol{\beta}} = \begin{pmatrix} \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_K \end{pmatrix} = \left[ \left( \boldsymbol{P}^{(K)} \right)^\top \boldsymbol{P}^{(K)} \right]^{-1} \left( \boldsymbol{P}^{(K)} \right)^\top \boldsymbol{y} \qquad \Rightarrow \qquad \hat{f}(\boldsymbol{x}) = \sum_{k=1}^{K} \hat{\beta}_k P_k(\boldsymbol{x}) \, .$$

2. Secondly, we set $\hat{\alpha}_m^{(K)} = \sum_{k=1}^{K} h_{mk} \hat{\beta}_k$, and obtain the final regressor

$$\hat{y}_n = \sum_{m=1}^{M} \hat{\alpha}_m^{(K)} \tilde{f}_m(\boldsymbol{x}_n) \, .$$

As for the diagnosis, we somehow consider the dropped miscellaneous term $\sum_{m=K+1}^{M} \beta_m P_m(\boldsymbol{x}_n)$ to merge with the idiosyncratic noise term $\varepsilon_n$, that means they add up to become another error term $\varepsilon'_n$, but as long as $\varepsilon_n$ is independent of $\boldsymbol{x}_n$, $\varepsilon_n$ is independent of $P_k(\boldsymbol{x}_n)$ for $k = 1, \cdots, K$, therefore $\varepsilon'_n$ is also independent of $P_k(\boldsymbol{x}_n)$ for $k = 1, \cdots, K$, then the same theory, namely behind the diagnosis of the significance of various coefficients, can still be applied as that of multiple linear regression diagnosis techniques.

Next, we shall illustrate PCR in **R**:

```
> pcreg <- function(PC, X, y, scale=TRUE, center=TRUE){
+     N <- dim(X)[1]
+     D <- dim(X)[2]
+     # Centering: subtract mean
+     ones = rep(1, N)
+     if (center){
+         MEAN <- colMeans(X)
+         X <- X - ones%*%t(MEAN)
+     }else{
+         MEAN <- rep(0, D)
+     }
+     if (scale) {
+         SD <- sqrt(apply(X, 2, var))
+         X <- X%*%diag(1/SD)
+     }else{
+         SD <- diag(D)
+     }
+     PC_proj <- X%*%PC
+     df_result <- matrix(ncol=3, nrow=D+1)
+     colnames(df_result) <- c("RMSE", "MSE", "R2")
+     rownames(df_result) <- c("(Intercept)", paste(1:D, "comps"))
+     MODEL <- c()
+     for (d in 0:D){
+         if (d==0){
+             data <- as.data.frame(y)
+             model <- lm(y~1, data=data)
+         }else{
+             data <- as.data.frame(cbind(PC_proj[,1:d], y))
```

```
+             names(data) <- c(names(data)[-length(names(data))], "y")
+             model <- lm(y~., data=data)
+         }
+         r <- model$residuals
+         RMSE <- sqrt(mean(r^2))
+         MSE <- mean(r^2)
+         R2 <- summary(model)$r.squared
+         df_result[(d+1),] <- c(RMSE, MSE, R2)
+         MODEL <- c(MODEL, list(model))
+     }
+     return (list(RMSE=df_result[,1], MSE=df_result[,2], R2=df_result[,3],
+                  MODEL=MODEL, center=MEAN, scale=SD))
+ }
```

Programme 5.3.1: `pcreg()` function for PCR in **R**.

Here the `pcreg()` function accepts five parameters, namely the `PC` parameter for the principle components (loadings), the `X` parameter for the feature vector, the `y` parameter for the label, `center` and `scale` are the boolean parameters for the indication of whether we subtract the mean and divide with standard deviation for each feature variables respectively. The function returns a list: `RMSE`, the root mean squared error; `MSE`, the mean squared error; `R2`, the coefficient of determination; `MODEL`, the list of `lm` objects for all the $D+1$ number of linear models; `center`, the vector consisting the mean of each feature variable, if we have the argument `center=TRUE`, or the zero vector in $\mathbb{R}^D$ otherwise; and `scale`, the vector consisting the standard deviation of each feature variable, if we have the argument `scale=TRUE`, or the one vector in $\mathbb{R}^D$ otherwise.

The RMSE, MSE, and $R^2$ help us to determine the number of principle components used in the regression. We therefore introducing the Boston dataset provided by the `MASS` library as an illustration. The Boston dataset is collected by the U.S. Census Service and is orginally published by Harrison and Rubinfeld (1978) which contains 13 feature variables to predict the median value (in $1000 unit) of owner-occupied homes in Boston. We first split the dataset into the training set and the test set in a ratio of 7:3 with the initial seed number `4012`:

```
> library(MASS)
> set.seed(4012)
> options(digits=5)
>
> train_ind <- sample(1:nrow(Boston), round(0.7*nrow(Boston)))
> d_train <- Boston[train_ind,]
> d_test <- Boston[-train_ind,]
> X_train <- as.matrix(d_train[-ncol(d_train)])
> y_train <- as.matrix(d_train[ncol(d_train)])
> X_test <- as.matrix(d_test[-ncol(d_test)])
> y_test <- as.matrix(d_test[ncol(d_test)])
>
```

```
> pca <- princomp(X_train, cor=TRUE)
> pcr_result <- pcreg(pca$loadings, X_train, y_train, scale=TRUE, center=TRUE)
>
> library(pls)
> pcr_model <- pcr(medv~., data=d_train, scale=T, center=T, validation="none")
>
> pcr_result$RMSE
(Intercept)      1 comps      2 comps      3 comps      4 comps      5 comps
     9.5506       7.7831       6.5231       5.6532       5.6301       5.2036
     6 comps      7 comps      8 comps      9 comps     10 comps     11 comps
     5.2017       5.1812       5.1503       5.1421       5.1264       5.0596
    12 comps     13 comps
     4.9361       4.8961
> RMSEP(pcr_model)
(Intercept)      1 comps      2 comps      3 comps      4 comps      5 comps
      9.551        7.783        6.523        5.653        5.630        5.204
     6 comps      7 comps      8 comps      9 comps     10 comps     11 comps
      5.202        5.181        5.150        5.142        5.126        5.060
    12 comps     13 comps
      4.936        4.896
>
> pcr_result$MSE
(Intercept)      1 comps      2 comps      3 comps      4 comps      5 comps
     91.215       60.576       42.551       31.958       31.698       27.078
     6 comps      7 comps      8 comps      9 comps     10 comps     11 comps
     27.058       26.844       26.525       26.441       26.280       25.599
    12 comps     13 comps
     24.365       23.972
> MSEP(pcr_model)
(Intercept)      1 comps      2 comps      3 comps      4 comps      5 comps
      91.21        60.58        42.55        31.96        31.70        27.08
     6 comps      7 comps      8 comps      9 comps     10 comps     11 comps
      27.06        26.84        26.53        26.44        26.28        25.60
    12 comps     13 comps
      24.36        23.97
>
> pcr_result$R2
(Intercept)      1 comps      2 comps      3 comps      4 comps      5 comps
    0.00000      0.33589      0.53351      0.64964      0.65249      0.70314
     6 comps      7 comps      8 comps      9 comps     10 comps     11 comps
    0.70336      0.70570      0.70920      0.71012      0.71189      0.71935
    12 comps     13 comps
    0.73288      0.73719
> R2(pcr_model)
```

```
(Intercept)       1 comps       2 comps       3 comps       4 comps       5 comps
     0.0000        0.3359        0.5335        0.6496        0.6525        0.7031
     6 comps       7 comps       8 comps       9 comps      10 comps      11 comps
     0.7034        0.7057        0.7092        0.7101        0.7119        0.7194
    12 comps      13 comps
     0.7329        0.7372
>
> par(mfrow=c(1,3), mar=c(4,4,2,2))
> plot(0:ncol(X_train), pcr_result$RMSE, type="l", main="medv",
+       xlab="number of components", ylab="RMSE")
> validationplot(pcr_model, val.type="MSE")
> validationplot(pcr_model, val.type="R2")
```

Programme 5.3.2: PCR using Programme 5.3.1 and `pcreg()` from `pls` library with Boston dataset in **R**.

Here the `pcr()` function from the `pls` library peforms the Principle Component Regression as a comparison for the self-defined function `pcreg()`. The input of the `pcr()` function is similar to that of the `lm()`, where the first parameter accepts the `formula` object, while we also have the two boolean parameters `center` and `scale`. In this Boston example, we do not use the cross-validation, because we want to compare this function with our self-defined function `pcreg()`. The plots of the number of components against RMSE, MSE, and $R^2$ are shown in Figure 5.3.1.

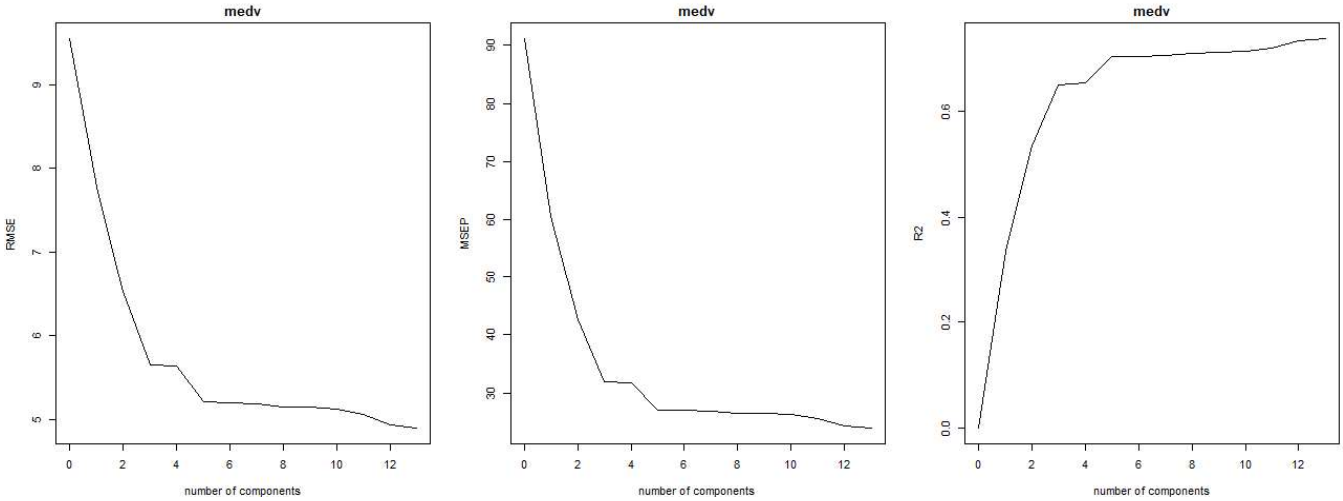



Figure 5.3.1: Plots for RMSE (Left); MSE (Middle); and $R^2$ (Right).

Finally, we make the prediction with the test set:

```
> pcreg_pred <- function(PC, PCR_model, X_test, ncomp){
+     N <- dim(X_test)[1]
+     ones <-  rep(1, N)
+     X_test <- X_test - ones%*%t(PCR_model$center)
+     X_test <- X_test%*%diag(1/PCR_model$scale)
+
+     model <- PCR_model$MODEL[ncomp+1][[1]]
```

```
+     df <- as.data.frame((X_test%*%PC)[,1:ncomp])
+     names(df) <- names(coef(model))[2:(ncomp+1)]
+     return (predict(model, newdata=df))
+ }
>
> y_hat_pcr <- predict(pcr_model, X_test, ncomp=5)
> y_hat_pcreg <- pcreg_pred(pca$loadings, pcr_result, X_test, ncomp=5)
> all.equal(array(y_hat_pcr), array(y_hat_pcreg))
[1] TRUE
> mean((y_test - y_hat_pcreg)^2)
[1] 20.756
```

Programme 5.3.3: Prediction of Boston test set in **R**.

Here, we use the means and the standard deviations from the training set to standardize the test set. Finally, both predictions of the test data using the `pls` library and the self-defined `pcreg_pred()` are the same, while the MSE of the test set is 20.756. However, one limitation is that we might not be able to explain the coefficients of the principle component regression model.

## 5.4 Kernel Regression

Usually in regression analysis, we are faced with the task of predicting the label vector $\boldsymbol{y} \in \mathbb{R}^Q$ by using the associated vector of feature variables $\boldsymbol{x} \in \mathbb{R}^D$, and one of the key quantities of interest in such prediction is the conditional mean of $\mathbb{E}(\mathbf{y}|\mathbf{x} = \boldsymbol{x})$. Throughout this section, for the sake of simplicity so as to illustrate the main idea, we shall only discuss the simplest case when $D = Q = 1$, while the general case can be inferred directly. In the general regression framework, we can write:

$$\mathrm{y} = m(\mathrm{x}) + \varepsilon\,,$$

where $m(x) = \mathbb{E}(\mathrm{y}|\mathrm{x} = x)$, and $\varepsilon$ and x are uncorrelated so that $\mathbb{E}(\varepsilon|\mathrm{x} = x) = 0$ for all $x$, and we further define $\sigma^2(x) := \mathrm{Var}(\varepsilon|\mathrm{x} = x) = \mathrm{Var}(\mathrm{y}|\mathrm{x} = x)$. Often, we assume that $m(x)$ is a piecewisely smooth function given the feature variable x evaluated at $x$, so that it is $C^{(J)}$ for the interior points and $C^{(J-1)}$ for various knots (boundary splitting points). When $m(x)$ is smooth, for a given point $x_0 \in \mathbb{R}$, we expect that $m(x_n)$ should be close to $m(x_0)$ for any observation $x_n$ in the proximity of $x_0$, and so a reasonable estimate for $m(x_0)$ should be an average of the responses $y_n = m(x_n) + \varepsilon_n$ of these $x_n$'s.

Now, usual Taylor's expansion up to the order $J$ gives us:

$$\begin{aligned} m(x) &\approx m(x_0) + m^{(1)}(x_0)(x - x_0) + \frac{1}{2!}m^{(2)}(x_0)(x - x_0)^2 + \cdots + \frac{1}{J!}m^{(J)}(x_0)(x - x_0)^J \\ &=: \beta_0 + \beta_1(x - x_0) + \beta_2(x - x_0)^2 + \cdots + \beta_J(x - x_0)^J\,, \end{aligned}$$

which resembles a parametric model, with parameters $\{\beta_j = m^{(j)}(x_0)\}_{j=0}^J$, which can be estimated by parametric regression with the observed data $x_n$'s scattering around $x_0$. However, this method is only a local one, and the estimated parameters may deviate significantly by using different datasets in various neighbourhoods of $x_0$, this results in biased estimation. Worse still, there may even be very few observations close enough to $x_0$, which further renders the estimation fairly inaccurate.

To remedy this shortcoming with a balanced tradeoff, we instead adopt a weighted least square approach by minimizing the following objective function which uses the whole observed dataset:

$$\min_{\beta_0,\cdots,\beta_J} \sum_{n=1}^{N} \Big( y_n - \beta_0 - \beta_1(x_n - x_0) - \cdots - \beta_J(x_n - x_0)^J \Big)^2 K_{h_N}(x_0 - x_n)\,,$$

here the weighting function $K_{h_N}$ is called a *kernel function*, and $h_N$ is its bandwidth whose subscript indicates that its choice usually depends on the sample size $N$; this also explains why this approach is named *Kernel Regression*. The role of $K_{h_N}$ is to control the amount of effect brought by each sample point $(x_n, y_n)$ so that its contribution to the estimation of $m(x)$ is weighed according to the distance between its feature value $x_n$ and the centroid $x_0$; the smaller the distance, the higher the amount of its influence and relevance. Instead of taking a quite general choice of $K_{h_N}$, we usually assume a simpler form of $K_{h_N}(x_0 - x_n) = \frac{1}{h_N}K\left(\frac{x_0 - x_n}{h_N}\right)$ in practice, where the choice of $K$ itself is totally independent of $h_N$. Furthermore, for technical reasons, we require the kernel function $K$ to satisfy the following requirements:

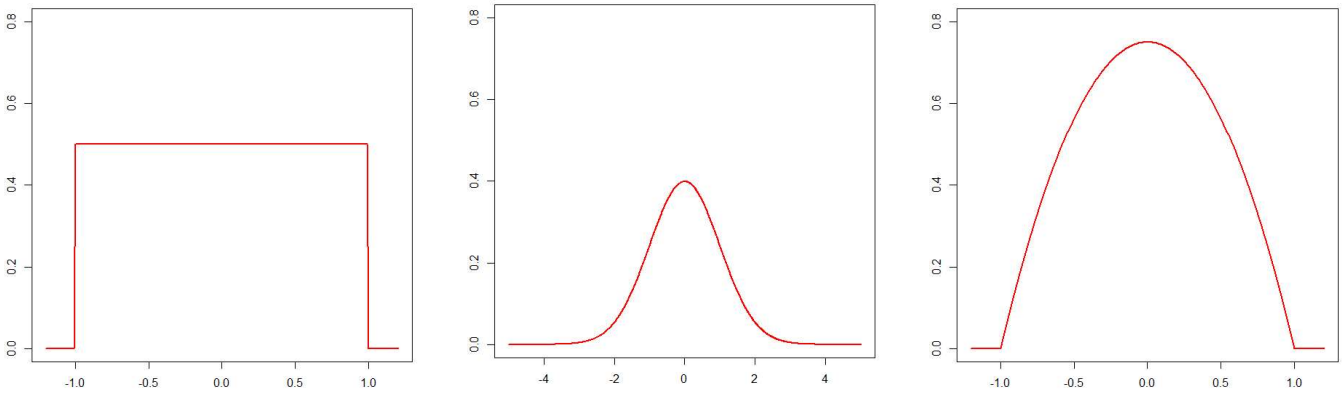
(1) $\int_{\mathbb{R}} K(u)\,du = 1$, where $K(u)$ is positive;

(2) $K(u) = K(-u)$, *i.e.* $K$ is symmetric about the origin; and

(3) $K(u)$ attains its global maximum at 0, which reflects that the weights are larger for the observations

$x_n$'s that are close to $x_0$.

Several commonly adopted choices of kernel functions are illustrated in Figure 5.4.1:




(a) Uniform kernel: $K(u) = \frac{1}{2}\mathbb{1}_{\{|u|\leq 1\}}$

(b) Gaussian kernel: $K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$

(c) Epanechnikov kernel: $K(u) = \frac{3}{4}(1-u^2)\mathbb{1}_{\{|u|\leq 1\}}$

Figure 5.4.1: Illustrations of three commonly adopted kernel functions.

Apparently all these kernel functions are smooth in $x$, and so is the resulting estimate $\hat{m}(x)$. Though different kernels give different numerical results, many commonly used ones tend to produce similar comparable estimates when the corresponding bandwidths are suitably rescaled.

For the construction of an estimator $\hat{\mathrm{m}}(x_0)$, for every $x_0 \in \mathbb{R}$, we first consider the simplest form when $m(x_0)$ is approximated by a constant, $\beta_0(x_0)$, regarded as an average of the labels mainly contributed by those data points $x_n$'s in a neighborhood of $x_0$. The resulting optimization problem is also called *local constant regression* (also see Loader (2006)), and the corresponding estimator, as a function of $x_0$, takes the following form:

$$\arg\min_{\beta_0(x_0)} \frac{1}{N}\sum_{n=1}^{N} \frac{1}{h_N} K\left(\frac{x_0 - \mathrm{x}_n}{h_N}\right)(\mathrm{y}_n - \beta_0(x_0))^2, \qquad \text{for any } x_0 \in \mathbb{R}\,. \tag{5.4.1}$$

The first-order condition, namely by equating with 0 the first-order derivative of the objective function with respect to $\beta_0(x_0)$, is:

$$-\frac{2}{Nh_N}\sum_{n=1}^{N} K\left(\frac{x_0 - \mathrm{x}_n}{h_N}\right)(\mathrm{y}_n - \beta_0(x_0)) = 0\,,$$

which genuinely gives the following estimator:

$$\hat{\mathrm{m}}(x_0) = \hat{\beta}_0(x_0) = \frac{\sum_{n=1}^{N} K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)\mathrm{y}_n}{\sum_{n=1}^{N} K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)}\,, \tag{5.4.2}$$

which is a local mean of $\mathrm{y}_1, \cdots, \mathrm{y}_n$ around $x_0$, and this formula is called the *Nadaraya-Watson estimator* (see [3, 4]) for $m(x_0)$ given a random sample $\{(\mathrm{x}_n, \mathrm{y}_n)\}_{n=1}^{N}$.

Under some regularity conditions to be introduced in Subsection 5.4.2, its bias and variance possess the following properties:

- $\text{Bias}(\hat{\text{m}}(x_0)) = \mathbb{E}(\hat{\text{m}}(x_0)) - m(x_0) \leq C_1 h_N^2$, where $C_1 > 0$ is a universal constant independent of $x_0$;
- $\text{Var}(\hat{\text{m}}(x_0)) \leq C_2/(Nh_N)$, where $C_2 > 0$ is another constant independent of $x_0$.

Firstly, the kernel function $K_{h_N}$ depends on both $x_0$ and $h_N$, the bandwidth, and as $h_N$ gets smaller, the estimator $\hat{\text{m}}(x)$ is less biased but induces a greater variance. Besides, the bandwidth $h_N$ also determines the smoothness (degree of smoothness): a larger $h_N$ increases the smoothness of $\hat{\text{m}}(x)$, while a smaller $h$ results in a more oscillatory $\hat{\text{m}}(x)$; this is sometimes called *bias-variance tradeoff* in the context of statistical machine learning.

### 5.4.1 Rosenblatt-Parzen Kernel Density Estimation

An alternative approach to motivate the formula of the Nadaraya-Watson estimator can be via the kernel density estimation of $f_{\text{x}}(x_0)$, namely the *Rosenblatt-Parzen kernel density estimator*, see [5, 6]:

$$\hat{f}_{\text{x}}(x_0) = \frac{1}{Nh_N} \sum_{n=1}^{N} K\left(\frac{x_0 - \text{x}_n}{h_N}\right), \tag{5.4.3}$$

and by definition $\int_{\mathbb{R}} \hat{f}_{\text{x}}(x)\, dx = 1$.

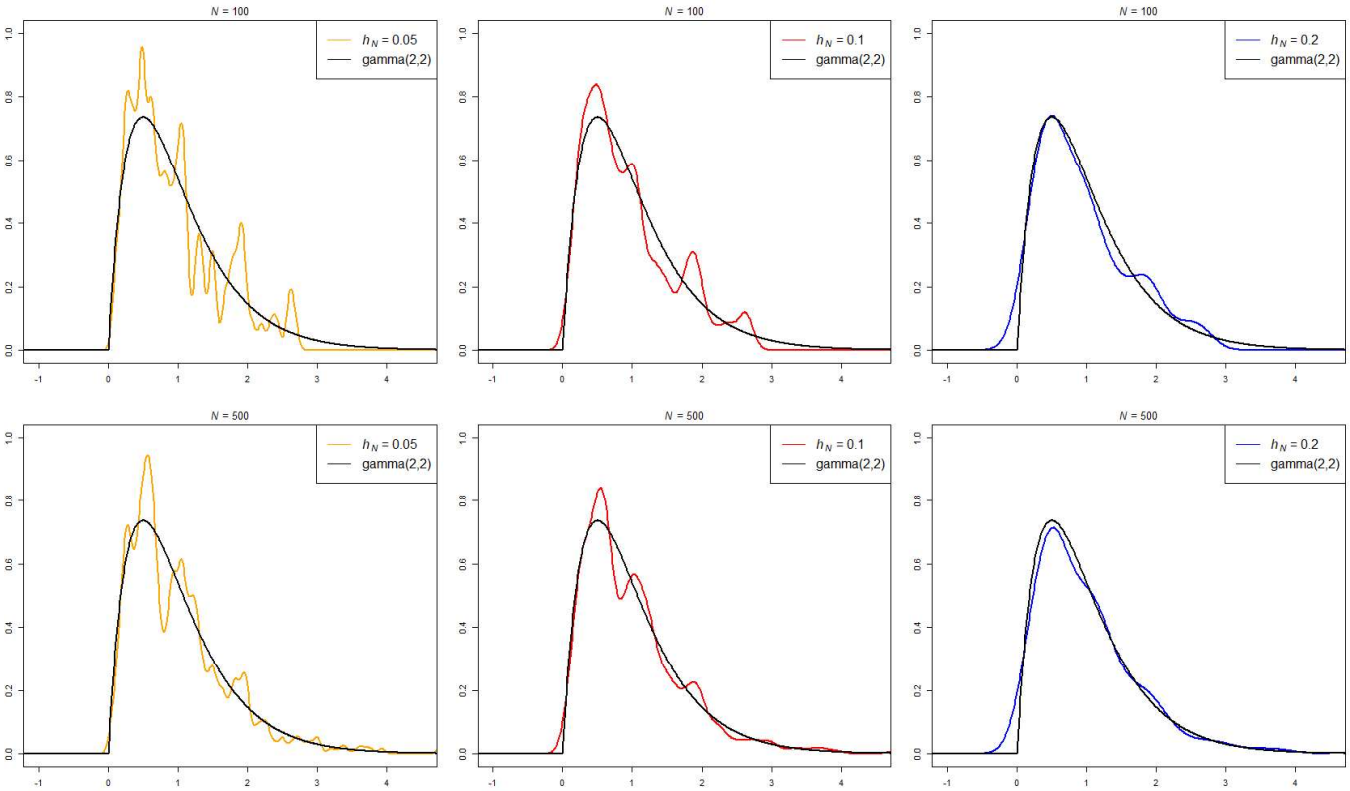



Figure 5.4.2: Kernel density estimator of a random sample following Gamma(2,2) with $h_N = 0.05, 0.1, 0.2$ and sample size $N = 100$ and 500.

An important property of this estimator is that it converges to the true density $f_{\text{x}}(x_0)$ in probability under the regularity conditions to be introduced in Subsection 5.4.2. Figure 5.4.2 illustrates 6 scenarios with different combinations of $h_N = 0.05, 0.1, 0.2$ with respective sample size $N = 100$ and 500, we observe that

$N = 500$ fits with the original theoretical distribution (Gamma(2,2)) better than $N = 100$ for every $h_N$. To see the claim of convergence, we first calculate the expectation of $\hat{f}_{\mathrm{x}}(x_0)$:

$$\begin{aligned}
\mathbb{E}[\hat{f}_{\mathrm{x}}(x_0)] &= \frac{1}{Nh_N}\sum_{n=1}^{N}\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)\right] \\
&= \frac{1}{h_N}\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)\right] \\
&= \int_{\mathbb{R}}\frac{1}{h_N}K\left(\frac{x_0-u}{h_N}\right)f_{\mathrm{x}}(u)\,du \\
&= \int_{\mathbb{R}}K(z)f_{\mathrm{x}}(x_0+h_Nz)\,dz \\
&= \int_{\mathbb{R}}K(z)f_{\mathrm{x}}(x_0)\,dz + O(h_N) \\
&= f_{\mathrm{x}}(x_0)+O(h_N); \qquad (5.4.4)
\end{aligned}$$

note that to obtain the second last equality, we decompose the integral in the fourth line above so that, for an arbitrarily small $\varepsilon > 0$,

$$\int_{\mathbb{R}}K(z)f_{\mathrm{x}}(x_0+h_Nz)\,dz = \int_{|z|>\frac{\varepsilon}{h_N}}K(z)f_{\mathrm{x}}(x_0+h_Nz)\,dz + \int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}K(z)f_{\mathrm{x}}(x_0+h_Nz)\,dz\,. \qquad (5.4.5)$$

Since $f_{\mathrm{x}}(x)$ is a valid density, there exists a finite number $M$ such that $f_{\mathrm{x}}(x) \le M$ almost everywhere in $\mathbb{R}\backslash(x_0-\varepsilon, x_0+\varepsilon)$, therefore,

$$\left|\int_{|z|>\frac{\varepsilon}{h_N}}K(z)f_{\mathrm{x}}(x_0+h_Nz)\,dz\right| \le M\int_{|z|>\frac{\varepsilon}{h_N}}|K(z)|\,dz \to 0, \qquad \text{as} \quad h_N\to 0\,,$$

which implies that the first integral term is $o(1)$ as $h_N \to 0$. Furthermore, since $f_{\mathrm{x}}$ is continuously differentiable by Assumption 3 in Subsection 5.4.2, we have by Taylor expansion $f_{\mathrm{x}}(x_0+h_Nz) = f_{\mathrm{x}}(x_0)+h_Nzf'_{\mathrm{x}}(\tilde{x}_0)$, where $\tilde{x}_0$ is some value between $x_0$ and $x_0+h_Nz$, while the continuity of $f'_{\mathrm{x}}$ also implies that there exists some $\delta_\varepsilon > 0$ such that $|f'_{\mathrm{x}}(x) - f'_{\mathrm{x}}(x_0)| < \delta_\varepsilon$ for any $x \in (x_0-\varepsilon, x_0+\varepsilon)$, hence $|f'_{\mathrm{x}}(x)| < |f'_{\mathrm{x}}(x_0)| + \delta_\varepsilon$ for all $x$ in this region. Hence, the second integral term in (5.4.5) can be decomposed as:

$$\begin{aligned}
\int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}K(z)f_{\mathrm{x}}(x_0+h_Nz)\,dz &= \int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}K(z)\{f_{\mathrm{x}}(x_0)+h_Nzf'_{\mathrm{x}}(\tilde{x}_0)\}\,dz \\
&= f_{\mathrm{x}}(x_0)\int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}K(z)\,dz + h_N\int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}zK(z)f'_{\mathrm{x}}(\tilde{x}_0)\,dz\,,
\end{aligned}$$

where $\int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}K(z)\,dz \to \int_{-\infty}^{\infty}K(z)\,dz = 1$ as $h_N \to 0$, and

$$\begin{aligned}
\left|h_N\int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}zK(z)f'_{\mathrm{x}}(\tilde{x}_0)\,dz\right| &\le h_N\{|f'_{\mathrm{x}}(x_0)|+\delta_\varepsilon\}\int_{-\frac{\varepsilon}{h_N}}^{\frac{\varepsilon}{h_N}}|z|K(z)\,dz \\
&\le h_N\{|f'_{\mathrm{x}}(x_0)|+\delta_\varepsilon\}\int_{-\infty}^{\infty}|z|K(z)\,dz \to 0, \qquad \text{as} \quad h_N\to 0\,,
\end{aligned}$$

since $K(z)$ has a finite second moment by Assumption 4 in Subsection 5.4.2, which yields $\int_{-\infty}^{\infty}|z|K(z)\,dz < \infty$. Finally,

$$\int_{-\infty}^{\infty}K(z)f_{\mathrm{x}}(x_0+h_Nz)\,dz = o(1)+f_{\mathrm{x}}(x_0)(1-o(1))+o(1) = f_{\mathrm{x}}(x_0)+o(1)\,.$$

Next, the variance of $\hat{f}_{\mathrm{x}}(x_0)$ is:

$$\begin{aligned}
\mathrm{Var}(\hat{f}_{\mathrm{x}}(x_0)) &= \frac{1}{Nh_N^2}\mathrm{Var}\left(K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)\right)\\
&= \frac{1}{Nh_N^2}\left\{\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)^2\right]-\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)\right]^2\right\}\\
&= \frac{1}{Nh_N^2}\left(\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)^2\right]-O(h_N^2)\right)\\
&= \frac{1}{Nh_N^2}\left(\int_{\mathbb{R}} K(z)^2 f_{\mathrm{x}}(x_0+h_N z)h_N\,dz - O(h_N^2)\right)\\
&= \frac{1}{Nh_N}\left(\int_{\mathbb{R}} K(z)^2 (f_{\mathrm{x}}(x_0)+O(h_N))\,dz - O(h_N)\right)\\
&= \frac{1}{Nh_N} f_{\mathrm{x}}(x_0)\int_{\mathbb{R}} K(z)^2\,dz + O\left(\frac{1}{N}\right),
\end{aligned}$$

note that $\int_{\mathbb{R}} K(z)^2\,dz < \infty$ by Assumption 4 in Subsection 5.4.2. Therefore for all $\delta > 0$, by Chebyshev's inequality,

$$\mathbb{P}\left(\left|\hat{f}_{\mathrm{x}}(x_0) - \mathbb{E}[\hat{f}_{\mathrm{x}}(x_0)]\right| \geq \delta\right) \leq \frac{\mathrm{Var}(\hat{f}_{\mathrm{x}}(x_0))}{\delta^2},$$

which tends to 0 when $h_N \to 0$ and $Nh_N \to \infty$ as $N \to \infty$ by Assumption 5 in Subsection 5.4.2, and hence we can conclude that $\hat{f}_{\mathrm{x}}(x_0) \xrightarrow{p} f_{\mathrm{x}}(x_0)$.

Now, given an observed dataset $\mathcal{S} = \{(x_n, y_n)\}_{n=1}^N$, by the *Plug-in Principle*[2] we can similarly estimate the joint density of $(\mathrm{x}, \mathrm{y})$ by the product kernel:

$$\hat{f}_{\mathrm{x,y}}(x,y) = \frac{1}{Nh_N^2}\sum_{n=1}^N K\left(\frac{x-\mathrm{x}_n}{h_N}\right)K\left(\frac{y-\mathrm{y}_n}{h_N}\right), \qquad \text{for any } (x,y)\in\mathbb{R}^2,$$

which allows us to further obtain the following estimate:

$$\begin{aligned}
\int_{\mathbb{R}} y f_{\mathrm{x,y}}(x_0,y)\,dy &\approx \int_{\mathbb{R}} y\frac{1}{Nh_N^2}\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)K\left(\frac{y-y_n}{h_N}\right)dy\\
&= \frac{1}{Nh_N}\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)\int_{\mathbb{R}}\frac{y}{h_N}K\left(\frac{y-y_n}{h_N}\right)dy\\
&= \frac{1}{Nh_N}\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)\left(\int_{\mathbb{R}}\frac{y-y_n}{h_N}K\left(\frac{y-y_n}{h_N}\right)dy + y_n\int_{\mathbb{R}}\frac{1}{h_N}K\left(\frac{y-y_n}{h_N}\right)dy\right)^3\\
&= \frac{1}{Nh_N}\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)y_n,
\end{aligned}$$

hence we can estimate $m(x)$ as follows:

$$m(x_0) = \frac{\int_{\mathbb{R}} y f_{\mathrm{x,y}}(x_0,y)\,dy}{f_{\mathrm{x}}(x_0)} \approx \frac{\frac{1}{Nh_N}\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)y_n}{\frac{1}{Nh_N}\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)} = \frac{\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)y_n}{\sum_{n=1}^N K\left(\frac{x_0-x_n}{h_N}\right)}, \qquad \text{for any } x_0\in\mathbb{R},$$

which is precisely the Nadaraya-Watson estimate given the dataset $\mathcal{S}$.

[2]This is a technique commonly adopted in statistics, namely a feature, let say the expected value, of a given distribution can be approximated by the same feature of the empirical distribution of a sample of observations drawn from the given distribution, also see detailed discussions in Rice (2006).

[3]By symmetry, the first integral in the bracket becomes 0, while the second term can be rewritten as $y_n\int_{\mathbb{R}} K\left(\frac{y-y_n}{h_N}\right)d\left(\frac{y-y_n}{h_N}\right)$, which equals $y_n$ by the very definition of the kernel function $K$.

More generally, at $x_0 \in \mathbb{R}$, $m(x_0)$ can be approximated by an $r$-th order polynomial in $\mathrm{x}_n - x_0$, rather just a local constant $\beta_0(x_0)$, then we obtain the so-called *local polynomial kernel regression estimator*; see Fan and Gijbels (2003) for more details. For instance, when $r = 1$, the minimization problem becomes:

$$\min_{\beta_0(x_0),\beta_1(x_0)} \sum_{n=1}^{N} \Big(\mathrm{y}_n - \beta_0(x_0) - \beta_1(x_0)(\mathrm{x}_n - x_0)\Big)^2 \cdot \frac{1}{h_N} K\left(\frac{x_0 - \mathrm{x}_n}{h_N}\right), \qquad \text{for any } x_0 \in \mathbb{R},$$

and by using the standard least squares method, we obtain the least squares estimator:

$$\begin{pmatrix} \hat{\beta}_0(x_0) \\ \hat{\beta}_1(x_0) \end{pmatrix} = (\mathbf{X}^\top \mathbf{K} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{K} \mathbf{y},$$

where

$$\mathbf{X} = \begin{pmatrix} 1 & \mathrm{x}_1 - x_0 \\ \vdots & \vdots \\ 1 & \mathrm{x}_N - x_0 \end{pmatrix}, \qquad \mathbf{K} = \operatorname{diag}\left(K\left(\frac{x_0 - \mathrm{x}_1}{h_N}\right), \cdots, K\left(\frac{x_0 - \mathrm{x}_N}{h_N}\right)\right), \qquad \mathbf{y} = \begin{pmatrix} \mathrm{y}_1 \\ \vdots \\ \mathrm{y}_N \end{pmatrix};$$

note that $\hat{\beta}_0(x_n)$ (resp. $\hat{\beta}_1(x_n)$) is essentially the dominant term for the function value (resp. first-order derivative) of $\hat{\mathrm{m}}$ at every observed data point $x_n$, particularly for a small value of $h_N$.

For a general value of $r$, the minimization problem takes the following form:

$$\min_{\beta_0(x_0),\beta_1(x_0),\cdots,\beta_r(x_0)} \sum_{n=1}^{N} \left(y_n - \sum_{j=0}^{r} \beta_j(x_0)(\mathrm{x}_n - x_0)^j\right)^2 \cdot \frac{1}{h_N} K\left(\frac{x_0 - \mathrm{x}_n}{h_N}\right)$$

$$= \min_{\boldsymbol{\beta}(x_0)} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}(x_0))^\top \mathbf{K} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}(x_0)), \qquad \text{for } x_0 \in \mathbb{R},$$

whose solution is the following least squares estimator:

$$\hat{\boldsymbol{\beta}}(x_0) = \begin{pmatrix} \hat{\beta}_0(x_0) \\ \vdots \\ \hat{\beta}_r(x_0) \end{pmatrix} = (\mathbf{X}^\top \mathbf{K} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{K} \mathbf{y},$$

where

$$\boldsymbol{X} = \begin{pmatrix} 1 & \mathrm{x}_1 - x_0 & \cdots & (\mathrm{x}_1 - x_0)^r \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \mathrm{x}_N - x_0 & \cdots & (\mathrm{x}_N - x_0)^r \end{pmatrix} \in \mathbb{R}^{N \times (r+1)}, \quad \mathbf{K} = \operatorname{diag}\left(K\left(\frac{x_0 - \mathrm{x}_1}{h_N}\right), \cdots, K\left(\frac{x_0 - \mathrm{x}_N}{h_N}\right)\right), \quad \mathbf{y} = \begin{pmatrix} \mathrm{y}_1 \\ \vdots \\ \mathrm{y}_N \end{pmatrix}.$$

### 5.4.2 Asymptotic Statistical Properties of $\hat{\mathrm{m}}$

We first make the following assumptions:

1. $m(x)$ is twice continuously differentiable in $x$;
2. $\sigma^2(x) = \mathrm{Var}(\varepsilon | \mathrm{x} = x)$ is continuous and uniformly positive in $x$;
3. $f_{\mathrm{x}}$ is continuously differentiable and is bounded away from zero on $\mathbb{R}$;
4. $K$ has a finite second moment and is square integrable, *i.e.* $\int_{\mathbb{R}} K(z) z^2 \, dz, \int_{\mathbb{R}} K^2(z) \, dz < \infty$;
5. $\{h_N\}$ is a deterministic sequence such that $N h_N \to \infty$ as $h_N \to 0$, *i.e.* $\frac{1}{h_N} = o(N)$.

We first note that by definition, for any $x_0 \in \mathbb{R}$, recall the Rosenblatt-Parzen kernel density estimator $\hat{f}$ of (5.4.3),

$$\begin{aligned}\frac{1}{Nh_N}\sum_{n=1}^{N} K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)\mathrm{y}_n &= \frac{1}{Nh_N}\sum_{n=1}^{N} K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)(m(x_0)+(m(\mathrm{x}_n)-m(x_0))+\varepsilon_n)\\ &= \hat{f}_{\mathrm{x}}(x_0)m(x_0)+\frac{1}{Nh_N}\sum_{n=1}^{N} K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)(m(\mathrm{x}_n)-m(x_0))+\frac{1}{Nh_N}\sum_{n=1}^{N} K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)\varepsilon_n\\ &=: \hat{f}_{\mathrm{x}}(x_0)m(x_0)+\hat{\mathrm{m}}_1(x_0)+\hat{\mathrm{m}}_2(x_0)\,,\end{aligned}$$

where we suppress the symbol $h_N$ in all the notations of $\hat{\mathrm{m}}_i$ without any cause of ambiguity. Hence,

$$\hat{\mathrm{m}}(x_0)=\frac{\frac{1}{Nh_N}\sum_{n=1}^{N} K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)\mathrm{y}_n}{\hat{f}_{\mathrm{x}}(x_0)} = m(x_0)+\frac{\hat{\mathrm{m}}_1(x_0)}{\hat{f}_{\mathrm{x}}(x_0)}+\frac{\hat{\mathrm{m}}_2(x_0)}{\hat{f}_{\mathrm{x}}(x_0)}\,,$$

we then take a closer look at the first two moments of $\hat{\mathrm{m}}_1$ and $\hat{\mathrm{m}}_2$. Firstly, since $\mathbb{E}(\varepsilon_n|\mathrm{x}_n=x_n)=0$ for all $n=1,\cdots,N$, we can easily obtain that

$$\mathbb{E}[\hat{\mathrm{m}}_2(x_0)]=\frac{1}{h_N}\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)\varepsilon_1\right]=\frac{1}{h_N}\mathbb{E}\left\{\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)\varepsilon_1\middle|\mathrm{x}_1\right]\right\}=0\,,$$

hence its variance is

$$\begin{aligned}\mathrm{Var}(\hat{\mathrm{m}}_2(x_0)) &= \frac{1}{(Nh_N)^2}\sum_{n=1}^{N}\mathbb{E}\left[\left(K\left(\frac{x_0-\mathrm{x}_n}{h_N}\right)\varepsilon_n\right)^2\right]\\ &= \frac{1}{nh_N^2}\mathbb{E}\left[K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)^2\sigma^2(\mathrm{x}_1)\right]\\ &= \frac{1}{Nh_N^2}\int_{\mathbb{R}} K\left(\frac{u-x_0}{h_N}\right)^2\sigma^2(u)f_{\mathrm{x}}(u)\,du\\ &= \frac{1}{Nh_N^2}\int_{\mathbb{R}} K(z)^2\sigma^2(x_0+h_Nz)f_{\mathrm{x}}(x_0+h_Nz)h_N\,dz\\ &= \frac{1}{Nh_N^2}\int_{\mathbb{R}} K(z)^2\sigma^2(x_0)f_{\mathrm{x}}(x_0)h_N\,dz+o\left(\frac{1}{Nh_N}\right)\\ &= \frac{1}{Nh_N}\sigma^2(x_0)f_{\mathrm{x}}(x_0)\int_{\mathbb{R}} K^2(z)\,dz+o\left(\frac{1}{Nh_N}\right)\,,\end{aligned}$$

where the second equality is obtained by applying the double expectation formula, and the third one makes use of the symmetry of $K$; here we also assume that $\sigma^2(u)$ and $f_{\mathrm{x}}(u)$ are smooth enough at every point $u\in\mathbb{R}$, which allows us to adopt an analogous argument to that for the second last equality of (5.4.4) to obtain the second last equality above. Finally, since every pair $(\mathrm{x}_n,\varepsilon_n)$ is independent of all the rest, we can then apply Central Limit Theorem which implies that

$$\sqrt{Nh_N}\hat{\mathrm{m}}_2(x_0)\xrightarrow{d}\mathcal{N}\left(0,\sigma^2(x_0)f_{\mathrm{x}}(x_0)\int_{\mathbb{R}}K^2(z)\,dz\right)\,.$$

Finally, by the property of (5.4.3) that $\hat{f}_{\mathrm{x}}(x_0)\xrightarrow{p}f_{\mathrm{x}}(x_0)$, Slutsky's theorem gives that

$$\sqrt{Nh_N}\,\frac{\hat{\mathrm{m}}_2(x_0)}{\hat{f}_{\mathrm{x}}(x_0)}\xrightarrow{d}\mathcal{N}\left(0,\frac{\sigma^2(x_0)}{f_{\mathrm{x}}(x_0)}\int_{\mathbb{R}}K^2(z)z^2\,dz\right)\,.$$

On the other hand, the mean of $\hat{\mathrm{m}}_1(x_0)$ is

$$\begin{aligned}
\mathbb{E}(\hat{\mathrm{m}}_1(x_0)) &= \frac{1}{h_N}\mathbb{E}\left(K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)(m(\mathrm{x}_1)-m(x_0))\right)\\
&= \int_{\mathbb{R}} K(z)(m(x_0+h_Nz)-m(x_0))f_{\mathrm{x}}(x_0+h_Nz)\,dz\\
&= \int_{\mathbb{R}} K(z)\left(m(x_0)+m'(x_0)h_Nz+m''(x_0)\frac{h_N^2z^2}{2}-m(x_0)\right)(f_{\mathrm{x}}(x_0)+f'_{\mathrm{x}}(x_0)h_Nz)\,dz+o(h_N^2)\\
&= h_Nm'(x_0)f_{\mathrm{x}}(x_0)\int_{\mathbb{R}}K(z)z\,dz+h_N^2\left(\frac{1}{2}m''(x_0)f_{\mathrm{x}}(x_0)+m'(x_0)f'_{\mathrm{x}}(x_0)\right)\int_{\mathbb{R}}K(z)z^2\,dz+o(h_N^2)\\
&= h_N^2\left(\frac{1}{2}m''(x_0)+m'(x_0)\frac{f'_{\mathrm{x}}(x_0)}{f_{\mathrm{x}}(x_0)}\right)f_{\mathrm{x}}(x_0)\int_{\mathbb{R}}K(z)z^2\,dz+o(h_N^2)\,,
\end{aligned}$$

where the third equation again uses the same trick as the second last equality in (5.4.4), and the first term in the second last line vanishes because of the symmetry about 0 for $K$. For its variance, we have

$$\begin{aligned}
\mathrm{Var}(\hat{\mathrm{m}}_1(x_0)) &= \frac{1}{Nh_N^2}\mathrm{Var}\left(K\left(\frac{x_0-\mathrm{x}_1}{h_N}\right)(m(\mathrm{x}_1)-m(x_0))\right)\\
&= \frac{1}{Nh_N^2}\left[\mathbb{E}\left(K\left(\frac{x_0-\mathrm{x}}{h_N}\right)^2(m(\mathrm{x})-m(x_0))^2\right)-O(h_N^6)\right]\\
&= \frac{1}{Nh_N^2}\left[\int_{\mathbb{R}}K(z)^2(m(x_0+h_Nz)-m(x_0))^2f_{\mathrm{x}}(x_0+h_Nz)h_N\,dz-O(h_N^6)\right]\\
&= \frac{1}{Nh_N}\left[\int_{\mathbb{R}}K(z)^2(m'(x_0)h_Nz+o(h_N^2))^2(f_{\mathrm{x}}(x_0)+O(h_N))\,dz-O(h_N^5)\right]\\
&= \frac{1}{Nh_N}\left[h_N^2(m'(x_0))^2f_{\mathrm{x}}(x_0)\int_{\mathbb{R}}K(z)^2z^2\,dz+o(h_N^2)\right]\\
&= \frac{h_N}{N}(m'(x_0))^2f_{\mathrm{x}}(x_0)\int_{\mathbb{R}}K(z)^2z^2\,dz+o\left(\frac{h_N}{N}\right).
\end{aligned}$$

Hence, as $h_N\to 0$ and $Nh_N\to\infty$, we can see by Chebyshev's inequality that

$$\sqrt{Nh_N}\left(\hat{\mathrm{m}}_1(x_0)-h_N^2\left(\frac{1}{2}m''(x_0)+m'(x_0)\frac{f'_{\mathrm{x}}(x_0)}{f_{\mathrm{x}}(x_0)}\right)f_{\mathrm{x}}(x_0)\int_{\mathbb{R}}K(z)z^2\,dz\right)\xrightarrow{p}0\,;$$

in addition, $\hat{f}_{\mathrm{x}}(x_0)\xrightarrow{p}f_{\mathrm{x}}(x_0)$ for each $x_0\in\mathbb{R}$, which implies

$$\sqrt{Nh_N}\left(\frac{\hat{\mathrm{m}}_1(x_0)}{\hat{f}_{\mathrm{x}}(x_0)}-h_N^2\left(\frac{1}{2}m''(x_0)+m'(x_0)\frac{f'_{\mathrm{x}}(x_0)}{f_{\mathrm{x}}(x_0)}\right)\int_{\mathbb{R}}K(z)z^2\,dz\right)\xrightarrow{p}0\,.$$

In summary, we conclude with the following asymptotic normality of $\hat{\mathrm{m}}(x_0)$: for each $x_0\in\mathbb{R}$,

$$\sqrt{Nh_N}\left(\hat{\mathrm{m}}(x_0)-m(x_0)-h_N^2\left(\frac{1}{2}m''(x_0)+m'(x_0)\frac{f'_{\mathrm{x}}(x_0)}{f_{\mathrm{x}}(x_0)}\right)\int_{\mathbb{R}}K(z)z^2\,dz\right)\xrightarrow{d}\mathcal{N}\left(0,\frac{\sigma^2(x_0)}{f_{\mathrm{x}}(x_0)}\int_{\mathbb{R}}K(z)^2\,dz\right),$$

based on which we see that the bias term is of order $O(h_N^2)$, and it is larger when $m(x)$ is curvier at $x_0$, *i.e.* larger in values of $m'(x_0)$ and $m''(x_0)$. Nevertheless, the Nadaraya-Watson estimator is still asymptotically consistent with a convergence rate of $\sqrt{Nh_N}$.

## 5.5 Logistic Regression

Logistic regression is a supervised classification (mostly) learner. However, logistic regression is fundamentally different from multiple linear regression as the latter specifies a prediction in value, while the former gives the chance of taking a certain label. In this section, we shall mainly focus on binary classification and its extension to multi-label classification is straightforward.

### 5.5.1 Introduction with Binary Response

The ordinary linear regression has its limitations and can be easily abused; indeed, recall that ordinary linear regression has the following assumptions:

1. The regression function of $y$ is a linear function in the independent variables $\boldsymbol{x}$'s, *i.e.*
$$\mathbb{E}(y|\boldsymbol{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_D x^{(D)};$$
2. The idiosyncratic noises $\varepsilon_i$'s are at least uncorrelated, if not totally iid, with a common distribution with the mean zero and finite variance $\sigma^2 < \infty$.

Usually, residuals plots are used to check for these assumptions. A Q-Q normal plot of the residuals is used to further check the normality assumption on the idiosyncratic noises, normally closer the plot to the main diagonal, more trust on their normality can be put. The plot of residuals versus fitted values of the observation is used to check the linearity and constant error variance assumptions, for instance, a uniform horizontal band of the plot means that the linear model is well-fitted with a homogeneous variance for the noise term. The plot of residuals versus lag (residuals) is used to check uncorrelation or the independence of noises, again a uniform horizontal band without any pattern stands for the absense of correlation. Let us look at the following financial example which demonstrates how the linear regression can be abused.

The data file "fin-ratio.csv" contains financial ratios of 680 securities listed in the main board of Hong Kong Stock Exchange in 2002, in which there are six financial variables, namely, **Earning Yield** (EY), **Cash Flow to Price** (CFTP), **logarithm of Market Value** (ln_MV), **Dividend Yield** (DY), **Book to Market Equity** (BTME), **Debt to Equity Ratio** (DTE). These financial variables are publicly available information, for instance, found in Yahoo Finance (`https://finance.yahoo.com/`). Among these companies, there are 32 Blue Chips which are the Hang Seng Index Constituent Stocks. The last column HSI is a binary variable indicating whether the stock is a Blue Chip or not. We now want to establish any relationship between HSI and these mentioned six financial variables. One can run an ordinary least square regression of HSI against these six financial variables by using **R**'s built-in function `lm()` (Linear Model), see the codes and output in the following Programme 5.5.1:

```
> d <- read.csv("fin-ratio.csv")
> names(d)
[1] "EY"    "CFTP"  "ln_MV" "DY"    "BTME"  "DTE"   "HSI"
> summary(lm(HSI~EY+CFTP+ln_MV+DY+BTME+DTE,data=d))
# linear model

```

```
Call:
lm(formula = d$HSI ~ d$EY + d$CFTP + d$ln_MV + d$DY + d$BTME + d$DTE)

Residuals:
     Min       1Q   Median       3Q      Max
-0.32104 -0.08546 -0.01672  0.05592  0.73866
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.4591209  0.0268310 -17.112  < 2e-16 ***
d$EY        -0.0017172  0.0016181  -1.061  0.28896
d$CFTP      -0.0103792  0.0037321  -2.781  0.00557 **
d$ln_MV      0.0810286  0.0040887  19.818  < 2e-16 ***
d$DY        -0.0027336  0.0017826  -1.534  0.12561
d$BTME       0.0004798  0.0007938   0.604  0.54575
d$DTE        0.0010610  0.0018035   0.588  0.55655
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 0.1689 on 673 degrees of freedom
Multiple R-Squared: 0.3708,     Adjusted R-squared: 0.3652
F-statistic: 66.09 on 6 and 673 DF,  p-value: < 2.2e-16
```

Programme 5.5.1: Codes and output of the ordinary least square regression of HSI against the six financial variables

From the output, the $p$-value of `d$DTE` is the largest and it is far greater than the acceptance threshold (say 0.1), this means the coefficient of `DTE` is not significantly different from zero. Therefore, we should exclude it in our regression. We then continue to fit the regression using only the remaining five independent financial variables:

```
> summary(lm(HSI~EY+CFTP+ln_MV+DY+BTME,data=d))
```

,

which we further found that the $p$-value of `d$BTME` was the largest far greater than 0.1, and this suggests that `d$BTME` should be excluded. We keep on excluding the variable with the largest $p$-value one by one until all the $p$-values are small and less than 0.1; this procedure is one of the common model selection methods in regression known as the **backward elimination**. Finally, we arrive at the following model, see Programme 5.5.2:

```
> summary(lm(HSI~CFTP+ln_MV,data=d))
Call:
lm(formula = d$HSI ~ d$CFTP + d$ln_MV)
Residuals:
 Min       1Q   Median       3Q      Max
-0.32409 -0.08559 -0.01729  0.05688  0.73488

```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.454781   0.026284 -17.303  < 2e-16 ***
d$CFTP      -0.012026   0.003475  -3.461 0.000573 ***
d$ln_MV      0.079630   0.004032  19.751  < 2e-16 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 0.1689 on 677 degrees of freedom
Multiple R-Squared: 0.3666,     Adjusted R-squared: 0.3648
F-statistic: 195.9 on 2 and 677 DF,  p-value: < 2.2e-16
```

Programme 5.5.2: The final regression model after the backward elimination

Although we arrive at the least square regression model:

$$\text{HSI} = -0.454781 - 0.01026(\text{CFTP}) + 0.07963(\text{ln\_MV})\,,$$

it does not mean that this model is useful in telling us whether the company is HSI or not. Firstly, the fitted values of HSI can be any real number rather than 0 and 1. Secondly, we need to look at the residuals of this regression model to check for the validity of the assumptions;

```
> reg <- lm(HSI~CFTP+ln_MV,data=d)      # save regression results
> names(reg)                            # display items in reg
[1] "coefficients"  "residuals"     "effects"       "rank"
[5] "fitted.values" "assign"        "qr"            "df.residual"
[9] "xlevels"       "call"          "terms"         "model"
> par(mfrow=c(3,2))                         # set a 3x2 graphical frame
> hist(reg$fitted.values)                   # fitted values histogram
> hist(reg$residuals)                       # residuals histogram
> plot(reg$fitted.values,reg$residuals) # residuals vs fitted values
> qqnorm(reg$residuals)                     # qq-normal plot of residuals
> qqline(reg$residuals)                     # add reference line
> res <- as.ts(reg$residuals)               # change res to time series
> plot(res,lag(res))                        # residuals vs lag(residuals)
> plot(reg$residuals)                       # residuals vs index number
```

These generate the plots below:

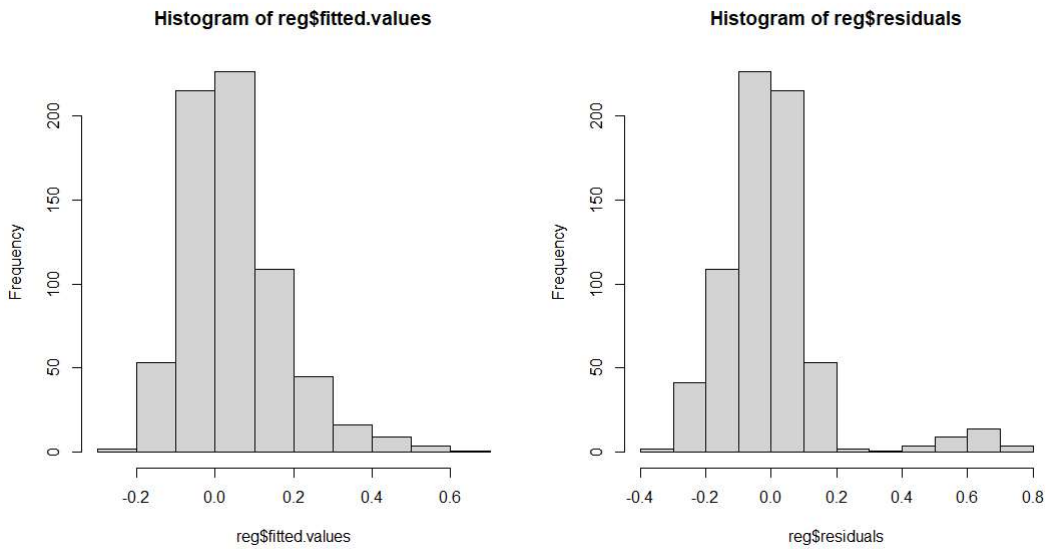



Figure 5.5.1: Histograms of the fitted values (Left) and the residuals (Right).

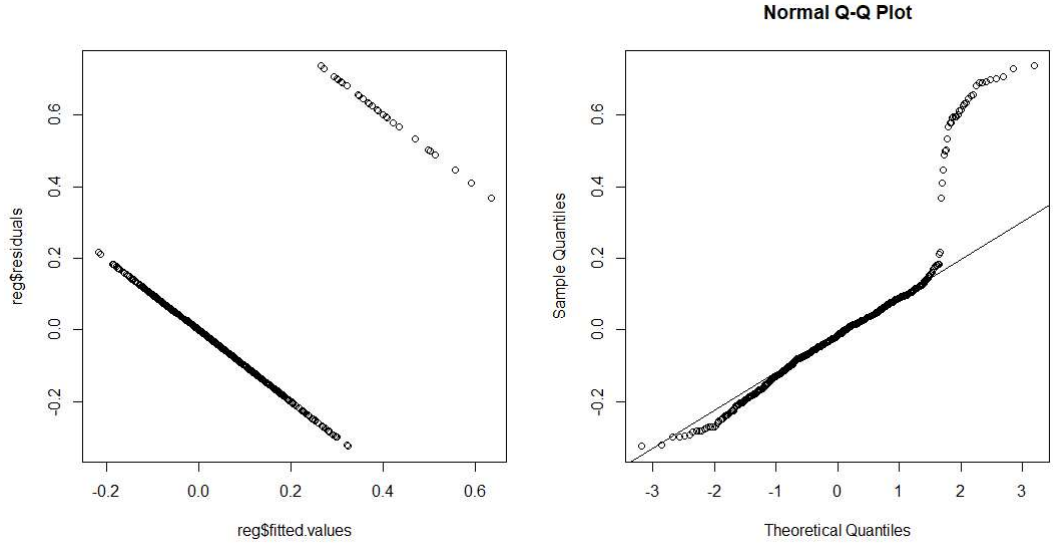



Figure 5.5.2: Plots of residuals vs fitted values (Left) and the normal Q-Q plot (Right).

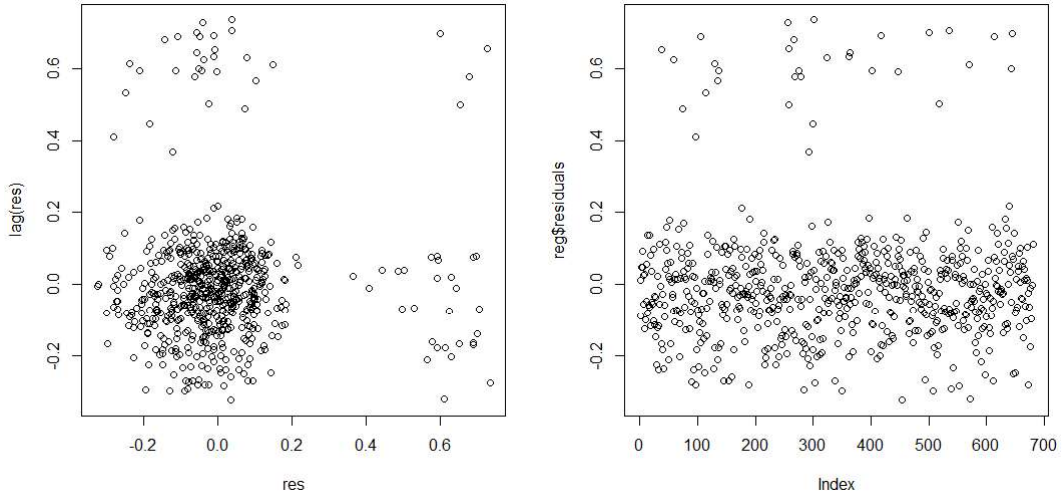



Figure 5.5.3: Residual plots against lag (Left) and against index number (Right).

If the normal assumption is correct, the normal Q-Q plot should be close to a straight line, but instead it is fairly fitted with an indication of heavy tail feature, the histograms of residuals on the right of Figure 5.5.1 should be normally distribted, and the serial residual plots in Figure 5.5.3 should be randomly distributed as a cluster without any patern. Also see Figure 5.5.2, while HSI takes only binary values of 0 or 1, the fitted values of HSI are distributed over a widespread interval. Also, the residuals, while dominantly distribute around 0, has a secondary peak with a large positive value around 0.6, see the right plot of Figure 5.5.3, and this secondary peak of large positive residual values is likely associated with those data having HSI = 1. In addition for the output of Programme 5.5.2, the linear regression assumptions are seriously violated since the adjusted $R^2$ is 0.3648, which is not surprising as the response $y$ (=HSI) is binary.

So far, we have seen the shortcomings of using a simple linear regression model in dealing with binary response variables; meanwhile, the *logistic regression* can be invoked to model the binary response variables. For logistic regression, we first define a parameter

$$\pi_n := \mathbb{P}(y_n = 1|\boldsymbol{x}_n)\,, \tag{5.5.1}$$

which is interpreted as the probability of $y_n = 1$ (probability of "success") given $\boldsymbol{x}_n$. A crucial assumption for the logistic regression model is:

$$\ln\left(\frac{\pi_n}{1-\pi_n}\right) = \beta_0 + \beta_1 x_n^{(1)} + \cdots + \beta_D x_n^{(D)} = \boldsymbol{x}_n^\top \boldsymbol{\beta}\,, \tag{5.5.2}$$

where $\boldsymbol{x}_n = (1, x_n^{(1)}, \cdots, x_n^{(D)}) \in \mathbb{R}^{D+1}$ and $\boldsymbol{\beta} = (\beta_0, \cdots, \beta_D)$. The left hand side of (5.5.2) is the **log-odd ratio** of probability of success, and here we assume that the log-odd ratio of success probability is an affine function in $x_n^{(\cdot)}$'s. The entire (5.5.2) is called the Bernoulli **link function**, the link function is used in the generalized linear model (GLM) setting to relate the linear predictor $\boldsymbol{x}_n^\top\boldsymbol{\beta}$ to the mean of the distribution function, now a Bernoulli one, $\mu = \pi_n$. Equivalently, (5.5.2) can be rewritten as

$$\pi_n = \frac{\exp(\boldsymbol{x}_n^\top\boldsymbol{\beta})}{1+\exp(\boldsymbol{x}_n^\top\boldsymbol{\beta})}\,, \tag{5.5.3}$$

which is called the **expit** transformation of $\boldsymbol{x}_n^\top\boldsymbol{\beta}$, which is also known as the **sigmoid** function in ANN. Obviously, $\pi_n$ always lies between 0 and 1 and this is consistent with the interpretation as a probability of success. Also, (5.5.3) satisfies

$$\nabla_{\boldsymbol{\beta}}\pi_n = \pi_n(1-\pi_n)\boldsymbol{x}_n \in \mathbb{R}^{D+1}\,. \tag{5.5.4}$$

The likelihood function for the data $(\boldsymbol{x}_n, y_n)$, for $n = 1, \cdots, N$, is

$$L(\boldsymbol{\beta}) = \prod_{n=1}^{N}\left(\pi_n^{y_n}(1-\pi_n)^{(1-y_n)}\right), \tag{5.5.5}$$

or the log-likelihood function is

$$\ln L(\boldsymbol{\beta}) = \sum_{n=1}^{N}\Big(y_n \ln\pi_n + (1-y_n)\ln(1-\pi_n)\Big)\,. \tag{5.5.6}$$

Once we obtain the maximum likelihood estimate for $\boldsymbol{\beta}$ by maximizing (5.5.6), we can then compute the predicted probability of success conditional on $\boldsymbol{x}_n$ using (5.5.3); indeed, the gradient of $\ln L(\boldsymbol{\beta})$ in $\boldsymbol{\beta}$ is given

by

$$\begin{aligned}\nabla_{\boldsymbol{\beta}} \ln L(\boldsymbol{\beta}) &= \sum_{n=1}^{N} \left( y_n \frac{\nabla_{\boldsymbol{\beta}} \pi_n}{\pi_n} - (1-y_n) \frac{\nabla_{\boldsymbol{\beta}} \pi_n}{1-\pi_n} \right) \\ &= \sum_{n=1}^{N} \left( y_n \frac{\pi_n(1-\pi_n)}{\pi_n} \boldsymbol{x}_n - (1-y_n) \frac{\pi_n(1-\pi_n)}{1-\pi_n} \boldsymbol{x}_n \right) \\ &= \sum_{n=1}^{N} \left( y_n(1-\pi_n)\boldsymbol{x}_n - (1-y_n)\pi_n \boldsymbol{x}_n \right) = \sum_{n=1}^{N} (y_n - \pi_n)\boldsymbol{x}_n \,.\end{aligned}$$

Generally, it is difficult to look for the MLE by solving for $\nabla_{\boldsymbol{\beta}} \ln L(\boldsymbol{\beta}) = 0$ directly due to the nonlinearity of the expit function in $\boldsymbol{\beta}$. Therefore, Gauss-Newton algorithm is adopted in solving this MLE, which will be introduced in the next subsection.

We first demonstrate the effectiveness of this logistic regression through the HSI example as shown below. To this end, **R** has a built-in `glm()` function (standing for generalized linear model) to perform logistic regression. The output format is quite similar to `lm()`. The argument `family=binomial` indicates that a binomial (Bernoulli) **link function** is used in the logistic regression programmed by `glm()`. Referring to Programme 5.5.3, the MLEs of the coefficients are extremely large, this may be caused by many outliers existing in the dataset. We shall further investigate the MLEs of the coefficients after removing the outliers in Subsection 5.5.3. Anyhow, let us keep these logistic regression output for the moment and see how the model performs.

```
> summary(glm(HSI~EY+CFTP+ln_MV+DY+BTME+DTE,data=d,family=binomial))
Call:
glm(formula = d$HSI ~ d$EY + d$CFTP + d$ln_MV + d$DY + d$BTME + d$DTE,
family = binomial)
Deviance Residuals:
       Min            1Q          Median            3Q          Max
-8.490e+00   -2.107e-08     -2.107e-08   -2.107e-08  8.490e+00
Coefficients:
               Estimate Std. Error    z value Pr(>|z|)
(Intercept) -4.121e+15  1.066e+07 -386410689   <2e-16 ***
d$EY         1.516e+13  6.431e+05   23570628   <2e-16 ***
d$CFTP      -6.364e+13  1.483e+06  -42902735   <2e-16 ***
d$ln_MV      4.945e+14  1.625e+06  304287297   <2e-16 ***
d$DY        -1.144e+14  7.085e+05 -161536188   <2e-16 ***
d$BTME      -7.907e+12  3.155e+05  -25063060   <2e-16 ***
d$DTE        8.744e+12  7.168e+05   12198713   <2e-16 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

```

```
Null deviance:  258.08  on 679  degrees of freedom
Residual deviance: 1153.40  on 673  degrees of freedom
AIC: 1167.4
```

Programme 5.5.3: Fitted coefficients of the first primitive logistic regression for HSI using the 2002 data

Here since all the $p$-values are small, we do not need to eliminate any variables.

```
> lreg <- glm(HSI~EY+CFTP+ln_MV+DY+BTME+DTE,data=d,family=binomial)
> names(lreg)           # display items in lreg
[1] "coefficients"      "residuals"         "fitted.values"
[4] "effects"           "R"                 "rank"
[7] "qr"                "family"           "linear.predictors"
[10] "deviance"          "aic"               "null.deviance"
[13] "iter"              "weights"           "prior.weights"
[16] "df.residual"       "df.null"           "y"
[19] "converged"         "boundary"          "model"
[22] "call"              "formula"           "terms"
[25] "data"              "offset"            "control"
[28] "method"            "contrasts"         "xlevels"

> pr <- (lreg$fitted.values>0.5)            # pr=True if fitted >0.
> table(d$HSI, pr)                       # tabulation of pr & HSI
 pr
  FALSE TRUE
0   634   14
1     2   30
```

Programme 5.5.4: Output and prediction of the first primitive logistic regression for HSI using the 2002 data

Referring to Programme 5.5.4, the `glm()` output is now saved in `lreg`, which contains many items as shown in the output of `names(lreg)`. An important one is the `fitted.values`, which is the estimated success probability in accordance with $\pi_n$ in (5.5.3) for each of these 680 stocks in 2002. All these numbers lie between 0 and 1 and represent the respective probability of HSI = 1 for each stock. Since our MLEs of the coefficients are extremely large, the fitted values are either very close to 0 or 1. Furthermore, we can assign a value of `pr=True` if the fitted value is greater than 0.5 let say, or `pr=False` otherwise. This can be interpreted as the stock being predicted as an HSI constituent stock or not. Finally, we can produce a confusion matrix of pr versus HSI to see how well this logistic model predicts. From the output, there are $634 + 30 = 664$ correct classifications, and $14 + 2 = 16$ misclassifications. There are 2 stocks with HSI = 1 being misclassified as `pr=False` while there are 14 stocks with HSI = 0 being misclassified as `pr=True`. The correct classification rate, *i.e.* the accuracy, is then given by $664/680 = 97.65\%$.

Next, we obtain the prediction `y_pred` by fitting the HSI dataset to the `lreg` model. Then, unlike in Example 4.4.2, we plot the ROC with the `ROCR` library.

```
> library(ROCR)
> y_pred <- predict(lreg, d)
> pred <- prediction(predictions=y_pred, labels=d$HSI)
> perf <- performance(pred, measure="tpr", x.measure="fpr")
> plot(perf, col="blue", lwd=3)
> segments(0, 0, 1, 1, col="red", lwd=3)
```

Programme 5.5.5: Plotting ROC of HSI dataset using the `ROCR` library in **R**

Here both functions `prediction()` and `performance()` return an object defined by the `ROCR` library. In particular, for the `prediction()` function, the parameter `predictions` only accepts continuous attributes, and the parameter `labels` is the true label; meanwhile for the `performance()` function, the arguments `measure="tpr"` indicates that TPR is used in the y-axis and `x.measure="fpr"` indicates that FPR is used in the x-axis. Finally, we plot the ROC curve with a blue line and add a diagonal red line to represent the ROC curve for a random classifier.

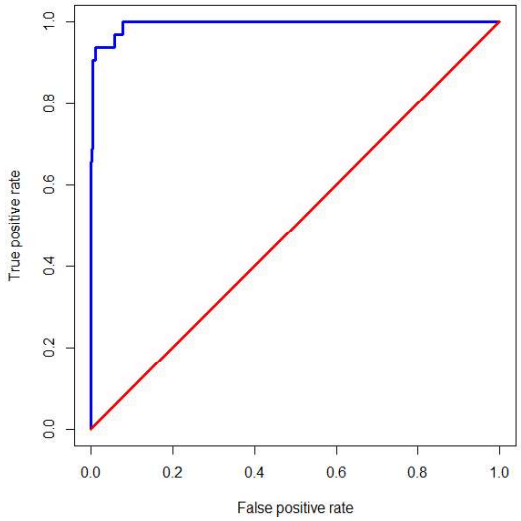



Figure 5.5.4: ROC curve for logistic regression with HSI dataset.

Apart from logistic regression, another commonly used method to tackle binary responses is via perceptron (See Section 7.1), which is essentially logistic regression but with a step function in place of the expit transform, and we here provide a generic comparison between these two methods under some representative cases. Particularly, referring to Figure 5.5.5, the datasets illustrate the classifications of linearly separable (two on the top) and inseparable (another pair at the bottom) ones, using perceptron (represented by the broken line) and logistic regression (represented by the solid line) respectively[4]. The advantage of logistic regression against perceptron is clear in classifying linearly separable data; indeed, though both methods correctly classify the data, the separating line generated by logistic regression tends to separate the two groups of data more evenly, and the one offered by perceptron is slightly off-center, especially shown in the

[4]In both algorithms using IRLS (See Subsection 5.5.2), 30 iterations are performed with a prior termination condition if, for instance in logistic regression, (5.5.9) is satisfied for $\varepsilon = 0.01\%$.

top left plot in Figre 5.5.5. This is due to the fact that the stochastic gradient descent used in perceptron ceases to update the slope parameter $\beta_1$ once all the data are correctly classified, while IRLS in addition picks the very one which maximizes the likelihood.

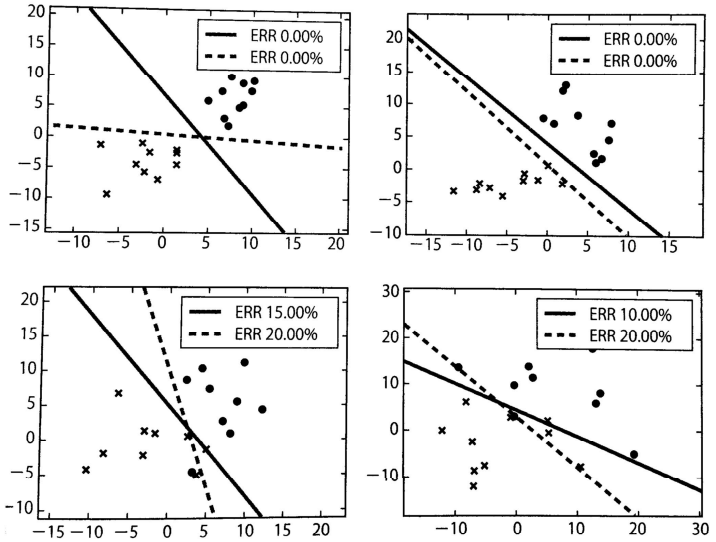



Figure 5.5.5: Comparsions for perceptron (broken line) and IRLS for logistic regression (solid line).

### 5.5.2 Gauss-Newton Algorithm for Logistic Regression: Iteratively Reweighted Least Square (IRLS)

In this section, we shall introduce the Iteratively Reweighted Least Square method based on the Gauss-Newton Algorithm, see Subsection 6.1.2. Denote the log-likelihood $\ln L(\boldsymbol{\beta})$ (as defined in (5.5.6)) by $l(\boldsymbol{\beta})$ for the sake of notational simplicity. With an initial $\boldsymbol{\beta}^{(0)}$, we construct recursively the sequence of approximants with (6.1.18), for $t = 0, 1, \cdots$:

$$\boldsymbol{\beta}^{(t+1)} := \boldsymbol{\beta}^{(t)} - [(\nabla_{\boldsymbol{\beta}} \nabla_{\boldsymbol{\beta}}^\top) l(\boldsymbol{\beta}^{(t)})]^{-1} \nabla_{\boldsymbol{\beta}} l(\boldsymbol{\beta}^{(t)}) = \boldsymbol{\beta}^{(t)} - \boldsymbol{H}^{-1}(\boldsymbol{\beta}^{(t)}) \nabla_{\boldsymbol{\beta}} l(\boldsymbol{\beta}^{(t)}) \,, \tag{5.5.7}$$

where $\boldsymbol{H}(\boldsymbol{\beta})$ is the Hessian matrix of $l(\boldsymbol{\beta})$, which we are about to compute out explicitly. Recall from (5.5.3) and (5.5.4), the logit function and its derivative are respectively given by:

$$\pi_n = \text{expit}(\boldsymbol{x}_n^\top \boldsymbol{\beta}) = \frac{\exp(\boldsymbol{x}_n^\top \boldsymbol{\beta})}{1 + \exp(\boldsymbol{x}_n^\top \boldsymbol{\beta})} \qquad \text{and} \qquad \nabla_{\boldsymbol{\beta}} \pi_n = \pi_n (1 - \pi_n) \boldsymbol{x}_n \in \mathbb{R}^{D+1} \,.$$

Hence for $i, j = 0, 1, \cdots, D$ and $n = 1, \cdots, N$, we have

$$\left. \frac{\partial \pi_n}{\partial \beta_i} \right|_{\boldsymbol{x}_n} = \pi_n (1 - \pi_n) x_{ni} \,,$$

$$\left. \frac{\partial l}{\partial \beta_i}(\boldsymbol{\beta}) \right|_{\boldsymbol{x}_n} = \sum_{n=1}^{N} \left( \frac{y_n}{\pi_n} - \frac{1 - y_n}{1 - \pi_n} \right) \frac{\partial \pi_n}{\partial \beta_i} = \sum_{n=1}^{N} (y_n - \pi_n) x_{ni} \,,$$

$$\left. \frac{\partial^2 l}{\partial \beta_i \partial \beta_j}(\boldsymbol{\beta}) \right|_{\boldsymbol{x}_n} = - \sum_{n=1}^{N} \pi_n (1 - \pi_n) x_{ni} x_{nj} \,,$$

where we also recall that $x_{n0} = 1$ for $n = 1, \cdots, N$. We may then write

$$\nabla_{\boldsymbol{\beta}} l(\boldsymbol{\beta}) = \boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{\pi}) \qquad \text{and} \qquad \boldsymbol{H}(\boldsymbol{\beta}) = -\boldsymbol{X}^\top \boldsymbol{R} \boldsymbol{X} \,,$$

where $\boldsymbol{X} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N)^\top \in \mathbb{R}^{N\times(D+1)}$, $\boldsymbol{R} = \text{diag}[\pi_1(1-\pi_1), \cdots, \pi_N(1-\pi_N)] \in \mathbb{R}^{N\times N}$, $\boldsymbol{y} = (y_1, \cdots, y_N)^\top \in \mathbb{R}^N$, and $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_N)^\top \in \mathbb{R}^N$. Therefore, the iteration (5.5.7) can be rewritten neatly as:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + (\boldsymbol{X}^\top \boldsymbol{R}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{\pi}^{(t)}), \tag{5.5.8}$$

where $\boldsymbol{R}^{(t)}$ and $\boldsymbol{\pi}^{(t)}$ are respectively $\boldsymbol{R}$ and $\boldsymbol{\pi}$ evaluated at $\boldsymbol{\beta}^{(t)}$. We remark that since

$$\pi_n = \text{expit}(\boldsymbol{x}_n^\top \boldsymbol{\beta}) = \frac{\exp(\boldsymbol{x}_n^\top \boldsymbol{\beta})}{1 + \exp(\boldsymbol{x}_n^\top \boldsymbol{\beta})} \in (0, 1), \qquad n = 1, \cdots, N\,.$$

The matrix $\boldsymbol{X}^\top \boldsymbol{R} \boldsymbol{X}$ is positive definite and hence the Hessian $\boldsymbol{H}$ is invertible. The process of finding $\boldsymbol{\beta}$ via (5.5.8) is called *Iteratively Reweighted Least Squares* (**IRLS**). As the change in the values of $\boldsymbol{\beta}^{(t)}$ tends to decrease as the log-likelihood $l(\boldsymbol{\beta})$ (as well as the likelihood $L(\boldsymbol{\beta})$) approaches its global maximum, we may terminate the iteration when the percentage change of $\boldsymbol{\beta}^{(t)}$ is smaller than a prescribed threshold $\varepsilon > 0$, *i.e.* stopping at iteration $t+1$ when

$$\frac{\|\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^{(t)}\|_2}{\|\boldsymbol{\beta}^{(t)}\|_2} < \varepsilon\,. \tag{5.5.9}$$

In the scenarios when computing the inverse of Hessian $\boldsymbol{X}^\top \boldsymbol{R} \boldsymbol{X}$ for each iteration is expensive, supposing that $\boldsymbol{\beta}^{(0)}$ is already quite close to the maximum point sought, we can replace it by an approximation, for instance, $\boldsymbol{H}(\boldsymbol{\beta}^{(t)}) = \boldsymbol{X}^\top \boldsymbol{R}^{(t)} \boldsymbol{X}$ by $\boldsymbol{H}(\boldsymbol{\beta}^{(0)}) = \boldsymbol{X}^\top \boldsymbol{R}^{(0)} \boldsymbol{X}$; such a shortcut approach is called *quasi-Newton method* (see Nocedal and Wright (2006)). Besides, the same method can be also very useful as there could be some limitations, as iterations proceed, some of the $\pi_n^{(t)}$'s approach rapidly to 0 or 1. Their values may then be incorrectly recorded by the computer program as 0 or 1 due to usual rounding procedure. Therefore, the Hessian $\boldsymbol{H}$ becomes singular and the iteration (5.5.8) would then be ill-conditioned; also see Figure 5.5.6.

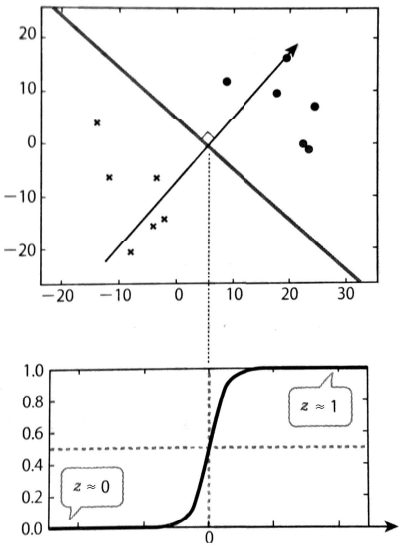



Figure 5.5.6: Causing singularity of $\boldsymbol{H}$ for logistic regression as $\pi_n^{(t)}$ approaching to 0 or 1, where the horizontal axis represents the values taken by $\boldsymbol{x}^\top \boldsymbol{\beta}$, and $z = \text{expit}(\boldsymbol{x}^\top \boldsymbol{\beta})$.

### 5.5.3 Outlier Detection

As pointed out for the output in Programme 5.5.3, the large magnitudes of coefficients could be caused by outliers, we next aim to use the Mahalanobis distance, as defined in (5.5.10), to detect outliers and then remove them, before we fit into a logistic regression learner. To this end, we first introduce a function in **R**,

`mdist()`, to compute the Mahalanobis distance; see Programme 5.5.6. The dataset "fin-ratio.csv" probably contains many outliers, through which we illustrate the effect of outliers. First, we read in the 2002 data and separate the data into two parts, `d0` for HSI = 0 and `d1` for HSI = 1, as shown in Programme 5.5.7. We detect and can only throw away the outliers in `d0`, since `d1` contains only 32 cases, with which we cannot tolerate to lose any more of them.

```
> mdist <- function(x) {
+   t <- as.matrix(x)             # transform x to a matrix
+   m <- apply(t,2,mean)              # compute column mean, 2 stands for column
+   s <- var(t)                   # compute sample cov. matrix
+   mahalanobis(t,m,s)            # built-in mahalanobis func.
+ }
```

Programme 5.5.6: **R**-code for the function `mdist()` in computing the Mahalanobis distance for the whole sample.

```
> d <- read.csv("fin-ratio.csv")          # read in dataset
> d0 <- d[d$HSI==0,]                      # select HSI=0
> d1 <- d[d$HSI==1,]                      # select HSI=1
> dim(d0)
[1] 648   7 # 7 variables = 6 (financial) + 1 (HSI)
> dim(d1)
[1] 32  7
```

Programme 5.5.7: **R**-code for separating the data in "fin-ratio.csv" based on the value of the HSI variable.

```
> x <- d0[,1:6]                           # save d0 to x
> md <- mdist(x)                          # compute mdist
> plot(md)                                # plot md
```

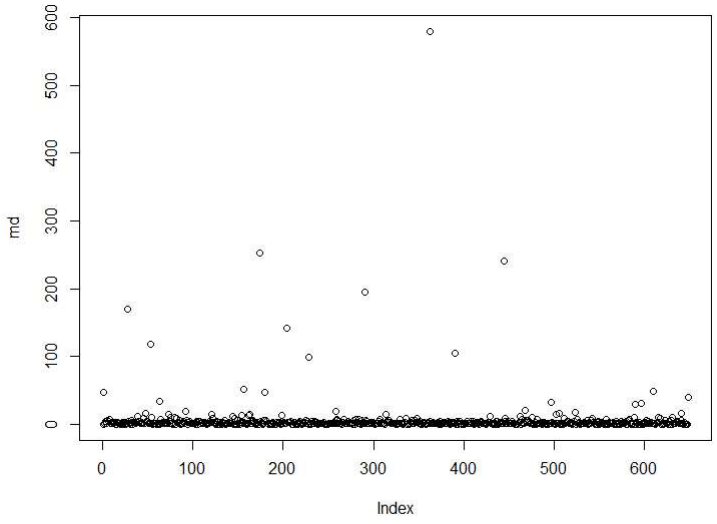



Figure 5.5.7: **R**-code and the resulting plot for the Mahalanobis distance of each stock in `d0`

As shown in Figure 5.5.7, points with a large Mahalanobis distance are the potential outliers which we want to discard. Yet, what is the cut-off value? To this end, we recall that for an iid sample of $\mathbf{X}_1, \cdots, \mathbf{X}_N \in \mathbb{R}^D$ following $\mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, let $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N$ denote their respective sample observations, then the Mahalanobis distance $D_n$ for the datum $\boldsymbol{x}_n$ is given by:

$$D_n^2 = (\boldsymbol{x}_n - \bar{\boldsymbol{x}}_N)^\top \boldsymbol{S}^{-1} (\boldsymbol{x}_n - \bar{\boldsymbol{x}}_N) \sim \chi^2_D, \qquad n = 1, 2, \cdots, N\,, \tag{5.5.10}$$

where $\bar{\boldsymbol{x}}_n$ is the sample mean and $\boldsymbol{S}$ is the observed sample covariance matrix; especially if $N$ is large enough, these $D_n^2$'s are also approximately independent of each other. Based on this, a more scientific way is to inspect the percentiles from a chi-square distribution with degrees of freedom $D$ (= 6 in our "fin-ratio.csv", the six financial variables). We then remove the stocks in `d0` with top 1% values of $D_n$, and combine the resulting class with `d1` to obtain the cleansed dataset `d3`, see Programme 5.5.8.

```
> (c <- qchisq(0.99,df=6))          # p=6, type-I error = 0.01
[1] 16.81189

> d2 <- d0[md<c,]                   # select case in d0 with md<c
> dim(d2)                           # throw away 648-626=22 cases
[1] 626   7
> d3 <- rbind(d1,d2)                    # combine d1 with d2
> dim(d3)
[1] 658   7
> # save the cleansed dataset
> write.csv(d3,file="fin-ratio1.csv",row.names=F)
```

Programme 5.5.8: **R**-code for cleansing the "fin-ratio.csv" dataset by dropping the top 1% outliers.

Next, we try to fit a logistic regression to this cleansed dataset and remove the financial variables with large $p$-value one by one by backward eliminations, and we finally arrive at the following model with the confusion matrix (cross tabulation table) as shown in Programme 5.5.9. The accuracy (correct classification rate) is now increased to $653/658 = 99.24\%$, and throwing away the outliers actually gives a simpler model and some better classification results.

```
> summary(glm(HSI~CFTP+ln_MV+BTME,data=d3,family=binomial))

Call:
glm(formula = HSI ~ CFTP + ln_MV + BTME,
family = binomial, data = d3)

Deviance Residuals:
       Min          1Q      Median          3Q         Max
-2.377e+00  -1.943e-04  -8.005e-06  -3.054e-07   1.738e+00

Coefficients:
```

```
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -69.9309    21.3821  -3.271  0.00107 **
CFTP         -3.0376     1.2178  -2.494  0.01262 *
ln_MV         7.2561     2.2284   3.256  0.00113 **
BTME          1.3222     0.6418   2.060  0.03940 *
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

> lreg <- glm(HSI~CFTP+ln_MV+BTME,data=d3,binomial)
> pr <- (lreg$fit>0.5)           # prediction
> table(pr,d3$HSI)               # classification table
> table(d3$HSI, pr)
 pr
  FALSE TRUE
0   624    2
1     3   29
```

Programme 5.5.9: **R** output and confusion matrix of the first primitive logistic regression on a cleansed dataset in 2002.

### 5.5.4 MM Algorithm and IRLS Algorithm

The first order derivative of (5.5.6) with respect to $\boldsymbol{\beta}$ is

$$\nabla_{\boldsymbol{\beta}} \ln L(\boldsymbol{\beta}) = \sum_{n=1}^{N} (y_n - \pi_n)\boldsymbol{x}_n = \boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{\pi})$$

and that of the second order derivative is

$$(\nabla_{\boldsymbol{\beta}} \nabla_{\boldsymbol{\beta}}^\top) \ln L(\boldsymbol{\beta}) = -\sum_{n=1}^{N} \pi_n(1-\pi_n)\boldsymbol{x}_n\boldsymbol{x}_n^\top .$$

With simple calculus (see Böhning and Lindsay (1988)), we know that with $0 < \pi_n < 1$, we have $\pi_n(1-\pi_n) \leq 1/4$, we may define the negative semi-definite matrix:

$$\boldsymbol{A} = -\frac{1}{4}\sum_{n=1}^{N} \boldsymbol{x}_n\boldsymbol{x}_n^\top = -\frac{1}{4}\boldsymbol{X}^\top\boldsymbol{X} \preceq 0 ,$$

such that $(\nabla_{\boldsymbol{\beta}} \nabla_{\boldsymbol{\beta}}^\top) \ln L(\boldsymbol{\beta}) - \boldsymbol{A} \succeq 0$, *i.e.* a positive semi-definite matrix; see Property 13 in Section 3.3.

Using the lower bound and the concave version of Proposition **??** and noting that $\ln L$ is concave, we have

$$\ln L(\boldsymbol{\beta}) \geq \ln L(\boldsymbol{\beta}^{(t)}) + (\nabla_{\boldsymbol{\beta}} \ln L(\boldsymbol{\beta}^{(t)}))^\top(\boldsymbol{\beta} - \boldsymbol{\beta}^{(t)}) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}^{(t)})^\top\boldsymbol{A}(\boldsymbol{\beta} - \boldsymbol{\beta}^{(t)}) := g(\boldsymbol{\beta}|\boldsymbol{\beta}^{(t)}) ,$$

such that $g(\boldsymbol{\beta}|\boldsymbol{\beta}^{(t)})$ minorizes $L(\boldsymbol{\beta})$ at $\boldsymbol{\beta}^{(t)}$. Consider the first order derivative of $g(\boldsymbol{\beta}|\boldsymbol{\beta}^{(t)})$ with respect to $\boldsymbol{\beta}$ and set it to zero yields:

$$\begin{aligned} & \nabla_{\boldsymbol{\beta}} g(\boldsymbol{\beta}|\boldsymbol{\beta}^{(t)}) = \nabla_{\boldsymbol{\beta}} \ln L(\boldsymbol{\beta}^{(t)}) + \boldsymbol{A}\boldsymbol{\beta} - \boldsymbol{A}\boldsymbol{\beta}^{(t)} = \boldsymbol{0} \\ \Rightarrow \quad & \boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \boldsymbol{A}^{-1}\nabla_{\boldsymbol{\beta}} \ln L(\boldsymbol{\beta}^{(t)}) = \boldsymbol{\beta}^{(t)} + 4(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{\pi}^{(t)}) , \end{aligned}$$

where

$$\pi_n^{(t)} = \frac{\exp(\boldsymbol{x}_n^\top\boldsymbol{\beta}^{(t)})}{1 + \exp(\boldsymbol{x}_n^\top\boldsymbol{\beta}^{(t)})} .$$

In comparison to the logistic regression derived by the celebrate Newton method:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + (\boldsymbol{X}^\top \boldsymbol{R}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^\top (\boldsymbol{y} - \boldsymbol{\pi}^{(t)}),$$

where $\boldsymbol{R}^{(t)} = \text{diag}[\pi_1^{(t)}(1-\pi_1^{(t)}), \cdots, \pi_N^{(t)}(1-\pi_N^{(t)})]$. We immediately observe that MM derivation only computes the inverse of the matrix $\boldsymbol{X}^\top \boldsymbol{X}$ once, but the celebrated Newton method computes the inverse of the matrix $\boldsymbol{X}^\top \boldsymbol{R}^{(t)} \boldsymbol{X}$ for every iteration $t$. The **R** code of the MM algorithm for logistic regression is as below:

```
> MM <- function(X, y, beta_0, tol=1e-8){
+     beta_t <- beta_0
+     beta_old <- beta_t
+     epsilon <- 100
+     inv_X <- solve(t(X)%*%X)
+     num_it <- 1
+     while (epsilon > tol){
+         pi_t <- exp(X%*%beta_t)/(1 + exp(X%*%beta_t))
+         beta_t <- beta_t + 4*inv_X%*%t(X)%*%(y - pi_t)
+         epsilon <- sqrt(sum((beta_t - beta_old)^2))
+         beta_old <- beta_t
+         num_it <- num_it + 1
+     }
+     return (list(beta=beta_t, num_it=num_it))
+ }
```

Programme 5.5.10: MM algorithm for logistic regression in **R**.

The MM algorithm stops until $\|\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^{(t)}\|_2$ is less than some preset tolerance level, for instance, `tol=1e-8`.

Similarly, for IRLS, $\boldsymbol{R}^{(t)}$ may be singular if $\pi_n^{(t)}(1-\pi_n^{(t)}) = 0$ for any $n$. There is a need to introduce a constant term, for instance `1e-2`, to $\pi_n^{(t)}(1-\pi_n^{(t)})$ for $n = 1, \cdots, N$, but $\boldsymbol{R}^{(t)}$ can still be singular when $N$ is large. The **R** code of the IRLS for logistic regression is as below:

```
> Newton <- function(X, y, beta_0, tol=1e-8){
+     beta_t <- beta_0
+     beta_old <- beta_t
+     epsilon <- 100
+     num_it <- 1
+     while (epsilon > tol){
+         pi_t <- exp(X%*%beta_t)/(1 + exp(X%*%beta_t))
+         R_t <- diag(as.vector(pi_t * (1 - pi_t) + 1e-2))
+         beta_t <- beta_t + solve(t(X)%*%R_t%*%X)%*%t(X)%*%(y - pi_t)
+         epsilon <- sqrt(sum((beta_t - beta_old)^2))
+         beta_old <- beta_t
+         num_it <- num_it + 1
+     }
```

```
+     return (list(beta=beta_t, num_it=num_it))
+ }
```

Programme 5.5.11: IRLS for logistic regression in **R**.

Finally, we use the cleansed HSI dataset in Programme 5.5.8 as an illustration. The sample size is $N = 658$ such that a constant term of `1e-2` is enough to ensure $\boldsymbol{R}^{(t)}$ is non-singular for all $t$. We include the intercept term to the feature vector $\boldsymbol{X}$ and $\boldsymbol{\beta}^{(0)}$ is initialized with the independent standard normal random variables.

```
> set.seed(4012)
> d <- read.csv("fin-ratio1.csv")
> X <- as.matrix(d[,-ncol(d)])
> X <- cbind(rep(1, nrow(X)), X)
> y <- as.vector(d[, ncol(d)])
> beta_0 <- rnorm(ncol(X))
>
> glm(HSI~EY+CFTP+ln_MV+DY+BTME+DTE,data=d,family=binomial)$coef
 (Intercept)            EY          CFTP         ln_MV            DY          BTME
-76.03491920   1.12561595  -3.88639187   7.82355732   0.16985350   1.48319289
         DTE
  0.06199068
> system.time(print(MM(X, y, beta_0)))
$beta
              [,1]
      -76.0347844
EY      1.1256133
CFTP   -3.8863841
ln_MV   7.8235436
DY      0.1698528
BTME    1.4831900
DTE     0.0619901

$num_it
[1] 161768

 user  system elapsed
15.19    0.03   15.37
> system.time(print(Newton(X, y, beta_0)))
$beta
               [,1]
      -76.03492533
EY      1.12561607
CFTP   -3.88639222
ln_MV   7.82355794
```

```
DY       0.16985353
BTME     1.48319301
DTE      0.06199071

$num_it
[1] 10931

 user  system elapsed
28.92    6.56   35.69
```

Programme 5.5.12: MM algorithm using Programme 5.5.10 and IRLS using Programme 5.5.11 with HSI dataset in **R**.

Although the number of iteration for IRLS is far less than that of MM algorithm, we do observe a much faster, at least two times faster, convergence for MM algorithm than that of IRLS to the preset tolerance level `tol=1e-8`, it is because we only need to compute the inverse once in MM algorithm, but we need to compute the inverse of the matrix $\boldsymbol{X}^\top \boldsymbol{R}^{(t)} \boldsymbol{X}$ for all $t$.

# *BIBLIOGRAPHY*


[1] Harrison Jr, D., and Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air. Journal of environmental economics and management, 5(1), 81-102.

[2] Loader, C. (2006). *Local regression and likelihood*. Springer Science & Business Media.

[3] Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1), 141-142.

[4] Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, 359-372.

[5] Rosenblatt, M. (1956). Remarks on Some Nonparametric Estimates of a Density Function. *Annals of Mathematical Statistics*, 27(3), 832-837.

[6] Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3), 1065-1076.

[7] Rice, J. A. (2006). *Mathematical statistics and data analysis*. Nelson Education.

[8] Fan, J., and Gijbels, I. (2003). *Local polynomial modelling and its applications*. Boca Raton: CRC Press.

[9] Nocedal, J., and Wright, S. (2006). *Numerical optimization*. New York: Springer.

[10] Böhning, D., and Lindsay, B. G. (1988). Monotonicity of quadratic-approximation algorithms. Annals of the Institute of Statistical Mathematics, 40(4), 641-663.