Imbalanced Classification and Sparsity

MATH 480: Honors Independent Study

Winter 2026

Author: Jiajun Zhang
Supervisor: Professor Abbas Khalili

Department of Mathematics and Statistics, McGill University
April 2026

Contents

Part I
Introduction

This report was written in the Winter 2026 semester as part of my undergraduate Honors independent study at McGill University, under Professor Abbas Khalili's supervision. This report serves as an introduction and servey to classical classification techniques and their applications in the setting of imbalance classification. Below is a list of topics I included in this article:

- **Chapter 1**: **Linear Classifiers**. This section introduces two classical linear classification, namely Linear Discriminant Analysis (LDA), and Separable Support Vector Machines (SSVMs). The fundamental idea for both methods are different: LDA uses a Bayesian-like approach by assuming the data is multivariate normal, and the decision boundary is obtained by equating the conditional likelihood equations of different classes. On the other hand, SSVMs aims to find the best separable hyperplane by maximizing the distance from each data to the plane.

- **Chapter 2: Non-linear Classifiers**. This section extends the idea from chapter one into a more general case. The methods we introduce are Non-separable Support Vector Machines (NSVMs), Quadratic Discriminant Analysis (QDA), and Logistic Regression. In QDA, we make similar assumption as in LDA, but assuming the covariance matrix for each class is different. By doing so, equating the conditional likelihood will result in a quadratic decision boundary. In NSVMs, the we use the idea from functional analysis and construct a feature map, so the non-separable data is mapped into a higher dimensional space where separation is possible. Finally, we briefly introduce logistic regression as another tool to model data.

- **Chapter 3: Tree-based Classification Algorithms**. In this section, we introduce a wide range of classification algorithms using a decision tree structure. The algorithms we consider are ID3 algorithm, CART algorithm, Bagging, Random Forest and Boosting. They are discussed in detail in Section 3.

- **Chapter 4: Neural Network Methods**. In this section, we introduce a brand-new data structure called neural networks from modern machine learning techniques. We provide the basic set up of a multi-layer, feed-forward, fully-connected neural network and discuss the methods to train neural network parameters.

- **Chapter 5: Classification Paradigms and Evaluations**. In this section, we will discuss the problem of class imbalance, and how to evaluate a classification algorithm based on different metrics. We first introduce three different classification paradigms based on different user needs, then we introduce the evaluation metrics, like the TPR/TNR rate. We finally introduce the SMOTE algorithm and its variant which has better performance in a imbalanced setting.

- **Chapter 6: Simulation Study**. In this section, we apply the theory from previous chapters and use simulated data as well as real data set to examine different classification methods under different settings. This concludes our report.

I sincerely wish the reader finds this article both informative and inspiring.

Part II
An Overview of Classical Classification Techniques

1 Linear Classifiers

1.1 Linear Discriminant Analysis (LDA)

The idea of linear discriminant analysis is to assume the conditional distribution of data $\boldsymbol{x} \in \mathbb{R}^p$ given the class label $y_i = c$, $c = 1, \cdots, k$ forms a multivariate normal distribution with mean $\boldsymbol{\mu}_c$ and common covariance matrix $\boldsymbol{\Sigma}$. That is, we assume

$$\boldsymbol{X} = \boldsymbol{x} | Y = c \sim f_c(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2} \cdot |\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_c) \right\}.$$

We may further assume a prior distribution on $p_c = \mathbb{P}(Y = c), c = 1, \cdots, k$. Then by Bayes theorem, the posterior distribution, $\mathbb{P}(Y = c | \boldsymbol{X} = \boldsymbol{x})$, will satisfy

$$\mathbb{P}(Y = c | \boldsymbol{X} = \boldsymbol{x}) \propto p_c \cdot f_c(\boldsymbol{x}).$$

If we assume fixed-design and homoskedasticity, the covariance matrix $\boldsymbol{\Sigma}$ will be constant, and further we may write the posterior probability as

$$\mathbb{P}(Y = c | \boldsymbol{X} = \boldsymbol{x}) = C p_c \cdot \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_c) \right\}, \tag{1.1}$$

for some constant C. The log posterior probability of (1.1) is then given by

$$\begin{aligned}
\log \mathbb{P}(Y = c | \boldsymbol{X} = \boldsymbol{x}) &= \log C + \log p_c - \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_c) \\
&= \log C + \log p_c - \frac{1}{2} \left[\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{x} - 2 \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c + \boldsymbol{\mu}_c \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c \right] \\
&= C' + \log p_c - \frac{1}{2} \boldsymbol{\mu}_c^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c,
\end{aligned}$$

where C' is a new constant that contains all the terms independent of the class c. From here, we define the *discriminant functions* as

$$\delta_c(\boldsymbol{x}) = \log p_c - \frac{1}{2} \boldsymbol{\mu}_c^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c, \quad c = 1, \cdots, k. \tag{1.2}$$

Hence the classification rule is simply compute each $\delta_c(\boldsymbol{x})$ and assign Y the label c' such that $\delta_{c'}(\boldsymbol{x})$ is maximized, i.e.,

$$\widehat{Y} = \arg\max_c \delta_c(\boldsymbol{x}) := \arg\max_c \mathbb{P}(Y = c | \boldsymbol{X} = \boldsymbol{x}).$$

The *decision boundary* is the set of points $\boldsymbol{x}$ such that the discriminant functions of two classes agree at $\boldsymbol{x}$. With k classes, we are able to get $\binom{k}{2}$ different decision boundaries. For example, the decision boundary between class l and m are given by the linear equation $\delta_l(\boldsymbol{x}) = \delta_m(\boldsymbol{x})$:

$$\log p_l - \frac{1}{2}\boldsymbol{\mu}_l^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_l + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_l = \log p_m - \frac{1}{2}\boldsymbol{\mu}_m^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_m + \boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_m. \tag{1.3}$$

Then the solution to (1.3) is given by

$$(\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_l - \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_m)^\top \boldsymbol{x} = \log \frac{p_m}{p_l} - \frac{1}{2}\left(\boldsymbol{\mu}_m^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_m - \boldsymbol{\mu}_l^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_l\right).$$

In practice, we often do not have access to the true probability density $f_c(\boldsymbol{x})$ and $p_c(\boldsymbol{x})$. We would replace them by their estimates based on the data we have. The in-class mean $\boldsymbol{\mu}_c$, prior probability $p_c(\boldsymbol{x})$, and the covariance matrix $\boldsymbol{\Sigma}$ can be estimated by

$$\widehat{\boldsymbol{\mu}}_c = \frac{1}{n_c}\sum_{i:y_i=c} \boldsymbol{x}_i, \quad \widehat{p}_c(\boldsymbol{x}) = \frac{n_c}{n}, \quad \widehat{\boldsymbol{\Sigma}} = \frac{1}{n-k}\sum_{c=1}^{k}\sum_{i:y_i=c} (\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_c)\cdot(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_c)^\top,$$

hence the estimated decision boundary in (1.3) can be written as $\widehat{\delta}_l(\boldsymbol{x}) = \widehat{\delta}_m(\boldsymbol{x})$:

$$(\widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{\mu}}_l - \widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{\mu}}_m)^\top \boldsymbol{x} = \log \frac{\widehat{p}_m}{\widehat{p}_l} - \frac{1}{2}\left(\widehat{\boldsymbol{\mu}}_m^\top \widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{\mu}}_m - \widehat{\boldsymbol{\mu}}_l^\top \widehat{\boldsymbol{\Sigma}}^{-1}\widehat{\boldsymbol{\mu}}_l\right),$$

which takes the form of a linear function

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0.$$

Apart from linear discriminant analysis, quadratic discriminant analysis (QDA) is also widely used. The set up is similar to LDA, however we assume hetroskedasticity where the covariance matrix depends on the class label c:

$$\boldsymbol{X} = \boldsymbol{x}|Y = c \sim f_c(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}\cdot|\boldsymbol{\Sigma}_c|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_c)^\top \boldsymbol{\Sigma}_c^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_c)\right\}.$$

Then the discriminant function is quadratic:

$$\begin{aligned}\delta_c(\boldsymbol{x}) &= \log p_c - \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}_c^{-1}\boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{\Sigma}_c^{-1}\boldsymbol{\mu}_c - \frac{1}{2}\boldsymbol{\mu}_c \boldsymbol{\Sigma}_c^{-1}\boldsymbol{\mu}_c \\ &= \boldsymbol{x}^\top A\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x} + \boldsymbol{c}.\end{aligned}$$

The within-class covariance matrix can be estimated by

$$\widehat{\boldsymbol{\Sigma}}_c^{-1} = \frac{1}{n_c - 1}\sum_{i:y_i=c} (\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_c)\cdot(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_c)^\top,$$

Finally we have the estimated decision boundary $\widehat{\delta}_l(\boldsymbol{x}) = \widehat{\delta}_m(\boldsymbol{x})$ between class l and m given by

$$\frac{1}{2}\boldsymbol{x}^\top\left(\widehat{\boldsymbol{\Sigma}}_l^{-1} - \widehat{\boldsymbol{\Sigma}}_m^{-1}\right)\boldsymbol{x} + \boldsymbol{x}^\top \widehat{\boldsymbol{\Sigma}}_m^{-1}\widehat{\boldsymbol{\mu}}_m - \boldsymbol{x}^\top \widehat{\boldsymbol{\Sigma}}_l^{-1}\widehat{\boldsymbol{\mu}}_l - \frac{1}{2}\widehat{\boldsymbol{\mu}}_m \widehat{\boldsymbol{\Sigma}}_m^{-1}\widehat{\boldsymbol{\mu}}_m + \frac{1}{2}\widehat{\boldsymbol{\mu}}_l \widehat{\boldsymbol{\Sigma}}_l^{-1}\widehat{\boldsymbol{\mu}}_l + \log\frac{\widehat{p}_m}{\widehat{p}_l} = 0.$$

We next present a study on `wine` dataset from **R**. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars (labeled cultivar 1 (lack), 2 (pink), 3 (green) in Figure 1 shown on the next page). The analysis determined the quantities of 13 constituents found in each of the three types of wines. We perform LDA and QDA based on six groups of binary covariates: Alcohol (V1) and Malic Acid (V2); Ash (V3) and Total phenols (V6); Alcalinity of ash (V4) and Nonflavanoid phenols (V8); Magnesium (V5) and Flavanoids (V7); Proanthocyanins (V9) and Color intensity (V10), Hue (V11) and Proline (V13). The data points are plotted in Figure 1, where different colors refer to different classes (cultivars). The LDA decision boundaries are shown in black, and the QDA decision boundaries are shown in red.

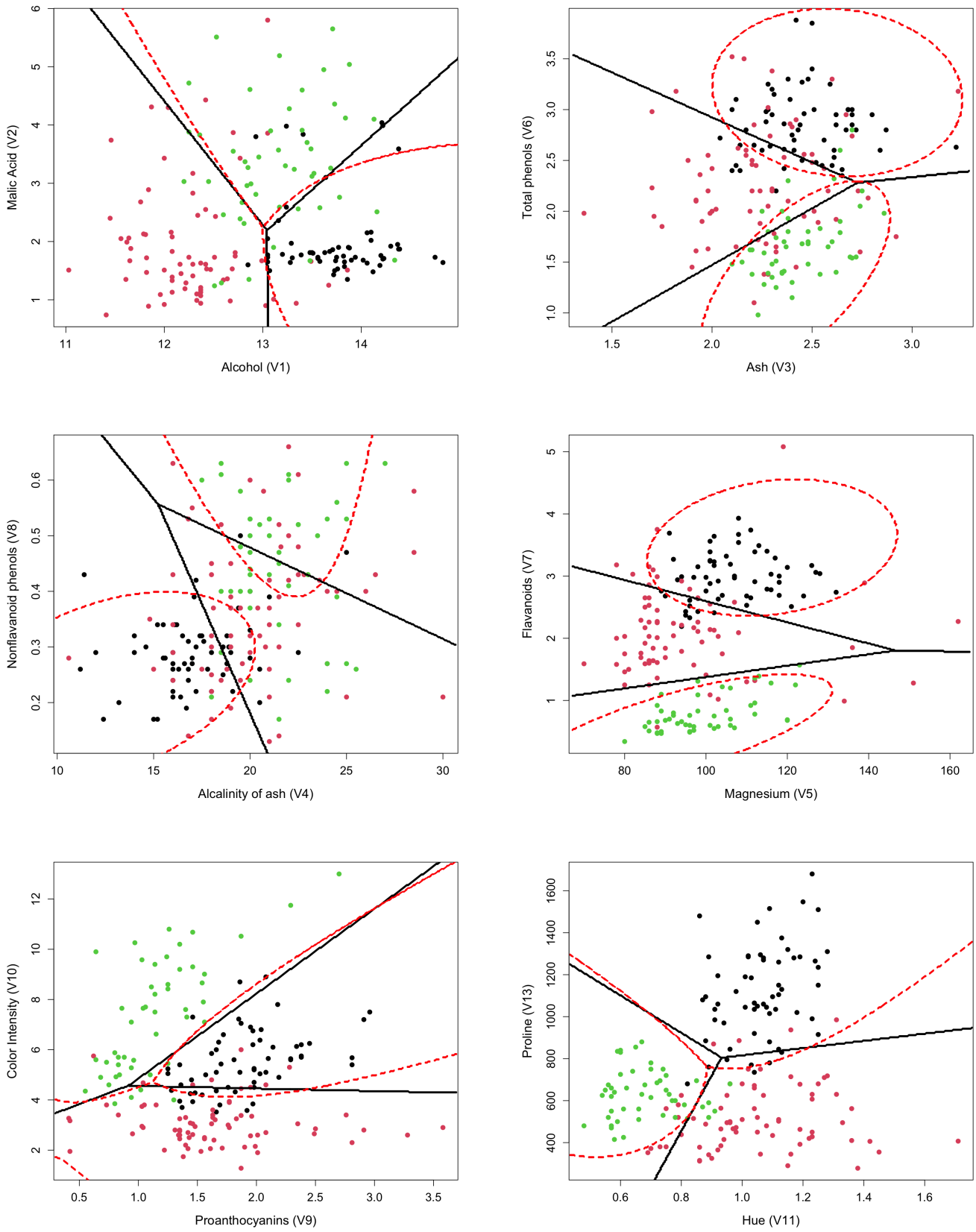

Figure 1: The decision boundaries of LDA and QDA based on different groups of binary covariates of wines from three cultivars.

1.2 Support Vector Machines (SVMs): Separable Case

We again consider a binary classification problem with data $\mathcal{D}_n := \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i$ is the p-dimensional covariates and $y_i \in \{-1, 1\}$ is binary. Support vector machines (SVMs) is another linear classifier used for classification, but the development of SVMs is different from LDA. Unlike LDA, SVMs do not assume any prior distribution on the data, but instead try to find the optimal p-dimensional vector $\boldsymbol{w}$ and scalar b such that

$$\begin{cases} \boldsymbol{x}^\top \boldsymbol{w} + b > 0 & \text{if } y = 1 \\ \boldsymbol{x}^\top \boldsymbol{w} + b < 0 & \text{if } y = -1 \end{cases},$$

and the decision boundary is given by the equation $\boldsymbol{x}^\top \boldsymbol{w} + b = 0$. In many cases, we are able to find multiple linear functions that perfectly separate the two classes. Apart from separability, we also want to maximize the closest distance to the linear function in each class, so we are able to make more accurate predictions for data points near the decision boundary. See the two examples shown in Figure 2

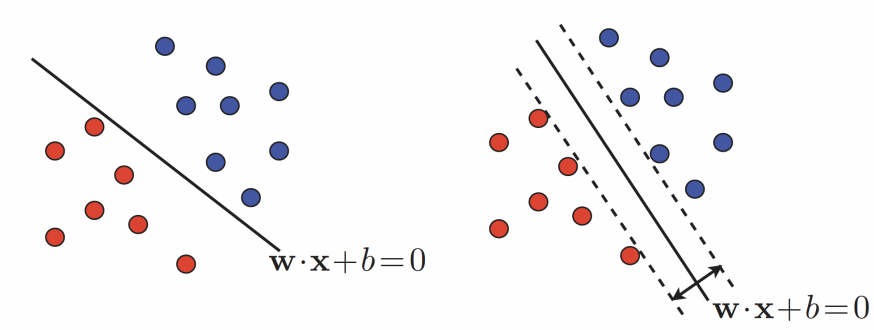

Figure 2: With the data set given, two linear functions are proposed. Both classifies the data well, but the one on the right has maximal separability. The figure is taken from Mohri et. al. [MRT12]

For any point $(\boldsymbol{x}_i, y_i) \in \mathcal{D}_n$, it is easy to see that the distance of $\boldsymbol{x}_i$ to the linear function $\boldsymbol{x}^\top \boldsymbol{w} + b = 0$ is given by

$$d_{\boldsymbol{x}_i} = \frac{|\boldsymbol{x}_i^\top \boldsymbol{w} + b|}{||\boldsymbol{w}||}.$$

For computational purposes, we may scale $\boldsymbol{w}, b$ such that $\min_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_n} |\boldsymbol{x}_i^\top \boldsymbol{w} + b| = 1$. Hence the shortest distance from a data point to $\boldsymbol{x}^\top \boldsymbol{w} + b = 0$ is simply 1. So for any data $\boldsymbol{x}_i$, we always have $|\boldsymbol{x}_i^\top \boldsymbol{w} + b| \geq 1$, and the minimum distance is given by $d_{\min} = 1/||\boldsymbol{w}||$. The *margin* (shortest distance between data from two different classes) is just $2/||\boldsymbol{w}||$. Then the SVM finds the optimal classifier such that the margin is maximized while correctly classifying all the given data. i.e., we solve the optimization problem

$$\min \frac{1}{2}||\boldsymbol{w}||^2 \quad \text{subject to } y_i \cdot (\boldsymbol{x}_i^\top \boldsymbol{w} + b) \geq 1, \tag{1.4}$$

for all $i = 1, \cdots, n$. We transform (1.4) to its Lagrangian:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{\alpha}) = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_{i=1}^{n} \alpha_i [y_i \cdot (\boldsymbol{x}_i^\top \boldsymbol{w} + b) - 1], \tag{1.5}$$

where $\alpha_i \geq 0, i = 1, \cdots, n$ are the Lagrange variables. The term (1.5) is not easy to optimize, so we usually introduce its dual problem given by

$$\mathcal{L}_{\text{dual}}(\boldsymbol{\alpha}) = \max_{\alpha} \min_{\boldsymbol{w}, b} \mathcal{L}(\boldsymbol{w}, b, \alpha).$$

Note that the original optimization problem in (1.4) is strictly convex, so we can derive the unique minimizer of $\boldsymbol{w}$ and b by taking partial derivatives of the equation (1.5) with respect to $\boldsymbol{w}, b$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0, \quad \frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0,$$

and equate them to zero. Hence we have

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i, \quad \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{1.6}$$

We plug (1.6) into (1.5), and the dual problem is now given by

$$\mathcal{L}_{\text{dual}}(\boldsymbol{\alpha}) = \frac{1}{2} \left|\left| \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \right|\right|^2 - \sum_{I=1}^{n} \alpha_i \left[y_i \cdot \left(\boldsymbol{x}_i^\top \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i + \sum_{i=1}^{n} \alpha_i y_i \right) - 1 \right].$$

After simplification, the optimization problem now is simply

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i^\top \boldsymbol{x}_j), \tag{1.7}$$

subject to $\alpha_i \geq 0 \wedge \sum_{i=1}^{n} \alpha_i y_i = 0$ for all $i = 1, \cdots, n$. The optimization problem is now a quadratic programming (QP) problem, and it can be solved by sequential minimal optimization (SMO) introduced later. Thus the solution suggested by SVMs is given by

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w} + b = \sum_{i=1}^{n} \alpha_i y_i (\boldsymbol{x}_i^\top \boldsymbol{x}) + b,$$

where b can be obtained via the support vector $\boldsymbol{x}_j$ such that $\boldsymbol{x}_j^\top \boldsymbol{w} + b = y_j$:

$$b = y_j - \sum_{i=1}^{n} \alpha_i y_i (\boldsymbol{x}_i^\top \boldsymbol{x}_j),$$

and it can be used directly to determine the hypothesis returned by SVMs, by comparing y_i and $\text{sgn}(f(\boldsymbol{x}_i))$.

1.3 Fisher's Discriminant Analysis

To achieve dimension reduction, our goal is to project our data $\boldsymbol{x}_i \in \mathbb{R}^p$ onto a smaller dimensional subspace $\mathbb{R}^m, m < p$ with the same label. Assume the smaller dimensional subspace can be spanned by vector $\boldsymbol{v}$, and without loss of generality we assume $||\boldsymbol{v}|| = 1$. The problem

now requires to find the optimal directional vector v such that the projections of x_i onto v can be well separated among different classes.

For simplicity, we first consider a dimensional reduction problem with two classes. The idea is to project each data point (x_i, y_i) onto the threshold function $x(t) = t \cdot v$. It can be shown that the projection of x_i onto the hyperplane is given by $a_i := v^\top x_i$, with the same label y_i.

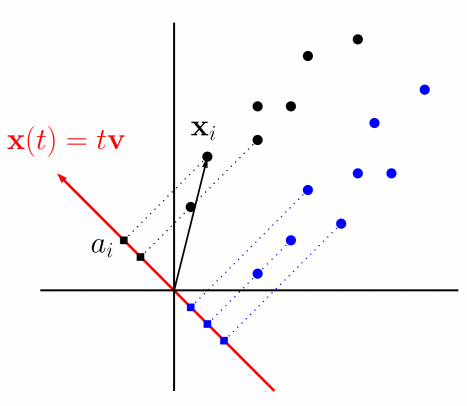

Figure 3: The geometric view of projecting each data point to the threshold function where each color represents a different class. The figure is taken from Chen [Che20]

Let $\mathcal{C}_1, \mathcal{C}_2$ be two different classes, n_i be the number of data which belong to class i ($i = 1, 2$). After projecting our data to the threshold function, we aim to measure the concentration of the data within the same class, and the separation of the data between different classes. We define the *with-class mean* by

$$\mu_1 = \frac{1}{n_1} \sum_{x_i \in \mathcal{C}_1} v^\top x_i = v^\top m_1, \quad \mu_2 = \frac{1}{n_2} \sum_{x_i \in \mathcal{C}_2} v^\top x_i = v^\top m_2, \tag{1.8}$$

and our first goal is to maximize $|\mu_1 - \mu_2|$. The larger the quantity is, the larger the separation between two classes occurs.

In the meantime we also would like to minimize the variance of data within each class, and we define the *in-class variance* by

$$s_1^2 = \sum_{x_i \in \mathcal{C}_1} (a_i - \mu_1)^2, \quad s_2^2 = \sum_{x_i \in \mathcal{C}_2} (a_i - \mu_2)^2, \tag{1.9}$$

and our second goal is to minimize both s_1^2 and s_2^2. The smaller the quantity is, the larger the concentration within one class occurs. From (4.1), it follows that

$$\begin{aligned}(\mu_1 - \mu_2)^2 &= (v^\top m_1 - v^\top m_2) \cdot (v^\top m_1 - v^\top m_2) \\ &= v^\top (m_1 - m_2) \cdot (m_1 - m_2)^\top v \\ &:= v^\top S_b v.\end{aligned}$$

The matrix $S_b := (m_1 - m_2) \cdot (m_1 - m_2)^\top \in \mathbb{R}^{p \times p}$ is called the *between-class scatter matrix*. It

is clear that S_b is symmetric and positive definite. From (4.2), for $j = 1, 2$, we also have

$$\begin{aligned} s_j^2 &= \sum_{x \in \mathcal{C}_j} (v^\top x_i - v^\top m_j)^2 \\ &= v^\top \left[\sum_{x_i \in \mathcal{C}_j} (x - m_j) \cdot (x_i - m_j)^\top \right] v \\ &= v^\top S_j v. \end{aligned} \tag{1.10}$$

The matrix $S_j := \sum_{x_i \in \mathcal{C}_j} (x - m_j) \cdot (x_i - m_j)^\top \in \mathbb{R}^{p \times p}$ is called the *within-class scatter matrix*. The *total within-class scatter matrix* is defined by $S_w = S_1 + S_2$, which is also symmetric and positive definite. Hence

$$s_1^2 + s_2^2 = v^\top (S_1 + S_2) v. \tag{1.11}$$

To maximize $|\mu_1 - \mu_2|$ and minimize s_1^2, s_2^2, we now introduce the optimization problem

$$v^* := \arg \max_{||v||=1} \frac{v^\top S_b v}{v^\top S_w v}. \tag{1.12}$$

Theorem 1. *Suppose S_w is invertible, then the maximizer v^* of (??) is given by the eigenvector corresponds to the largest eigenvalue v_1 of $S_w^{-1} S_b$.*

Proof. □

The idea for multi-classes $\mathcal{C}_1, \cdots, \mathcal{C}_k$ are the same. We wish to find $k - 1$ directional vectors such that the projected mean onto any vector between any two classes are maximized. For each class j, $j = 1, \cdots, k$, the *within-class scatter matrix* follows the same definition as before:

$$S_j = \sum_{x_i \in \mathcal{C}_j} (x - m_j) \cdot (x - m_j)^\top, \quad m_j = \frac{1}{n_j} \sum_{x_i \in \mathcal{C}_j} x_i,$$

and the *total within-class scatter matrix* is simply

$$S_w = S_1 + \cdots + S_k.$$

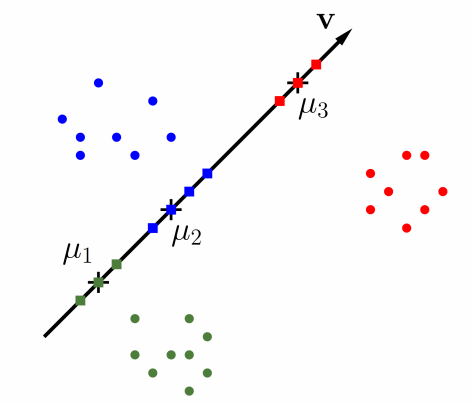

Figure 4: The geometric view of projections with three different classes. μ_i represents the projected mean, and v is one of the two directional vectors. The figure is taken from Chen [Che20].

To measure the separability among different classes, we use the same definition for within-class mean as before: For $j = 1, \cdots, k$,

$$\mu_j = \frac{1}{n_j} \sum_{x_i \in \mathcal{C}_j} v^\top x_i := v^\top m_j.$$

The weighted average is defined by

$$\mu = \frac{1}{n_1 + \cdots + n_k} \sum_{j=1}^{k} n_j \mu_j := v^\top m.$$

Finally the between-class scatter quantity is given by

$$\begin{aligned} \sum_{j=1}^{k} n_j (\mu_j - \mu)^2 &= \sum_{j=1}^{k} n_j (v^\top (m_j - m))^2 \\ &= v^\top \cdot \left[\sum_{j=1}^{k} n_j (m_j - m) \cdot (m_j - m)^\top \right] \cdot v \\ &'' = v^\top S_b v, \end{aligned}$$

where $S_b := \sum_{j=1}^{k} n_j (m_j - m) \cdot (m_j - m)^\top$ is the *between-class scatter matrix*. The optimization problem is eventually the same as (1.12):

$$v^* := \arg \max_{||v||=1} \frac{v^\top S_b v}{v^\top S_w v},$$

and the solutions are given by the eigenvectors with the largest eigenvalues. It suffices to solve

$$S_w^{-1} S_b v = \lambda v.$$

Note that when $k = 2$, we have

$$\mu = \frac{1}{n_1 + n_2}(n_1 \mu_1 + n_2 \mu_2), \quad m = \frac{1}{n}(n_1 m + n_2 m_2).$$

Despite S_w is invertible, it can be shown that $\text{rank}(S_b) \leq k - 1$, hence one can find at most $k - 1$ eigenvectors as our solution. The figure below illustrates the three-class case in Figure 4:

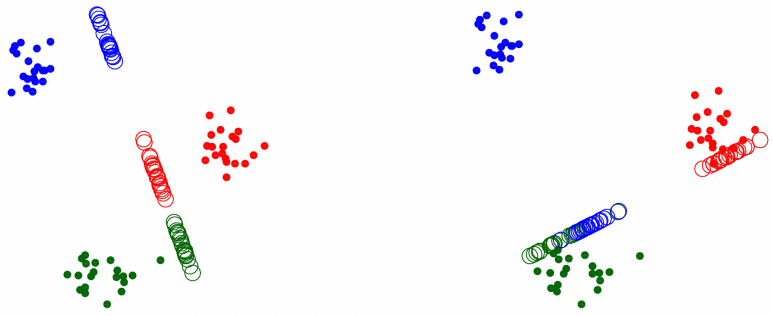

Figure 5: The multi-class case with $k = 3$. We find 2 eigenvectors as the solution, the threshold functions are simply the span for each eigenvector, as illustrated in the figure. The figure is taken from Chen [Che20].

We summarize the general procedure of linear discriminant analysis as the following algorithm:

Algorithm 1 Multiclass LDA

Require: Design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ with k classes $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$.
Ensure: At most $k-1$ threshold functions and projections of $\boldsymbol{X}$ onto them.
1: Compute the between-class and total within-class scatter matrix:

$$\boldsymbol{S}_w = \sum_{j=1}^{k} \sum_{\boldsymbol{x}_i \in \mathcal{C}_j} (\boldsymbol{x}_i - \boldsymbol{m}_j) \cdot (\boldsymbol{x}_i - \boldsymbol{m}_j)^\top, \quad \boldsymbol{S}_b = \sum_{j=1}^{k} n_j (\boldsymbol{m}_j - \boldsymbol{m}) \cdot (\boldsymbol{m}_j - \boldsymbol{m})^\top.$$

2: Solve the generalized eigenvalue problem $\boldsymbol{S}_b \boldsymbol{v} = \lambda \boldsymbol{S}_w \boldsymbol{v}$ to find all non-zero eigenvectors $\boldsymbol{V}_m := [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m]$ for some $m \leq k-1$.
3: Project the data $\boldsymbol{X}$ onto them: $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{V}_m \in \mathbb{R}^{n \times m}$.

2 Non-linear Classifiers

2.1 Support Vector Machines (SVMs): Non-separable case

2.1.1 Kernel Methods

We now return to the binary classification problem using support vector machines discussed in Section 1.2. It is worth to note that separability is not always ideal, since in many cases we would have $y_i \cdot (\boldsymbol{x}_i^\top \boldsymbol{w} + b) \ngeq 1$ for any linear function, as shown in Figure 6.

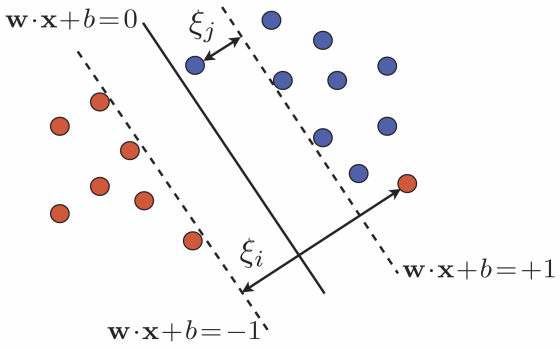

Figure 6: The case when the data can not be fully separated by a linear function. The figure is taken from Mohri et. al. [MRT12].

Under this situation, we may also want to minimize the "error" made by the linear function ξ_i, which measures the separation quantity of a data point from the decision boundary. The optimization problem in this case is given by

$$\min_{\boldsymbol{w}, b, \boldsymbol{x}_i} \frac{1}{2} ||\boldsymbol{w}||^2 + C \sum_{i=1}^{n} \xi_i^p,$$

subject to $y_i \cdot (\boldsymbol{x}_i^\top \boldsymbol{w} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$, where $p \geq 1$ and C is a constant denoting the trade-off between margin maximization and error minimization. A common method to obtain C is by using n-fold cross validation. In fact Mohri et. al. (Section 4.3) [MRT12] showed that the optimization problem is identical to separable SVMs:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\boldsymbol{x}_i^\top \boldsymbol{x}_j), \tag{2.1}$$

subject to $C \geq \alpha_i \geq 0 \wedge \sum_{i=1}^{n} \alpha_i y_i = 0$ for all $i = 1, \cdots, n$ and some constant C.

Kernel methods can be used to solve non-separable SVMs. The idea is to introduce a map $\Phi : \mathcal{X} \to \mathcal{H}$ from the input data space $\mathcal{X}$ to a higher dimensional space $\mathcal{H}$ where separation can be done easily, and kernels are introduced to solve this problem. Briefly speaking, a kernel is a function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$,

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle, \tag{2.2}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product on a Hilbert space $\mathcal{H}$. We say K is positive definite symmetric (PDS) if for any $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathcal{X}$, the gram matrix

$$\boldsymbol{K} := \Big[K(\boldsymbol{x}_i, \boldsymbol{x}_j) \Big]_{i,j} \in \mathbb{R}^{n \times n}$$

is symmetric positive semi-definite (SPSD). One property of PDS kernel is called reproducing kernel Hilbert space (RKHS), which is the idea of equation (2.2), i.e., the kernel function evaluated at points $\boldsymbol{x}, \boldsymbol{x}'$ can be viewed as the inner product between $\Phi(\boldsymbol{x})$ and $\Phi(\boldsymbol{x}')$ for our choice of the map $\Phi : \mathcal{X} \to \mathcal{H}$. One choice of kernel is the polynomial kernel (of degree d), which is defined by

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + c)^d,$$

for some constant $c > 0$. If we consider the classification problem in Figure 7 (left), we see that in $\mathbb{R}^2$ it is non-separable. However if we apply a polynomial kernel of degree 2, we have

$$K(\boldsymbol{x}, \boldsymbol{x}') = (x_1 x_1' + x_2 x_2' + c)^2 = \begin{pmatrix} x_1^2 & x_2^2 & \sqrt{2} x_1 x_2 & \sqrt{2c} x_1 & \sqrt{2c} x_2 & c \end{pmatrix} \cdot \begin{pmatrix} {x_1'}^2 \\ {x_2'}^2 \\ \sqrt{2} x_1' x_2' \\ \sqrt{2c} x_1' \\ \sqrt{2c} x_2' \\ c \end{pmatrix},$$

and we project the transformed data into a 2-dimensional subspace (see Figure 7, right), we can easily see the that the data can be classified by a hyperplane.

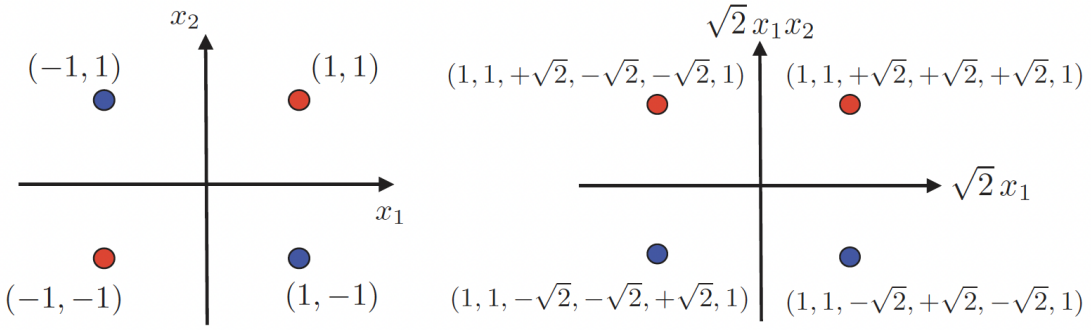

Figure 7: A binary XOR classification problem where linear separability is impossible in $\mathbb{R}^2$ (Figure on the left). After applying a polynomial kernel the transformed data can be easily classified using a hyperplane (The figure on the right). The figure is taken from Mohri et. al. [MRT12].

Other choices of kernel functions include the Gaussian kernel: $K(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-||\boldsymbol{x}-\boldsymbol{x}'||_2^2/2\sigma^2\right)$, sigmoid kernel $K(\boldsymbol{x},\boldsymbol{x}') = \tanh(a(\boldsymbol{x}^\top\boldsymbol{x}')+b)$, and many others. Once a specific kernel is chosen, we may replace the inner product term $\boldsymbol{x}_i^\top\boldsymbol{x}_j$ appeared in (2.2) by a corresponding kernel estimate, which leads to the following optimization problem:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j). \tag{2.3}$$

Denote $\boldsymbol{K}$ as the gram matrix, the optimization problem can also be written in the matrix form:

$$\max_{\boldsymbol{\alpha}} \boldsymbol{21}^\top\cdot\boldsymbol{\alpha} - (\boldsymbol{\alpha}\circ\boldsymbol{y})^\top\boldsymbol{K}(\boldsymbol{\alpha}\circ\boldsymbol{y}), \quad \text{subject to } \boldsymbol{0}\le\boldsymbol{\alpha}\le\boldsymbol{C}\wedge\boldsymbol{\alpha}^\top\boldsymbol{y}=0.$$

2.1.2 Sequential Minimal Optimization [2]

The dual optimization problem is in fact a quadratic programming (QP) problem where specific algorithms have been developed. In practice, the *sequential minimal optimization* (SMO) proposed by Platt ([Pla98]) is often used due to its fast compiling time. Briefly speaking, SMO is an iterative algorithm that breaks (2.3) down into a series of small-sized problems. We give a short overview of SMO algorithm, while a detailed implementation can be found in Platt [Pla98] as well as Mak [Mak00]. We first introduce the Karush-Kuhn-Tucker (KKT) condition commonly used for non-linear optimization problems. For each $i=1,\cdots,n$, if it satisfies

$$\begin{cases}\alpha_i = 0 & \iff y_i\cdot(\boldsymbol{x}_i^\top\boldsymbol{w}-b)\ge 1\\ 0<\alpha_i<C & \iff y_i\cdot(\boldsymbol{x}_i^\top\boldsymbol{w}-b)=1\\ \alpha_i = C & \iff y_i\cdot(\boldsymbol{x}_i^\top\boldsymbol{w}-b)\le 1\end{cases}, \tag{2.4}$$

then the quadratic programming problem defined in (2.3) has been solved. We divide the SMO algorithm into four parts: ① The set up; ② Updating multipliers; ③ Updating b; ④ Repeat until converge. A heuristic and simple derivation is given as follow.

① **The set up:** The SMO chooses to solve the smallest optimization problem at each step, and in our case the smallest possible problem is to solve the optimization problem involving

two Lagrange multipliers α_1, α_2 (without loss of generality, we shall denote α_1, α_2 as the two Lagrange multipliers in each problem). The SMO then computes the constraints on these multipliers and solves for the constrained minimum. In each case, we have

$$y_1\alpha_1 + y_2\alpha_2 = k, \quad \text{subject to } 0 \le \alpha_1, \alpha_2 \le C \tag{2.5}$$

where γ is assumed to be a constant given by $k := -\sum_{i=3}^{n} y_i\alpha_i$. Since $y_1, y_2 = \pm 1$, so this 2D optimization problem is straightforward. The figure below illustrates the two cases with y_1, y_2 having the same and opposite sign:

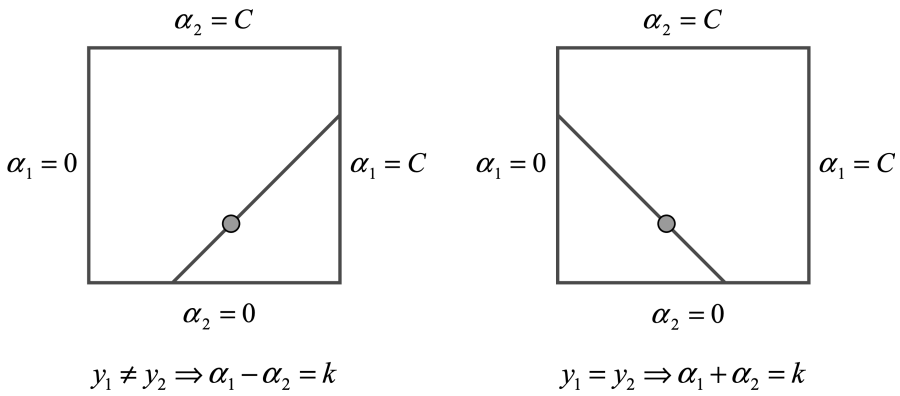

Figure 8: The 2D geometric view of the optimization problem defined in (2.5). The box represents the boundary conditions $0 \le \alpha_1, \alpha_2 \le C$, and the diagonal segment is given by the equation $y_1\alpha_1 + y_2\alpha_2 = k$ with slope ± 1 depending on $\text{sgn}(y_1y_2)$. The figure is taken from Platt [Pla98].

From Figure 8, we can see that the equation $y_1\alpha_1 + y_2\alpha_2 = k$ will intercept the constraint box, we denote the intercept on horizontal axis as L and the intercept on vertical axis as H, where we restrict $0 \le L, H \le C$. Then it is easy to see that

$$\begin{cases} L = \max\{0, \alpha_2 - \alpha_1\}, H = \min\{C, C + \alpha_2 - \alpha_1\}, & \text{if } \text{sgn}(y_1y_2) = -1 \\ L = \max\{0, \alpha_2 + \alpha_1 - C\}, H = \min\{C, \alpha_2 + \alpha_1\}, & \text{if } \text{sgn}(y_1y_2) = 1 \end{cases}.$$

② **Updating α_1, α_2:** The objective function given by (2.3) with two multipliers α_1, α_2 is given by

$$\mathcal{F}_{1,2} = \alpha_1 + \alpha_2 - \frac{1}{2}\left(\alpha_1^2 K(\boldsymbol{x}_1, \boldsymbol{x}_1) + 2\alpha_1\alpha_2 y_1 y_2 K(\boldsymbol{x}_1, \boldsymbol{x}_2) + \alpha_2^2 K(\boldsymbol{x}_2, \boldsymbol{x}_2)\right),$$

since $y_i^2 = 1, i = 1, 2$ and $K(\cdot, \cdot)$ is symmetric. If we substitute α_1 using $\alpha_1 = (k - y_2\alpha_2)/y_1$ defined through (2.5), we have

$$\begin{aligned} \mathcal{F}_{1,2}(\alpha_2) = &\left(K(\boldsymbol{x}_1, \boldsymbol{x}_2) - \frac{1}{2}K(\boldsymbol{x}_1, \boldsymbol{x}_1) - \frac{1}{2}K(\boldsymbol{x}_2, \boldsymbol{x}_2)\right)\alpha_2^2 + \left(ky_2K(\boldsymbol{x}_1, \boldsymbol{x}_1) - ky_2K(\boldsymbol{x}_1, \boldsymbol{x}_2) - \frac{y_2}{y_1} + 1\right)\alpha_2 \\ &+ \frac{k}{y_1} + \frac{1}{2}k^2K(\boldsymbol{x}_1, \boldsymbol{x}_1). \end{aligned} \tag{2.6}$$

To optimize (2.6), we may use the Newton's method to find the critical points of $\mathcal{F}_{1,2}(\alpha_2)$ and update the new multiplier α_2^{new} as

$$\alpha_2^{\text{new}} = \alpha_2 - \frac{\mathcal{F}'_{1,2}(\alpha_2)}{\mathcal{F}''_{1,2}(\alpha_2)}.$$

The first derivative of (2.6) is given by

$$\frac{\partial \mathcal{F}_{1,2}(\alpha_2)}{\partial \alpha_2} = 2\alpha_2 \cdot \left(K(\boldsymbol{x}_1, \boldsymbol{x}_2) - \frac{1}{2}K(\boldsymbol{x}_1, \boldsymbol{x}_1) - \frac{1}{2}K(\boldsymbol{x}_2, \boldsymbol{x}_2) \right) + ky_2 K(\boldsymbol{x}_1, \boldsymbol{x}_1) - ky_2 K(\boldsymbol{x}_1, \boldsymbol{x}_2) - \frac{y_2}{y_1} + 1.$$

We define the predicted value u_i and the error e_i as

$$u_i = f(\boldsymbol{x}_i) = \sum_j \alpha_j y_j K(\boldsymbol{x}_j, \boldsymbol{x}_i) + b; \quad e_i = u_i - y_i,$$

then it can be shown that $\mathcal{F}'_{1,2} = y_2(e_1 - e_2)$. Meanwhile the second derivative of (2.6) is given by

$$\frac{\partial^2 \mathcal{F}_{1,2}(\alpha_2)}{\partial \alpha_2^2} = -\left(K(\boldsymbol{x}_1, \boldsymbol{x}_1) + K(\boldsymbol{x}_2, \boldsymbol{x}_2) - 2K(\boldsymbol{x}_1, \boldsymbol{x}_2)\right) := -\eta.$$

Hence the updated multiplier α_2^{new} is given by

$$\alpha_2^{\text{new}} = \alpha_2 + \frac{y_2(e_1 - e_2)}{\eta}. \tag{2.7}$$

To insure the updated multiplier lies within the desired constraints, we define the clipped α_2^{new} as

$$\alpha_2^{\text{new,clipped}} = \begin{cases} H & \text{if } \alpha_2^{\text{new}} > H \\ \alpha_2^{\text{new}} & \text{if } L < \alpha_2^{\text{new}} < H \\ L & \text{if } \alpha_2^{\text{new}} \leq L \end{cases}.$$

Note that as we update α_2, the other multiplier α_1 will change as well. The updated α_1 can be directly computed by

$$\alpha_1^{\text{new}} = \alpha_1 + \text{sgn}(y_1 \cdot y_2) \cdot (\alpha_2 - \alpha_2^{\text{new,clipped}}),$$

using the constraint that

$$y_1\alpha_1 + y_2\alpha_2 = k \quad \text{and} \quad y_1\alpha_1^{\text{new}} + y_2\alpha_2^{\text{new,clipped}} = k.$$

③ Updating b: The decision function is given by

$$f(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b,$$

hence once we have obtained $\alpha_2^{\text{new,clipped}}; \alpha_1^{\text{new}}$, the parameter b needs to be updated accordingly in order to fulfill the KKT conditions for both multipliers. If $\boldsymbol{x}_i$ is a support vector, then we have $y_i f(\boldsymbol{x}_i) = 1$ and furthermore

$$b = y_i - \sum_{j=1}^{n} \alpha_j y_j K(\boldsymbol{x}_j, \boldsymbol{x}_i).$$

We now replace α_1, α_2 by their updates. and hence we have

$$f^{\text{new}}(\boldsymbol{x}_1) = f(\boldsymbol{x}_1) + (\alpha_1^{\text{new}} - \alpha_1)y_1 K(\boldsymbol{x}_1, \boldsymbol{x}_1) + (\alpha_2^{\text{new,clipped}} - \alpha_2)y_2 K(\boldsymbol{x}_2, \boldsymbol{x}_1) + (b^{\text{new}} - b).$$

Define $e_1 = f(\boldsymbol{x}_1) - y_1$, the updated b_1 is given by

$$b_1 = b - e_1 - y_1(\alpha_1^{\text{new}} - \alpha_1)K(\boldsymbol{x}_1, \boldsymbol{x}_1) - y_2(\alpha_2^{\text{new,clipped}} - \alpha_2)K(\boldsymbol{x}_2, \boldsymbol{x}_1),$$

and the same idea applies for b_2:

$$b_2 = b - e_2 - y_1(\alpha_1^{\text{new}} - \alpha_1)K(\boldsymbol{x}_1, \boldsymbol{x}_1) - y_2(\alpha_2^{\text{new,clipped}} - \alpha_2)K(\boldsymbol{x}_2, \boldsymbol{x}_1).$$

The updated b^{new} is then given by

$$b^{\text{new}} = \begin{cases} b_1 & \text{if } 0 < \alpha_1^{\text{new}} < C \\ b_2 & \text{if } 0 < \alpha_2^{\text{new,clipped}} < C \\ \frac{b_1+b_2}{2} & \text{otherwise} \end{cases}.$$

④ **Repeat until convergence:** The final step of the algorithm is to check is there still any α_i that violates the KKT conditions. If yes, then we will perform another SMO step. If not, then algorithm is then terminated, and output the desired $\boldsymbol{\alpha}$ and b. Compared to (2.4), often times we say the SMO converges when all KKT conditions are satisfied with small tolerance. That is, for a fixed ϵ, we check for all $\boldsymbol{x}_i$, if

$$\begin{cases} \alpha_i = 0 & \implies y_i f(\boldsymbol{x}_i) \geq 1 - \epsilon \\ 0 < \alpha_i < C & \implies |y_i f(\boldsymbol{x}_i) - 1| \leq \epsilon \\ \alpha_i = C & \implies y_i f(\boldsymbol{x}_i) \leq 1 + \epsilon \end{cases}.$$

To summarize ①,②,③,④, we present the persudo code of the SMO algorithm as follow:

Algorithm 2 Sequential Minimal Optimization (SMO)

Require: Training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $y_i \in \{-1, +1\}$, kernel K, penalty C
Ensure: Optimal dual variables $\boldsymbol{\alpha}$, b

```
Initialize α_i ← 0 for all i, b ← 0
while KKT conditions are violated do
    Select indices i ≠ j
    Optimize (α_i, α_j) subject to 0 ≤ α_i, α_j ≤ C, α_i y_i + α_j y_j = k using algorithm described in ②, ③.
    Update threshold α_i, α_j, b.
end while
return (α, b)
```

We now return to the `wine` data set in **R**. We implement SVMs using a Gaussian kernel to find the optimal decision boundaries among three classes. We choose 4 groups of binary covariates: (V1) and (V2); (V3) and (V6); (V4) and (V8); (V9) and (V10) (see Figure 1 for more detail about the covariates), all of which are tricky to classify using LDA or QDA. The data points are plotted in Figure 9, where different colors refer to different classes (cultivars). The SVM decision regions are shown in three different colors representing three cultivars.

To test the accuracy of the classifier f, we define the training error as

$$\mathbb{P}(f(\boldsymbol{x}_i) \neq y_i) := \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{f(\boldsymbol{x}_i) \neq y_i\}. \tag{2.8}$$

We may also use (2.8) to compute the training error on `wine` data set using different classification methods we discussed so far (LDA, QDA, SVM) and their performance is shown in Table 1. We can easily see that SVMs have the least training error compared to LDA and QDA. The choice of the kernel is also important, where Table 2 compares the training error using SVMs but with different kernels (linear SVM, polynomial kernal with degree 2, 3, sigmoid kernel, and Gaussian kernel).

	LDA	QDA	SVM with Gaussian kernel
V1 and V2	0.1910	0.1853	0.1685
V3 and V6	0.2978	0.2752	0.2528
V4 and V8	0.4157	0.3876	0.3483
V9 and V10	0.1685	0.1404	0.1236

Table 1: The table illustrates the training error for each classification method (LDA, QDA, SVMs with Gaussian Kernel) and each binary covariates.

	Linear	Poly_d2	Poly_d3	Gaussian	Sigmoid
V1 and V2	0.2134831	0.3876404	0.2078652	0.1685393	0.2303371
V3 and V6	0.2977528	0.5393258	0.4157303	0.2528090	0.3651685
V4 and V8	0.4044944	0.5561798	0.4887640	0.3483146	0.4719101
V9 and V10	0.1348315	0.3539326	0.1516854	0.1235955	0.2808989

Table 2: The training error of the SVM using different kernels.

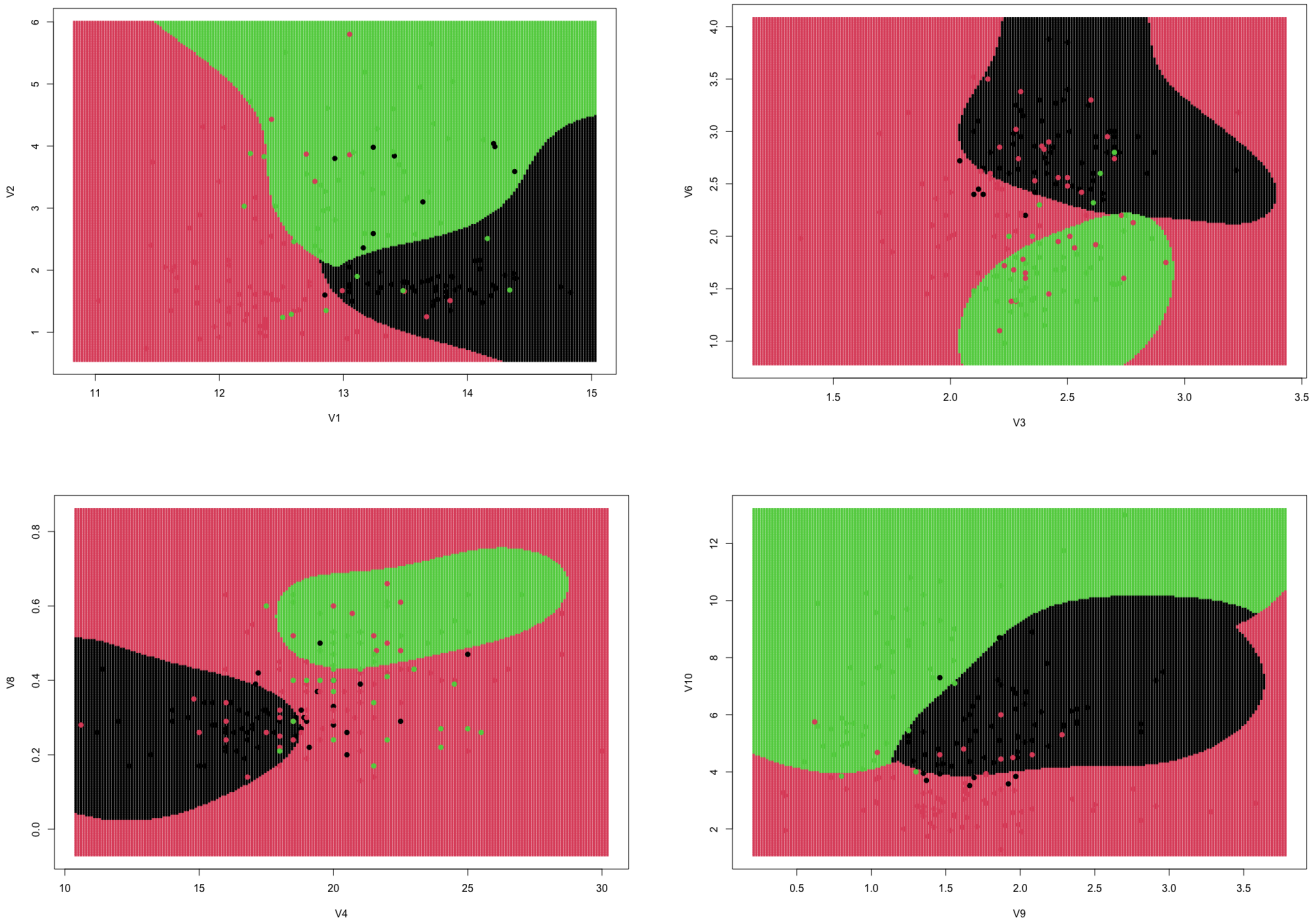

Figure 9: The decision regions using SVM with Gaussian kernel given by $K(\boldsymbol{x}, \boldsymbol{x}') = \exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2)$.

2.2 Logistic Regression

Binary logistic regression is one of the easiest classification model to consider. We assume the response y_i is a Bernoulli random variable with $\mathbb{P}(Y_i = 1|\boldsymbol{x}_i) = p_i, \mathbb{P}(Y_i = 0|\boldsymbol{x}_i) = 1 - p_i$. We further assume p_i is a function of the covariates $\boldsymbol{x}_i \in \mathbb{R}^p$ with coefficients vector $\boldsymbol{\beta}$, then assuming independence, the likelihood function of the sample $(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)$ can be written as

$$L(y_1, \cdots, y_n|\boldsymbol{\beta}) = \prod_{i=1}^{n} p_i(\boldsymbol{\beta})^{y_i} \cdot [1 - p_i(\boldsymbol{\beta})]^{1-y_i}.$$

The first choice for p_i was the cumulative distribution function of a standard normal random variable. For each $p_i \in [0, 1]$, we can find a value $\widehat{z}$ such that

$$p_i = \Phi(\widehat{z}) := \int_{-\infty}^{\widehat{z}} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right) du,$$

and the function $\Phi^{-1}(p_i)$ is called the *probit function*. However the probit model can be computationally challenging since it involves numerical methods to evaluate an integral, and the interpretation of parameters is not obvious [Ste26]. An alternative approach by Berkson is to use the following model:

$$p_i = \frac{\exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})}{1 + \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})}. \tag{2.9}$$

The function $f(x) = e^x/(1 + e^x)$ is also known as the sigmoid function. Its graph is plotted in Figure 10

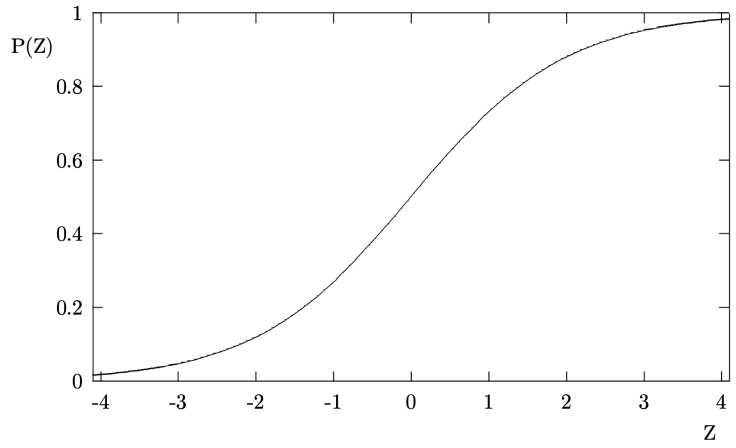

Figure 10: The graph of the sigmoid function $f(x) = e^x/(1 + e^x)$. The figure is taken from Cramer [Cra02]

The sigmoid function was invented in the 19th century, and was originally used to describe the growth of certain population under resource-limited environment, as well as the course of chemical reactions. It originates from the following differential equation:

$$\frac{dW(t)}{dt} = \beta W(t) \cdot (\Omega - W(t)), \tag{2.10}$$

where β, Ω are constants. It is easy to show that the solution to (2.10) is given by

$$W(t) = \frac{\Omega e^{\beta t + C}}{1 + e^{\beta t + C}},$$

for some constant C that depends on the initial condition. A detailed history about logistic regression can be found in Cramer [Cra02]. Under the assumption given in (2.9), the log-likelihood function of the sample is given by

$$\ell(y_1, \cdots, y_n | \boldsymbol{\beta}) = \sum_{i=1}^{n} y_i \cdot \frac{\exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})}{1 + \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})} + (1 - y_i) \cdot \left[1 - \frac{\exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})}{1 + \exp(\boldsymbol{x}_i^\top \boldsymbol{\beta})} \right],$$

and the optimal solution $\widehat{\boldsymbol{\beta}} = (\widehat{\beta}_0, \widehat{\beta}_1, \cdots, \widehat{\beta}_{p-1})^\top$ can be obtained by solving

$$\frac{\partial \ell(y_1, \cdots, y_n | \boldsymbol{\beta})}{\partial \beta_i} = 0.$$

Numerical methods are commonly used to obtain the zeros of the first derivative of $\ell(y_1, \cdots, y_n | \boldsymbol{\beta})$, for example: Newton's method and gradient descent. After obtaining our estimate $\widehat{\boldsymbol{\beta}}$, the predicted logistic function is then

$$\widehat{p}_i = \frac{\exp(\boldsymbol{x}_i^\top \widehat{\boldsymbol{\beta}})}{1 + \exp(\boldsymbol{x}_i^\top \widehat{\boldsymbol{\beta}})} \in [0, 1]. \tag{2.11}$$

A simulation is shown in Figure 11.

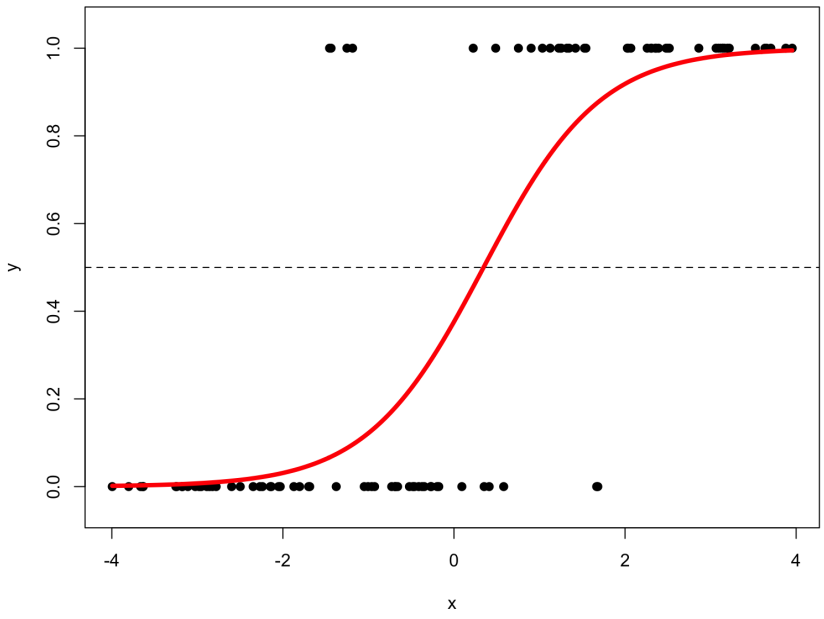

Figure 11: A simulation of a single covariate logistic regression. The response $y_i = \{0, 1\}$ is binary and the model assumes $p_i = \exp(\beta_0 + \beta_1 x)/(1 + \exp(\beta_0 + \beta_1 x)$. The regression function is shown in red and the decision boundaries is the given by $p = 0.5$.

3 Tree-based Classification Algorithms

3.1 Decision Trees: An Introduction

Unlike many other classification algorithms that produce black-box models, decision trees offer transparency by representing the decision processes as a sequence of simple and intuitive rules (Mienye and Jere [MJ24]). The basic structure of a decision tree consists a bunch of nodes where decisions are made based on the attributes (or covariates) of the data. Each decision node has several children indicating different outcomes based on the attribute, and all the leaves nodes represent a set of classified data according to a sequence of decision-making processes. The idea is to recursively split the data into subsets based on attributes until some stopping criterion is met. The splitting algorithm first identifies the feature and threshold that leads to the best split based on specific criterion, then each children of the node represents a group of classified data according to the features determined by the root node. Each children will then be the root of a new tree and this process will continue recursively until some stopping criterion is met, typically when all the data inside each node is homogeneous, or the tree reaches a certain depth.

One crucial rule for constructing a decision tree is to define the splitting rules, i.e., how one should divide the data into subsets based on the attributes, since different splitting rules may lead to different tree structures. A deep decision tree has higher accuracy but is computationally expensive; a shallow decision tree is more computationally efficient but may be less accurate and may not fully capture the correlations and decision rules among attributes.

The following table consists of a dataset taken from Quinlan [Qui85], where we want to know whether a day is suitable for playing tennis based on the whether conditions. The attributes are 'Outlook' (sunny, overcast, rain), 'Temperature' (hot, mild, cool), 'Humidity' (high, normal), 'Wind' (strong weak), and the response of interest is simply 'Yes' or 'No'.

Date	Outlook	Temp	Humidity	Wind	Play Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Mild	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Table 3: The whether condition collected for 14 different days.

The figure on the next page illustrates two different decision trees of the data in Table 3 based on two different splitting rules and stopping criterion.

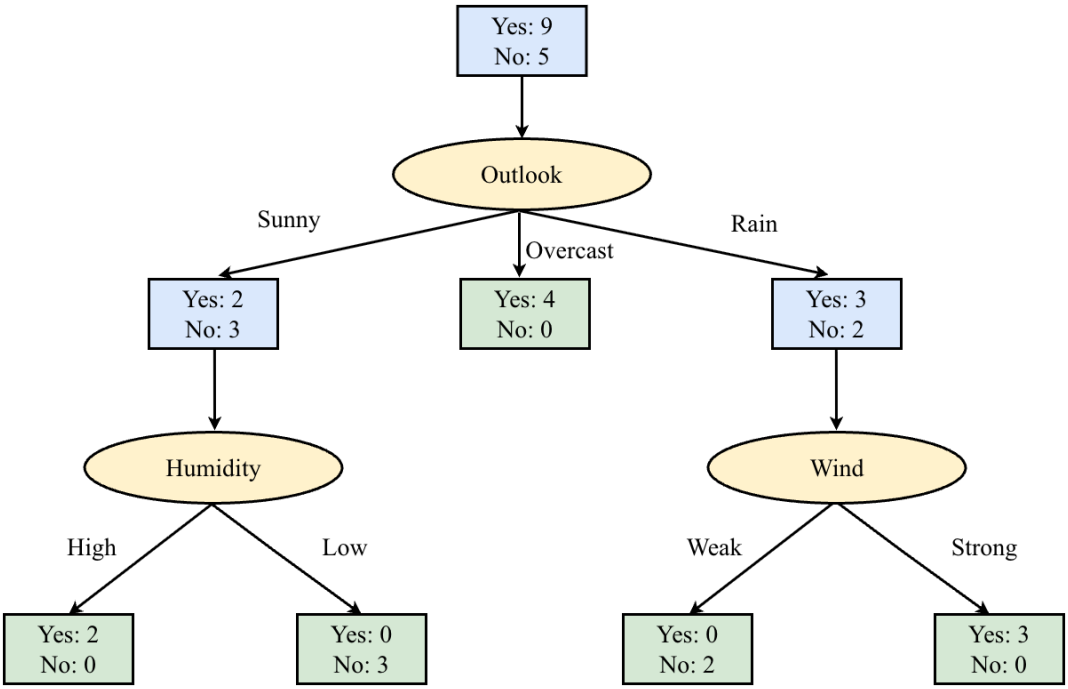

Figure 12: A decision tree constructed based on the whether data provided by Quilan. Each yellow node represents a decision node where current data are split according to the attributes, and each green node represents a leaf node where the data are homogeneous according to some decision rules.

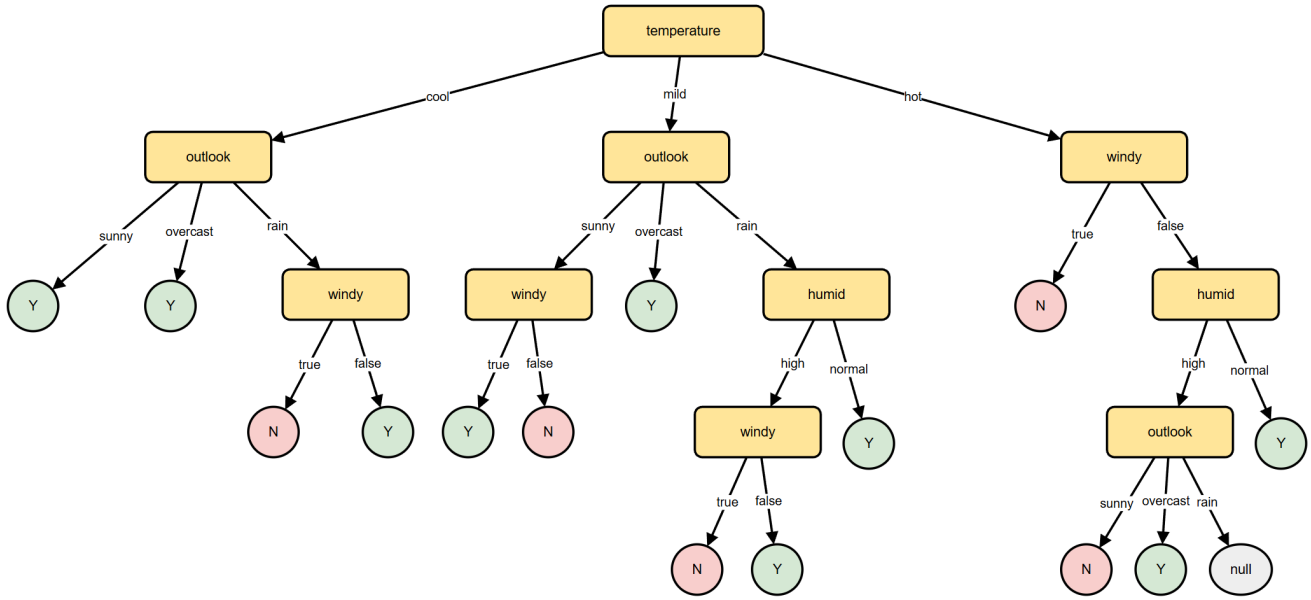

Figure 13: A more complicated decision tree structure for data given by Table 3. Each leaf corresponds to a unique date, and each internal node represents a decision making process based on some attributes.

From Figure 12 and 13, we can see that one advantage of decision tree is that the decision criteria is clear and easy to follow from the tree structure.

Several splitting rules had been proposed, and each rule led to a splitting algorithm. Namely the Information Gain (GI) used in Iterative Dichotomiser 3 (ID3) algorithm (Quinlan [Qui85]); Gini Index used in Classification and Regression Trees (CART) algorithm (Breiman et. al. [BFOS84]); the Chi-Square (χ^2) splitting criterion used in Chi-Squared Automatic Interaction Detection (CHAID) algorithm (Kass [Kas80]), etc.

3.2 Decision Tree Algorithms

3.2.1 Information Gain (IG) and ID3 Algorithm [3]:

The idea comes from the notion of entropy used in information theory, which measures the unpredictability or randomness in a set of data. The entropy E of a set S with attribute A is defined by

$$E(S_A) = -\sum_{x \in S} p_x \log_2 p_x,$$

where x represents different target classes in S, and p_x is the proportion of the samples in S that belong to class x. For example, the entropy of Table 3 is given by

$$E(S_A) = -\left(\frac{9}{14} \times \log_2 \frac{9}{14} + \frac{5}{14} \times \log_2 \frac{5}{14}\right) = 0.9403,$$

since there are 9 'yes's and 5 'no's in the target class. The information gain (GI) for a split on S with attribute A is given by

$$IG(S, A) = E(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} E(S_v), \tag{3.1}$$

where Values(A) are different values that attribute A can take, $S_v \subset S$ is the subset of data that has attribute v. Equation (3.1) computes the change in entropy from the original set S to S_v after the split, and a higher IG indicates a more effective attribute for splitting, since it results in more homogeneous subsets ([MJ24]). In Table 3, if we compute its IG based on attributes $A =$ outlook, $B =$ humidity, $C =$ temperature, $D =$ wind, we see that

$$\begin{aligned} IG(S, A) &= 0.9403 - \left(\frac{5}{14} \cdot E(S_{A1}) + \frac{4}{14} \cdot E(S_{A2}) + \frac{5}{14} \cdot E(S_{A3})\right) \\ &= 0.9403 - \left(\frac{5}{14} \cdot 0.971 + \frac{4}{14} \cdot 0 + \frac{5}{14} \cdot 0.971\right) \\ &= 0.2467; \end{aligned} \tag{3.2}$$

$$\begin{aligned} IG(S, B) &= 0.9403 - \left(\frac{7}{14} \cdot E(S_{B1}) + \frac{7}{14} \cdot E(S_{B2})\right) \\ &= 0.9403 - \left(\frac{7}{14} \cdot 0.9852 + \frac{7}{14} \cdot 0.5917\right) \\ &= 0.1518; \end{aligned} \tag{3.3}$$

$$
\begin{aligned}
IG(S,C) &= 0.9403 - \left(\frac{3}{14}\cdot E(S_{C1}) + \frac{7}{14}\cdot E(S_{C2}) + \frac{4}{14}\cdot E(S_{C3})\right) \\
&= 0.9403 - \left(\frac{3}{14}\cdot 0.9183 + \frac{7}{14}\cdot 0.8631 + \frac{4}{14}\cdot 0.8112\right) \\
&= 0.0802; \qquad (3.4)
\end{aligned}
$$

$$
\begin{aligned}
IG(S,D) &= 0.9403 - \left(\frac{8}{14}\cdot E(S_{D1}) + \frac{6}{14}\cdot E(S_{D2})\right) \\
&= 0.9403 - \left(\frac{8}{14}\cdot 0.8113 + \frac{6}{14}\cdot 1\right) \\
&= 0.0481. \qquad (3.5)
\end{aligned}
$$

Hence based on our computation ((3.2); (3.3); (3.4); (3.5)), $IG(S,A)$ (split based on outlook) maximizes the information gain, so Table 3 in fact shows the best split based on this criteria. The decision tree algorithm based on information gain is called Iterative Dichotomiser 3 (ID3) algorithm proposed by Quinlan ([Qui85]).

Algorithm 3 ID3 Decision Tree Algorithm

Require: Training dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
Ensure: Decision tree T

function ID3($\mathcal{D}$)
  if $\mathcal{D} = \varnothing$ **then**
    return a terminal node with default class c_{default}
  end if
  if all instances in $\mathcal{D}$ have the same class label y **then**
    return a terminal node with class y
  end if
  if attribute set $\mathcal{J}$ is empty **then**
    return a terminal node with the prevalent class in $\mathcal{D}$
  end if
  Select the feature f that best splits the data using information gain
  Create a decision node for f
  for each value b_i of feature f **do**
    Create a branch for b_i
    Let $\mathcal{D}_i = \{(x, y) \in \mathcal{D} : x_f = b_i\}$
    Recursively build the subtree $T_i \leftarrow$ ID3($\mathcal{D}_i$)
    Attach T_i to the branch for b_i
  end for
    return the decision node
end function

To summarize, the ID3 decision tree algorithm is a top-to-bottom, greedy search algorithm for constructing a decision tree. It begins with the entire dataset and iteratively divide the dataset based on the attribute that maximizes the information gain. The attribute that maximizes the information gain is then chosen as a decision node and partition the remaining

dataset, and the process terminates when some stopping criteria is met. One disadvantage for ID3 decision tree algorithm is that it only works with discrete attributes, lacking the treatment for continuous cases. Also it is unable to prevent overfitting.

3.2.2 Gini Index and CART Algorithm [4]:

The Gini index measures the probability of a randomly chosen sample being incorrectly classified if it was randomly labeled. It can also be used to evaluate the split in the tree, and the Gini index for a set S is given by

$$\text{Gini}(S) = 1 - \sum_{i=1}^{n} p_i^2, \tag{3.6}$$

where n is the number of unique classes in the set, and p_i is the proportion of the samples in the set that belong to class i. The formula in (3.6) is equal to the probability that a randomly chosen sample $\boldsymbol{x} \in S$ with random class assignment $\mathbb{P}(y = i) = p_i$ is misclassified. It is not hard to see that $\text{Gini}(S) \in [0, 1]$, and a smaller Gini index represents a more homogeneous set. By splitting the set S into two subsets S_1 and S_2, we may compute the Gini index of the split by

$$\text{Gini}(S_1, S_2) = \frac{|S_1|}{|S|} \cdot \text{Gini}(S_1) + \frac{|S_2|}{|S|} \cdot \text{Gini}(S_2), \tag{3.7}$$

where a smaller $\text{Gini}(S_1, S_2)$ would result in a better split. The Classification and Regression Tree (CART) algorithm proposed by Breiman et. al. [BFOS84] finds the best split based on minimizing the split Gini index defined in (3.7), where the idea is to create a binary decision tree such that at each decision node a question regarding the attributes is being asked (e.g., is the wind condition weak?). The answer is then simply 'Yes' or 'No', resulting in a partition of the data. The algorithm then applies iteratively to each set in the partition and the binary decision tree structure follows. The Table 4 below shows the Gini index for different splits on the root node of the tennis data.

Attribute	Question	$(Y, N)_{S_1}$	$\text{Gini}(S_1)$	$(Y, N)_{S_2}$	$\text{Gini}(S_2)$	$\text{Gini}(S_1, S_2)$	$\Delta\text{Gini}(S)$
Outlook	Overcast?	(4,0)	0.0000	(5,5)	0.5000	**0.3571**	**0.1020**
Outlook	Rain?	(3,2)	0.4800	(6,3)	0.4444	0.4571	0.0020
Outlook	Sunny?	(2,3)	0.4800	(7,2)	0.3457	0.3937	0.0655
Temp	Cool?	(3,1)	0.3750	(6,4)	0.4800	0.4500	0.0092
Temp	Hot?	(1,2)	0.4444	(8,3)	0.3967	0.4069	0.0523
Temp	Mild?	(5,2)	0.4082	(4,3)	0.4898	0.4490	0.0102
Humidity	High?	(3,4)	0.4898	(6,1)	0.2449	0.3673	0.0918
Humidity	Normal?	(6,1)	0.2449	(3,4)	0.4898	0.3673	0.0918
Wind	Strong?	(3,3)	0.5000	(6,2)	0.3750	0.4286	0.0306
Wind	Weak?	(6,2)	0.3750	(3,3)	0.5000	0.4286	0.0306

Table 4: The computed Gini index for each potential split on root the node of the data given in 3. The Gini impurity reduction $\Delta\text{Gini}(S)$ on the last column is defined as $\Delta\text{Gini}(S) = \text{Gini}(S) - \text{Gini}(S_1, S_2)$. The attribute 'Outlook' with question 'Overcast' has the smallest Gini split index.

Suppose at one step we have the data $S := \{(\boldsymbol{x}_1, y_1), \cdots, (x_n, y_n)\}$, a set of unique attributes $f_1, \cdots, f_m$, and furthermore each attribute has several categories: $f_i := \{f_{i1}, \cdots, f_{ik(i)}\}$. To find the best split, we first find all binary partitions of S that are induced by a single attribute taking one category and its complement:

$$S_{1ij} := \left\{\boldsymbol{x} \in S : \boldsymbol{x}(f_i) = j\right\}, \quad S_{2ij} = S \setminus S_{1ij}, \tag{3.8}$$

for all $i \in [m], j \in [k(i)]$. Then we choose the best partition (S_1', S_2') with the lowest value of split Gini index:

$$\text{Gini}(S_1', S_2') := \min_{i \in [m], j \in [k(i)]} \text{Gini}(S_{1ij}, S_{2ij}). \tag{3.9}$$

Algorithm 4 CART Algorithm

Require: Training dataset $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$
Ensure: Decision tree T

function CART($\mathcal{S}$)
  if $\mathcal{S} = \varnothing$ **then**
    return a terminal node with default class c_{default}
  end if
  if all instances in $\mathcal{S}$ have the same class label y **then**
    return a terminal node with class y
  end if
  if attribute set $\mathcal{J}$ is empty **then**
    return a terminal node with the prevalent class in $\mathcal{S}$
  end if
  Partition the data set $\mathcal{S}$ into $\mathcal{S}_1, \mathcal{S}_2$ based on the criterion defined in (3.8) and (3.9) (If the attribute is continuous then use (3.10), (3.11)).
  Recursively build the subtree $T_1 \leftarrow \text{CART}(\mathcal{S}_1)$ and $T_2 \leftarrow \text{CART}(\mathcal{S}_2)$.

  Attach T_1, T_2 to the decision node.
    return the decision node
end function

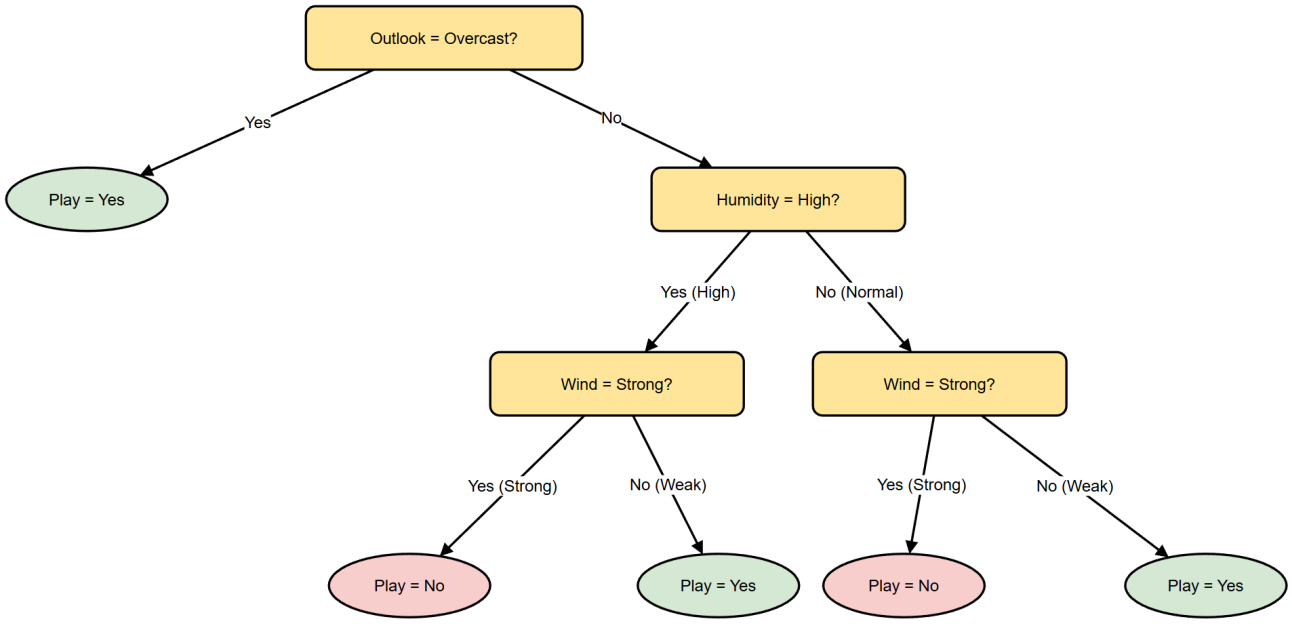

Figure 14: The decision tree of the data from 3 constructed using CART algorithm

Unlike ID3 algorithm, one advantage of CART algorithm is that it can also deal with continuous attributes. Since each node is simply a 'Yes' or 'No' question, we may partition the data into 2 sets based on the value of the attribute being greater or smaller than a chosen threshold. In this case, instead of the Gini index, we will use the sample variance as the measure of impurity. For a set S with continuous attribute Y, we define

$$\mathbb{V}\mathbf{ar}(S) = \frac{1}{|S|}\sum_{i=1}^{m}(Y_i - \bar{Y})^2, \tag{3.10}$$

where $\bar{Y}$ is the sample mean. The variance of the split $S = S_1 \cup S_2$ is then a weighted sum of the variance on each set:

$$\mathbb{V}\mathbf{ar}(S_1, S_2) = \frac{|S_1|}{|S|} \cdot \mathbb{V}\mathbf{ar}(S_1) + \frac{|S_2|}{|S|} \cdot \mathbb{V}\mathbf{ar}(S_2). \tag{3.11}$$

Then the CART algorithm will choose the split that has the minimum variance. The Figure 15 below shows the CART decision tree for the housing price based on longitude and latitude in California [Sha15]. The value in each grid represents the logarithm of the median housing price in that region.

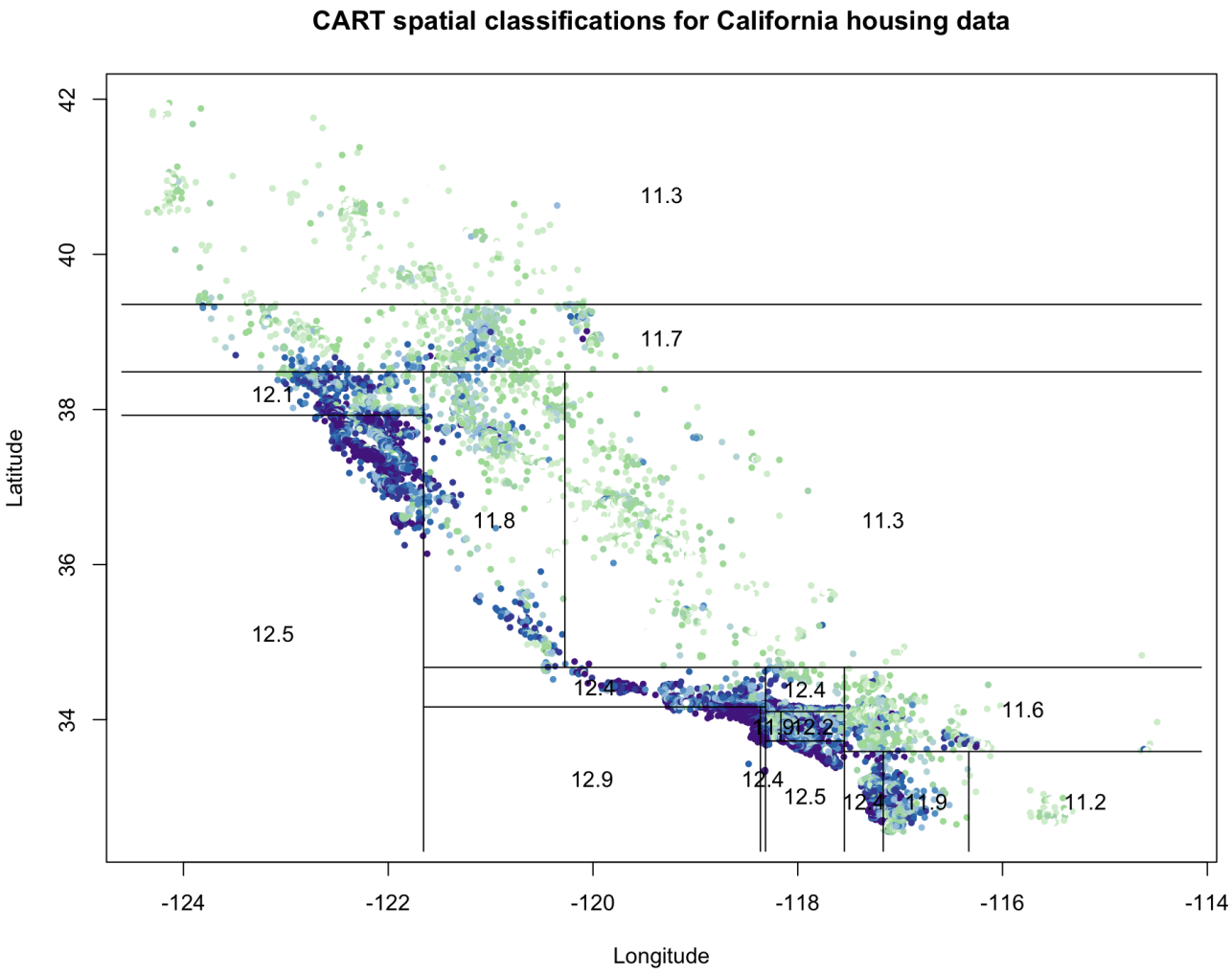

Figure 15: The CART decision tree for California housing price data.

3.3 Averaging Classifiers: Bagging

The decision trees suffer from the issue of high variance: If we randomly split the data into two parts and fit a decision tree on each half, then the results could be very different ([JWHT21], Chapter 8.2). If we have n i.i.d random variables X_i with variance σ^2, then we know the sample mean has variance given by $\mathbb{V}\mathbf{ar}(\bar{X}) = \sigma^2/n$. Which means, we may average a set of observations to reduce the variance.

Since we only have one data set $\mathcal{D}_n = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)\}$, the idea is to use *bootstrap sampling* to obtain K i.i.d samples. In each bootstrap sample of size n, objects are chosen randomly with replacement from the dataset. The Figure 16 below provides a graphical illustration of bootstrap sampling.

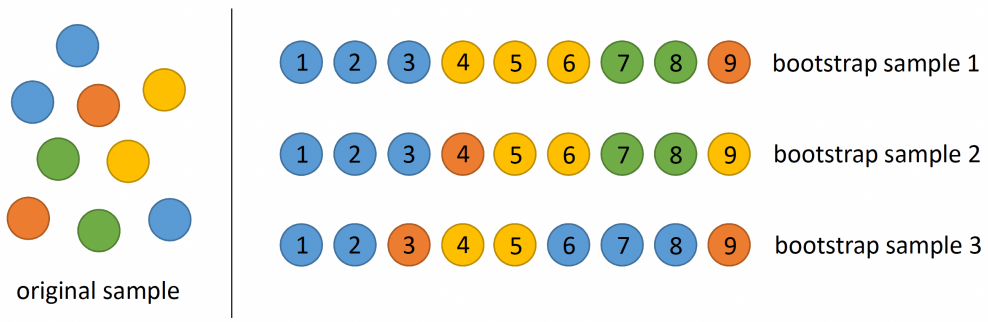

Figure 16: An illustration of bootstrap sampling. The original sample has 4 different classes (color), and the right is an example of 3 bootstrap samples. Since object replacement is allowed hence there may be multiple objects belonging to the same class. This means some classes may be over-represented (blue class in sample 3), also some classes may be under-represented (green class in sample 3). The Figure is taken from Galdi and Tagliaferri ([GT17]).

After we have obtained K i.i.d samples, we would fit a decision tree T_i on each sample i, and average the resulting predictions.

$$T_{\text{avg}} = \frac{1}{K}\sum_{k=1}^{K} T_k.$$

In our classification problem, the outcome of $T_i(\boldsymbol{x})$ for a data $\boldsymbol{x}$ is a categorical class label $y \in \{1, \cdots, m\}$. So the label of $\boldsymbol{x}$ is determined by the majority vote of class in T_k, i.e., the outcome class y is the most common class among the K predictions.

$$T_{\text{avg}}(\boldsymbol{x}) = j, \quad \text{where } j = \max\left(\mathbf{1}\left\{\sum_{i=1}^{n} T_i(\boldsymbol{x}) = j\right\}\right), \forall i \in [K].$$

In the bagging processes, the trees are grown deep without pruning, meaning each tree have large variance and low bias. Averaging those K decision trees will decrease the variance. The test error of the bagging model can be easily estimated. In each bootstrap sample B, the probability that a specific data $\boldsymbol{x}_i$ is not being selected into this sample is given by

$$\mathbb{P}(\boldsymbol{x}_i \notin B) = \left(\frac{n-1}{n}\right)^n \approx e^{-1},$$

for large sample size n. It means that each Bootstrap sample uses roughly $2/3$ of the data in $\mathcal{D}_n$. The remaining $\sim 1/3$ not used in this Bootstrap sample are referred as the *out-of-bag*

(OOB) observations. Which means we can predict the response of $\boldsymbol{x}_i$ based on those Bootstrap samples where $\boldsymbol{x}_i$ is an OOB observation. This will yield $\sim K/3$ predictions for $\boldsymbol{x}_i$, where K is the Bootstrap sample size. To obtain one single prediction for $\boldsymbol{x}_i$, we may take the majority vote among those predictions.

One disadvantage of bagging is that it fails to interpret the resulting model through a clean visualization like decision trees. When we bag a large number of decision trees, it is hard to represent the resulting hierarchy structure by a single tree, and it is no longer clear which attribute plays the most important decision roles. In other words, bagging improves the prediction accuracy at the expense of losing interpretability. [JWHT21].

Despite bagging is much more difficult to interpret than a classification tree, one can still obtain an overall summary of the importance of each attribute. For classification tree we compute the overall decrease of Gini index for a given attribute averaging over all K trees. The Figure 17 below illustrates the attribute importance plot from the `Heart` data set. The variance importance is computed using the mean of decrease in Gini index over all trees. The importance level of the important attribute (`Thal`) is set to 100, and the importance level of other attributes are expressed relative to `Thal`.

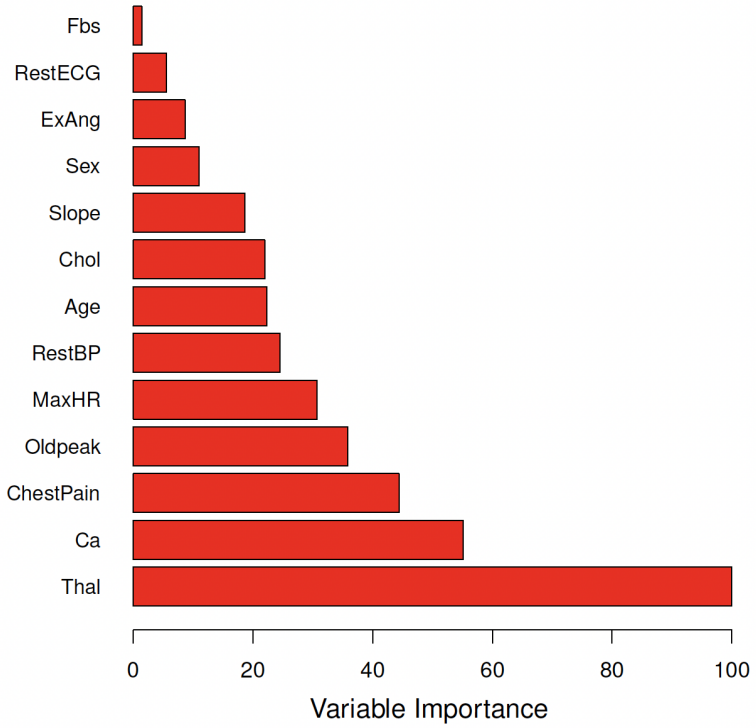

Figure 17: The attribute importance plot from the `Heart` data. The figure is taken from [JWHT21].

3.4 Random Forests

3.4.1 Introduction

The basic idea of random forest is to generate a set of decision trees using different subsets of the input samples and then combine their outputs to obtain a final decision ([MJ24], also

see Figure 18). The algorithm uses two main techniques called Bootstrap Sampling and Feature Randomization. Bootstrap sampling is briefly discussed in Section 3.3, and what differs random forests from bagging is the feature Randomization. Briefly speaking, feature randomization is done by selecting random features for each decision tree, which reduces the probability of selecting the same feature for different trees. Those two techniques would improve the diversity and accuracy of the algorithm and also reduce the variance and prevent overfitting.

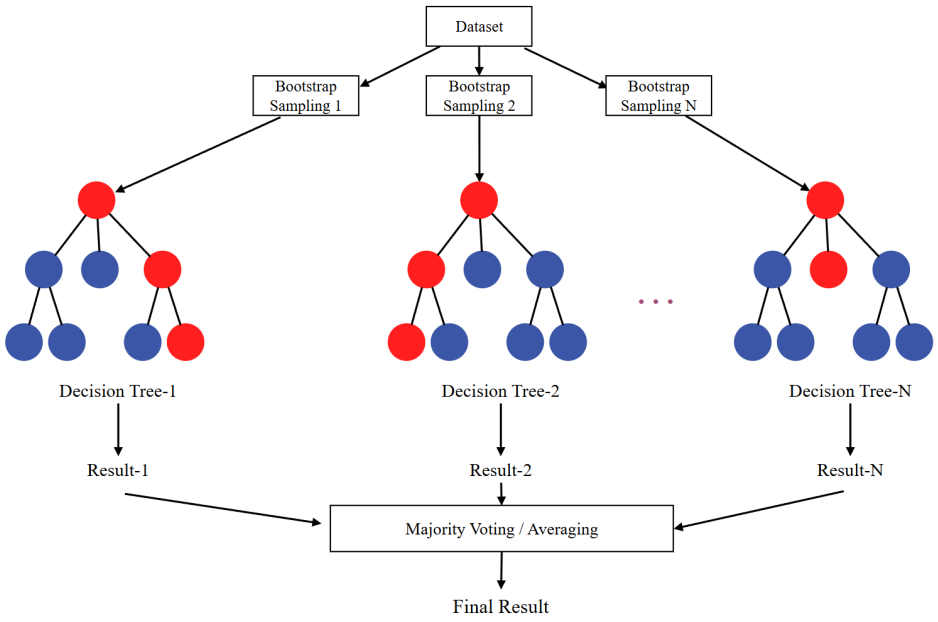

Figure 18: The structure of a random forest

3.4.2 Convergence of Random Forests

The convergence of random forest algorithm was proven by Breiman [Bre01], followed by the consistency proof by Biau et. al. [BDL08] and Scornet et. al. [SBV15]. Suppose the data are jointly distributed random variables $(\boldsymbol{X}, Y)$, where $Y \in \{1, \cdots, m\}$. Denote Θ as a random vector representing the "randomness" (i.e., bootstrap sampling, feature randomization, etc.) in the procedure of generating random forests. $\Theta_1, \cdots, \Theta_K$ are a sequence of i.i.d random vectors and a sequence of decision trees $T(\boldsymbol{X}, \Theta_i) = T_i(\boldsymbol{X})$ are grown using those random vectors. The margin function is defined as

$$mg_K(\boldsymbol{X}, Y) = \frac{1}{K}\sum_{i=1}^{K} \mathbf{1}\{T_i(\boldsymbol{X}) = Y\} - \max_{j \neq y} \frac{1}{K}\sum_{i=1}^{K} \mathbf{1}\{T_i(\boldsymbol{X}) = j\}.$$

The margin function hence measures the extent to which the average number of votes at $\boldsymbol{X}$, Y for the right class exceeds the average vote for any other class [Bre01]. It is easy to see that a larger margin yields more confidence in the classification. When $mg_K(\boldsymbol{X}, Y) < 0$, it means the maximum average vote on the wrong class exceeds the right class, and the generalization error is given by

$$PE_K^* = \mathbb{P}_{\boldsymbol{X},Y}(mg_K(\boldsymbol{X}, Y) < 0),$$

and Breiman showed that, as $k \to \infty$, by the strong law of large numbers, we have

$$PE_K^* \to \mathbb{P}_{\boldsymbol{X},Y}\Big(\mathbb{P}_\Theta(T(\boldsymbol{X}) = Y) - \max_{j \neq Y} \mathbb{P}_\Theta(T(\boldsymbol{X}) = j) < 0\Big), \quad \text{almost surely.}$$

An upper bound for the generalization error was proven by Breiman. We start by defining the margin function of a random forest as

$$mr(\boldsymbol{X}, Y) = \mathbb{P}_\Theta(T(\boldsymbol{X}) = Y) - \max_{j \neq Y} \mathbb{P}_\Theta(T(\boldsymbol{X}) = j), \tag{3.12}$$

with generalization error $PE^* = \mathbb{P}_{\boldsymbol{X},Y}(mr(\boldsymbol{X}, Y) < 0)$. We define $s = \mathbb{E}_{\boldsymbol{X},Y}[mr(\boldsymbol{X}, Y)]$, and by Chebyshev's inequality, we have

$$PE^* \leq \mathbb{P}\Big(|mr(\boldsymbol{X}, Y) - s| > s\Big) \leq \frac{\mathbb{V}\mathbf{ar}(mr(\boldsymbol{X}, Y))}{s^2}. \tag{3.13}$$

Let $\widehat{j}(\boldsymbol{X}, Y) = \arg\max_{j \neq Y} \mathbb{P}_\Theta(T(\boldsymbol{X} = j))$, then the margin function in (3.12) can be written as

$$mr(\boldsymbol{X}, Y) = \mathbb{E}_\Theta\Big[\mathbf{1}\{T(\boldsymbol{X}) = Y\} - \mathbf{1}\{T(\boldsymbol{X}) = \widehat{j}(\boldsymbol{X}, Y)\}\Big],$$

where we define the raw margin function as $rmg(\Theta, \boldsymbol{X}, Y) = \mathbf{1}\{T(\boldsymbol{X}) = Y\} - \mathbf{1}\{T(\boldsymbol{X}) = \widehat{j}(\boldsymbol{X}, Y)\}$. The variance of $mr(\boldsymbol{X}, Y)$ can then be written as

$$\begin{aligned}
\mathbb{V}\mathbf{ar}(mr(\boldsymbol{X}, Y)) &= \mathbb{E}_{\boldsymbol{X},Y}[mr(\boldsymbol{X}, Y)^2] - \Big(\mathbb{E}_{\boldsymbol{X},Y}[mr(\boldsymbol{X}, Y)]\Big)^2 \\
&= \mathbb{E}_{\boldsymbol{X},Y}\Big[\Big(\mathbb{E}_\Theta(rmg(\Theta, \boldsymbol{X}, Y)\Big)^2\Big] - \Big(\mathbb{E}_{\boldsymbol{X},Y}\Big[\mathbb{E}_\Theta[rmg(\Theta, \boldsymbol{X}, Y)]\Big]\Big)^2 \\
&= \mathbb{E}_{\boldsymbol{X},Y}\Big[\mathbb{E}_{\Theta,\Theta'}\Big(rmg(\Theta, \boldsymbol{X}, Y) \cdot rmg(\Theta', \boldsymbol{X}, Y)\Big)\Big] - \\
&\mathbb{E}_{\boldsymbol{X},Y}\Big[\mathbb{E}_\Theta[rmg(\Theta, \boldsymbol{X}, Y)]\Big] \cdot \mathbb{E}_{\boldsymbol{X},Y}\Big[\mathbb{E}_{\Theta'}[rmg(\Theta', \boldsymbol{X}, Y)]\Big],
\end{aligned} \tag{3.14}$$

where Θ, Θ' are i.i.d random vectors. Then using the fact that

$$\rho_{X,X'} = \frac{\mathbf{Cov}(\boldsymbol{X}, \boldsymbol{X}')}{\sqrt{\mathbb{V}\mathbf{ar}(\boldsymbol{X}) \cdot \mathbb{V}\mathbf{ar}(\boldsymbol{X}')}} = \frac{\mathbb{E}[\boldsymbol{X}\boldsymbol{X}'] - \mathbb{E}[\boldsymbol{X}]\mathbb{E}[\boldsymbol{X}']}{\sqrt{\mathbb{V}\mathbf{ar}(\boldsymbol{X}) \cdot \mathbb{V}\mathbf{ar}(\boldsymbol{X}')}},$$

and the linearity of expectation, the variance given in Equation (3.14) is now

$$\mathbb{V}\mathbf{ar}(mr(\boldsymbol{X}, Y)) = \mathbb{E}_{\Theta,\Theta'}\Big[\rho\Big(rmg(\Theta, \boldsymbol{X}, Y), rmg(\Theta', \boldsymbol{X}, Y)\Big) \cdot \sqrt{\mathbb{V}\mathbf{ar}(rmg(\Theta, \boldsymbol{X}, Y)) \cdot \mathbb{V}\mathbf{ar}(rmg(\Theta', \boldsymbol{X}, Y))}\Big].$$

For simplicity, we define the average correlation $\bar{\rho}$ as

$$\bar{\rho} = \frac{\rho\Big(rmg(\Theta, \boldsymbol{X}, Y), rmg(\Theta' \boldsymbol{X}, Y)\Big)}{\mathbb{E}_\Theta\Big[\mathbf{sd}(rmg(\Theta, \boldsymbol{X}, Y))\Big]^2},$$

where $\mathbf{sd}(\cdot)$ represents the standard deviation. Then

$$\begin{aligned}\mathbb{V}\mathbf{ar}(mr(\boldsymbol{X},Y)) &= \bar{\rho}\cdot\Big(\mathbb{E}_\Theta(\mathbf{sd}(rmg(\Theta,\boldsymbol{X},Y))\Big)^2\\ &\le \bar{\rho}\cdot\mathbb{E}_\Theta\mathbb{V}\mathbf{ar}(rmg(\Theta,\boldsymbol{X},Y))\\ &= \bar{\rho}\cdot\mathbb{E}_\Theta\Big(\mathbb{E}(rmg(\Theta,\boldsymbol{X},Y))^2 - s^2\Big)\\ &\le \bar{\rho}\cdot(1-s^2).\end{aligned}$$

Hence combine with (3.13), we obtain the upper bound for the generalization error:

$$PE^* \le \frac{\bar{\rho}\cdot(1-s^2)}{s^2}. \tag{3.15}$$

The bound in (3.15) shows that the two ingredients involved in the generalization error for infinite random forests are the strengths $s := \mathbb{E}_{\boldsymbol{X},Y}[mr(\boldsymbol{X},Y)]$ of each classifier and the correlation between them in terms of the raw margin function ([Bre01]). The ratio of the average correlation and s^2 is called the c/s^2 ratio, and the smaller the c/s^2 ratio, the bettwe the random forest. Further we may define s_j as

$$s_j = \mathbb{E}_{\boldsymbol{X},Y}(\mathbb{P}_\Theta(T(\boldsymbol{X})=Y) - \mathbb{P}_\Theta(T(\boldsymbol{X})=j)),$$

then we have

$$\begin{aligned}PE^* &= \mathbb{P}_{\boldsymbol{X},Y}\Big(\mathbb{P}_\Theta(T(\boldsymbol{X})=Y) - \max_{j\neq Y}\mathbb{P}_\Theta(T(\boldsymbol{X})=j) < 0\Big)\\ &\le \bigcup_j \mathbb{P}_{\boldsymbol{X},Y}\Big(\mathbb{P}_\Theta(T(\boldsymbol{X})=Y) - \mathbb{P}_\Theta(T(\boldsymbol{X})=j) < 0\Big)\\ &\le \sum_j \mathbb{P}_{\boldsymbol{X},Y}\Big(\mathbb{P}_\Theta(T(\boldsymbol{X})=Y) - \mathbb{P}_\Theta(T(\boldsymbol{X})=j) < 0\Big)\\ &\le \sum_j \frac{\mathbb{V}\mathbf{ar}\Big(\mathbb{P}_\Theta(T(\boldsymbol{X})=Y) - \mathbb{P}_\Theta(T(\boldsymbol{X})=j)\Big)}{s_j^2},\end{aligned}$$

where the variance can again be replaced by the average correlation ρ, as before.

3.5 Averaging Classifiers: Boosting

3.5.1 AdaBoost

The idea of boosting is to combine the outputs of many "weak" classifiers into a "strong one". An example from Freund and Schapire ([FS97]) illustrates this process as a gambler inviting his friends to give the best betting strategy: The gambler will wager a fixed sum of money in every round but will apportion his money among his friends based on how well they are doing. Boosting was originally designed for classification problems, but it could be generalized to regression problems and can be implemented on decision trees. Compare to bagging and random forests which grow decision trees based on independent bootstrap sampling, boosting does not involve bootstrap sampling, and the trees are grown sequentially. At each step, a tree is grown using information from previously grown trees.

One of the most famous boosting algorithm due to Freund and Schapire is called AdaBoost.M1 ([FS97]). Consider a two class problem $Y \in \{-1, 1\}$ (a multi-class version can be generalized) and $\boldsymbol{X}$ as the data set. For a given classifier f, the misclassification rate (or error rate) on the training sample is defined as

$$\bar{\text{err}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{f(\boldsymbol{x}_i) \neq y_i\}, \tag{3.16}$$

where N is the number of training data. The purpose of boosting is to sequentially apply the "weak" classification problem (i.e., whose error rate is only slightly better then random guessing) to the repeatedly modified data. Hence by doing so K times, we have a sequence of K "weak" classifiers $f_1, f_2, \cdots, f_K$, and the final classifier is a weighted sum of those K "weak" classifiers:

$$f(\boldsymbol{x}) = \text{sgn}\left(\sum_{i=1}^{K} \alpha_i f_i(\boldsymbol{x})\right),$$

where $\alpha_1, \cdots, \alpha_K$ are parameters which can be computed by the algorithm. Apart from α_i's, each training data is also assigned a weight parameter ω_i. The weight are individually adjusted at each step in order to control the misclassification rate. At each step, the weights ω_i increases if $\boldsymbol{x}_i$ was falsely classified step and decreases if $\boldsymbol{x}_i$ was correctly classified in the previous step. By doing so, data which are difficult to classify will receive increasing influence ans weight, hence each successive classifier is forced to concentrate on those data which were misclassified ([HTF09]). The expression (3.16) is the case where all the weights are set to $\omega_i = 1/N$, which is also a natural choice for the initiating weights in Algorithm 5, since usually we have no reason to favor any of the samples in the beginning ([FS97]). The details of the AdaBoost algorithm is given below:

Algorithm 5 AdaBoost.M1.

Require: Training dataset $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$; Number of iterations K.
Ensure: Final classifier f.

function ADABOOST($\mathcal{S}$)
  for $i = 1, \cdots, K$ **do**
    for $j = 1, \cdots, N$ **do**
      Initialize the weights $\omega_j \leftarrow 1/N$;

      Fit a classifier f_i using $\mathcal{S}$ and ω_i;

      $\text{err}_i \leftarrow \dfrac{\sum_{j=1}^{N} \omega_j \cdot \mathbf{1}\{f_i(\boldsymbol{x}_j) \neq y_j\}}{\sum_{j=1}^{N} \omega_j}$; $\quad \alpha_i \leftarrow \log((1 - \text{err}_i)/\text{err}_i)$;

      Update the weight: $\omega_j \leftarrow \omega_j \cdot \exp(\alpha_i \cdot \mathbf{1}\{f_i(\boldsymbol{x}_j) \neq y_j\})$.
    end for
  end for
  Return The final decision $f(\cdot) = \text{sgn}\left(\sum_{i=1}^{K} \alpha_i f_i(\cdot)\right)$
end function

We next provide an simulation on how AdaBoost algorithm can turn a "weak" classifier into a "strong" one. This simulation can also be found in Hastie et. al. (Chapter 10.2) [HTF09]. We would consider 10 features X_1 to X_{10}, each is a standard normal random variable and X_i's are independent. The class label $Y \in \{-1, 1\}$ for each data $\boldsymbol{X} = (X_1, \cdots, X_{10})^\top$ is determined by

$$Y = \begin{cases} 1 & \text{if } \sum_{i=1}^{10} X_i^2 > 9.34 \\ 0 & \text{otherwise} \end{cases}.$$

Since we know $\sum_{i=1}^{10} X_i^2 \sim \chi^2_{(10)}$, we will take the median of a $\chi^2_{(10)}$ distribution so that the generated data Y are roughly equally distributed for both classes. We have generated $2,000$ training data, $1,000$ approximately for each label, and an additional $10,000$ testing data. We use different tree-based classification methods to obtain a classifier based on the training data, and test on the testing data. The test error is defined through (3.16).

The first classifier is a "single stump", i.e., a decision tree with only one decision node and two children. This classifier has a test error of roughly 45.8%, which is only around $\sim$ 4% better than random guessing. However, if we apply AdaBoost.M1 algorithm [5] to the single stump with 400 iterations, we would obtain a weighted classifier of test error around 5.8%, significantly lower than single stump. The Figure 19 shows the relation between the number of iterations / numbers of trees and the test error:

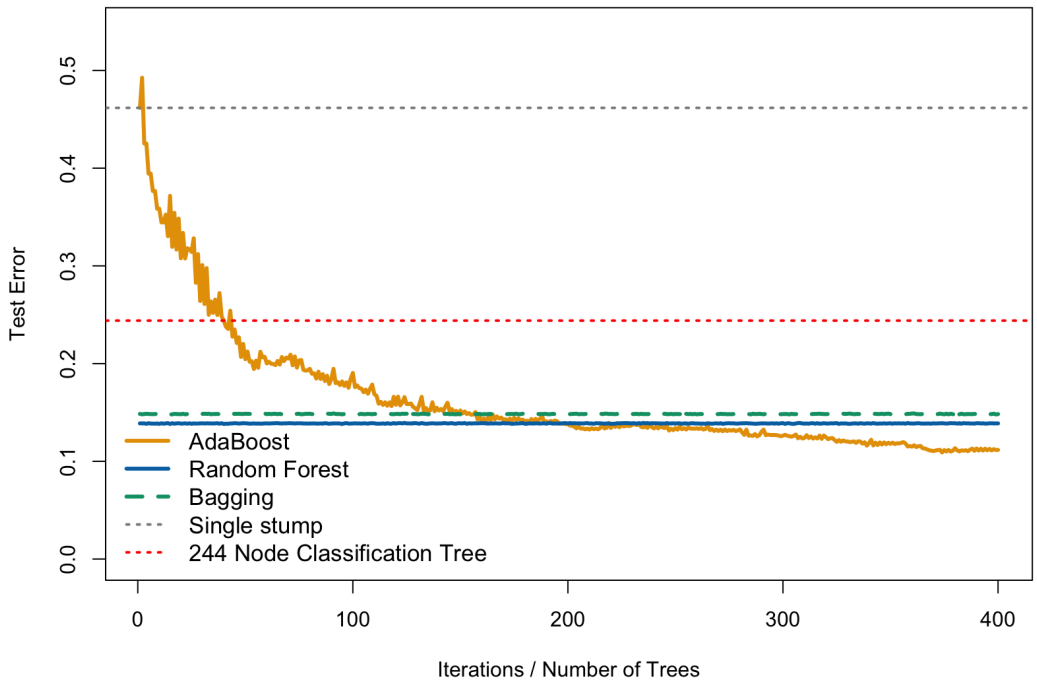

Figure 19: The performance of different tree-based classifiers measured on test error. We first can observe that the test error is significantly lower than the single stump and a 244 node classification tree using CART algorithm [4]; We can see that the test error of AdaBoost is also lower than random forest and bagging after around 200 iterations.

3.5.2 Loss Functions

Boosting is a way of fitting an additive expansion in a set of elementary basis functions ([HTF09]) which takes the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{K} \beta_i b(\boldsymbol{x}, \gamma_i),$$

In the example of AdaBoost, the basis functions are classifiers $f_i(\boldsymbol{x}) \in \{-1, 1\}$. The models are fit by minimizing the following loss

$$\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, \beta b(\boldsymbol{x}_i, \gamma)), \tag{3.17}$$

for some basis function b and parameters β, γ. Forward stagewise additive modeling algorithm (6) solves the optimization problem in (3.17) through the following algorithm:

Algorithm 6 Forward Stagewise Additive Modeling

1: **function** FORWARD STAGEWISE ADDITIVE MODELING
2: Initialize $f_0(\boldsymbol{x}) = 0$.
3: **for** $i = 1, \cdots, K$ **do**
4:
$$(\beta_i, \gamma_i) \leftarrow \arg\min_{\beta,\gamma} \sum_{j=1}^{N} L(y_i, f_{i-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i, \gamma)).$$
5: $f_i(\boldsymbol{x}) \leftarrow f_{i-1}(\boldsymbol{x}) + \beta_i b(\boldsymbol{x}, \gamma_i)$.
6: **end for**
7: **end function**

At each iteration i, one solves for the optimal basis function $b(\boldsymbol{x}, \gamma_i)$ and corresponding coefficient β_i to add to the current expansion $f_{i-1}(\boldsymbol{x})$. This produces $f_i(\boldsymbol{x})$, and the process is repeated. This algorithm approximates the solution to (3.17) by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those that have already been added.

The AdaBoost algorithm (5) is equivalent to forward stagewise additive modeling algorithm (6) with the exponential loss function defined as

$$L(y, f(\boldsymbol{x})) = \exp(-yf(\boldsymbol{x})),$$

(see Hastie et. al., Chapter 10.4. [HTF09]). Suppose $Y \in \{-1, 1\}$, at the population level, minimizing the exponential loss is equivalent to the following minimization problem:

$$f^*(\boldsymbol{x}) := \arg\min_{f(\boldsymbol{x})} \mathbb{E}_{Y|\boldsymbol{x}}\left[e^{-Yf(\boldsymbol{x})}\right]. \tag{3.18}$$

Since $Y|\boldsymbol{x}$ is a discrete random variable which only takes value -1 or 1, we can rewrite (3.18) as

$$\mathbb{E}_{Y|\boldsymbol{x}}\left[e^{-Yf(\boldsymbol{x})}\right] = e^{-f(\boldsymbol{x})} \cdot \mathbb{P}(Y = 1|\boldsymbol{x}) + e^{f(\boldsymbol{x})} \cdot \mathbb{P}(Y = -1|\boldsymbol{x}). \tag{3.19}$$

From (3.19), it is obvious that the minimizer $f^*(\boldsymbol{x})$ to (3.18) is given by

$$f^*(\boldsymbol{x}) = \frac{1}{2}\log\frac{\mathbb{P}(Y=1|\boldsymbol{x})}{\mathbb{P}(Y=-1|\boldsymbol{x})}.$$

Thus, AdaBoost is estimating one-half of the log-odds of $\mathbb{P}(Y=1|\boldsymbol{x})$, we can also see that it justifies using the sign function as the classification rule.

4 Neural Networks Methods

4.1 Basic Structures of Neural Networks

In recent years, neural networks have emerged as, by far, the most important machine learning technology for practical applications [BB24]. The development of neural networks is based on the approximation theory, where the universal approximation theorem (Cybenko, [Cyb89]) states that any continuous function can be well approximated by a single hidden layer neural network to any desired accuracy. Suppose the data has feature vector $\boldsymbol{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_p \end{pmatrix}^\top \in \mathbb{R}^p$ and has 2 different class labels $y \in \{0, 1\}$, one way to build a classifier is to consider a weighted sum of features composed with an *activation function* σ:

$$f(\boldsymbol{x}) = \sigma\left(b + \sum_{i=1}^{p} w_i x_i\right), \tag{4.1}$$

where $w_i \in \mathbb{R}, i \in [p]$ are weights and $b \in \mathbb{R}$ is a bias term. $\sigma(\cdot)$ is known as the activation function. The goal of the activation function is to introduce non-linearity, common activation functions include the sigmoid function $\sigma(x) = \dfrac{1}{1+e^{-x}}$; tanh function $\sigma(x) = \tanh(x)$, the ReLu and GeLU function, and so on. So far, the expression in (4.1) contains only an input layer and an output layer. The goal of neural network is to implement several hidden layers in between and view (4.1) as a composition of different weighted linear combinations equipped with activation functions. Now suppose we have one hidden layer (as illustrated in Figure 20),

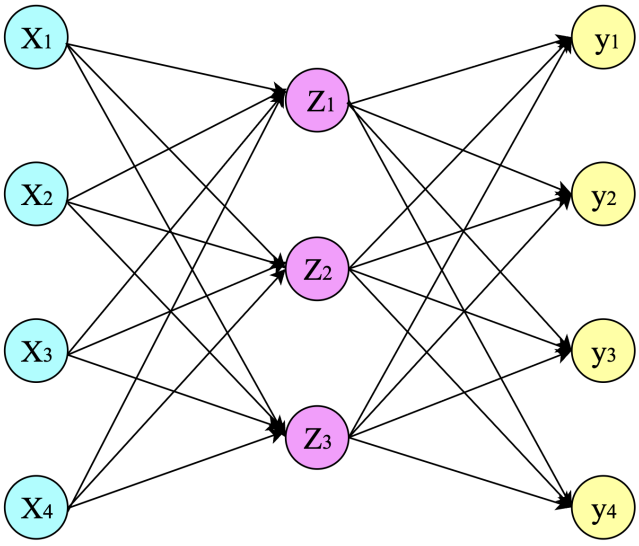

Figure 20: An example of a single layer, fully connected, feed forward neural network. Blue neurons represent the input layer, each neuron represents one feature; Purple neurons represent the hidden layer; Yellow neurons represent the output layer, where each neuron represents a different class label in the task of classification.

In Figure 20, the input data $\boldsymbol{x}$ has 4 features, namely x_1, x_2, x_3, x_4. The output $\boldsymbol{y}$ represents the class labels. The hidden layer (shown in purple) in our case is given by neurons z_1, z_2, z_3.

Each z_i (or $z_i^{(1)}$) can be viewed as the result of an activation function σ applied to a weighted sum of x_i's with a bias term:

$$z_i^{(1)} = \sigma\left(b_i^{(1)} + \sum_{j=1}^{4} w_{ij}^{(1)} x_j\right), \quad \text{for } i = 1, 2, 3, \tag{4.2}$$

The equation (4.2) can also be written in a more compact way using matrix representation:

$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{b}^{(1)} + \boldsymbol{W}^{(1)}\boldsymbol{x}\right),$$

where $\sigma(\cdot)$ is applied component-wise, and we define the linear combination inside the activation function as

$$\boldsymbol{\alpha}^{(1)} = \boldsymbol{b}^{(1)} + \boldsymbol{W}^{(1)}\boldsymbol{x}.$$

For the output layer, the idea is similar: The linear combination term is given by

$$\alpha_i^{(2)} = b_i^{(2)} + \sum_{j=1}^{3} w_{ij}^{(2)} z_i^{(1)}, \quad \text{or equivalently } \boldsymbol{\alpha}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)}.$$

The activation function $\sigma(\cdot)$ need not necessarily be the same as before. For multi-class classification, the most common one used is the *softmax* function defined by

$$y_i = \widehat{p}_i = \frac{e^{\alpha_j^{(2)}}}{\sum_{i=1}^{4} e^{\alpha_j^{(2)}}},$$

and the class label of $\boldsymbol{x}$ is determined by the label i that maximizes $\widehat{p}_i$. From here, the structure of a multi-hidden layer neural network can be easily generalized. Assume we have n hidden layers, and at hidden layer i, where $1 \leq i \leq n-1$, we have the following equations:

$$\begin{aligned}
\boldsymbol{\alpha}^{(i+1)} &= \boldsymbol{b}^{(i+1)} + \boldsymbol{W}^{(i+1)}\boldsymbol{z}^{(i)}, \\
\boldsymbol{z}^{(i)} &= \sigma\left(\boldsymbol{b}^{(i)} + \boldsymbol{W}^{(i)}\boldsymbol{z}^{(i-1)}\right), \\
\boldsymbol{z}^{(i+1)} &= \sigma(\boldsymbol{\alpha}^{(i+1)}).
\end{aligned}$$

From the last layer to the output layer, we have

$$\widehat{\boldsymbol{p}} = \text{softmax}\left(\boldsymbol{z}^{(n)}\right),$$

where the softmax function is applied component-wise, and the predicted class label is given by

$$y = \arg\max_i \widehat{p}_i.$$

4.2 Training Neural Networks

As we have seen before, a neural network has parameters $\boldsymbol{\theta} = \{\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \cdots\}$. The goal is to find the best parameter $\boldsymbol{\theta}$ for a desired neural network. For simplicity, let us assume fitting a single hidden layer neural network $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ on the data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$, where $\boldsymbol{x} \in \mathbb{R}^p$, $y = \{1, \cdots, M\}$, and the hidden layer has K neurons.

The parameters are given by $\boldsymbol{\theta} = \{\boldsymbol{b}^{(1)}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(2)}, \boldsymbol{W}^{(2)}\}$, where $\boldsymbol{b}^{(1)} \in \mathbb{R}^K, \boldsymbol{W}^{(1)} \in \mathbb{R}^{K \times p}$, $\boldsymbol{b}^{(2)} \in \mathbb{R}^M, \boldsymbol{W}^{(2)} \in \mathbb{R}^{M \times K}$. For each hidden layer $z_i^{(1)}, i \in [K]$, we have

$$z_i^{(1)} = \sigma\left(b_i^{(1)} + \sum_{j=1}^{p} w_{ij}^{(1)} x_j\right), \tag{4.3}$$

and the output for $y_i, i \in [M]$ is given by

$$\widehat{p}_i = \text{softmax}\left(b_i^{(2)} + \sum_{j=1}^{K} w_{ij}^{(2)} z_j^{(1)}\right), \tag{4.4}$$

In classification, the optimal parameter $\boldsymbol{\theta}$ is obtained by minimizing the cross-entropy loss defined by

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\sum_{m=1}^{M} y_{im} \log(\widehat{p}_{im}), \tag{4.5}$$

where $y_{im} = 1$ if and only if the true class label of y_i is m, and $\widehat{p}_{im}$ can be computed by (4.3) and (4.4). The optimal parameter is hence

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}). \tag{4.6}$$

This optimization is typically carried out by gradient-based methods such as gradient descent or stochastic gradient descent [Rud17]. These methods require the gradient of the loss with respect to all weights and biases. In neural networks, such gradients are computed efficiently by *backpropagation* (Rumelhart et. al. [RHW86]), which applies the chain rule recursively from the output layer back to the earlier layers. Thus, backpropagation provides the gradient information, while gradient descent uses that information to update the parameters.

4.2.1 Gradient Descent

In the optimization problem defined in (4.6), we aim to find its minimum. If we first consider minimizing a univariate differentiable function $f(x) : \mathbb{R} \to \mathbb{R}$, then we know at its local minimum x_0, we have $f'(x_0 - h) < 0$ and $f'(x_0 + h) > 0$ for small values of $h > 0$. Hence if we choose an initial point $x^{(0)}$ and compute $f'(x^{(0)})$, we can try to move $x^{(0)}$ to the left if $f'(x^{(0)}) > 0$ and to the right if $f'(x^{(0)}) < 0$, and by doing so we are moving $x^{(0)}$ towards x_0. We may update the value $x^{(1)}$ as

$$x^{(1)} = x^{(0)} - \eta \cdot f'(x^{(0)}), \tag{4.7}$$

for some positive parameter η, where η is called the *learning rate*. For a multivariate function $f(\boldsymbol{x})$, a Taylor expansion around $\boldsymbol{x}$ will yield

$$f(\boldsymbol{x} + \boldsymbol{h}) = f(\boldsymbol{x}) + \boldsymbol{h}^\top \cdot \nabla f(\boldsymbol{x}) + \mathcal{O}(||\boldsymbol{h}||_2^2),$$

in other words, up to second-order terms in $\boldsymbol{h}$, the direction of steepest descent is given by the negative gradient. So the algorithm is given by

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - \boldsymbol{\eta} \cdot \nabla f(\boldsymbol{x}^{(n)}).$$

Part III
The Problem of Class Imbalance

5 Classification Paradigms and Evaluations

5.1 Three Classification Paradigms

In this section, we review three basic classification paradigms that are defined by different objective functions, namely classical classification (CC) paradigms; cost-sensitive (CS) paradigm and Neyman-Pearson (NP) paradigm. Classical classification paradigm aims to minimize the overall missclassification rate; cost-sensitive paradigm aims to minimize a weighted sum of type (I) and (II) error; Neyman-Pearson paradigm aims to minimize type (II) error subject to a constraint induced by type (I) error [FZT21].

In binary classification case, let $f : \mathcal{X} \to \{0, 1\}$ be the classifier, $(\boldsymbol{X}, y)$ be the data point. Misclassification rate is defined by $\mathbb{E}[\mathbf{1}(f(\boldsymbol{X}) \neq y)]$. The classical classification paradigm chooses the optimal classifier f^* by

$$f^* = \arg\min_f \mathbb{E}[\mathbf{1}(f(\boldsymbol{X}) \neq y)], . \tag{5.1}$$

The solution to (5.1) is given by $f^* = \mathbf{1}(\eta(\boldsymbol{X}) > 1/2)$, where $\eta(\boldsymbol{X}) = \mathbb{E}[y|\boldsymbol{X} = \boldsymbol{x}]$. To see this, first note that $\mathbb{E}[\mathbf{1}(f(\boldsymbol{X}) \neq y)] = \mathbb{P}(f(\boldsymbol{X}) \neq y)$. Let f^* be the optimal classifier, f be any other classifier, then

$$\begin{aligned}
\mathbb{P}(f(\boldsymbol{X}) \neq y) &= 1 - \mathbb{P}(f(\boldsymbol{X}) = y) \\
&= 1 - \mathbb{P}(f(\boldsymbol{X}) = 0, y = 0|\boldsymbol{X} = \boldsymbol{x}) - \mathbb{P}(f(\boldsymbol{X}) = 1, y = 1|\boldsymbol{X} = \boldsymbol{x}) \\
&= 1 - \mathbf{1}\{f(\boldsymbol{X}) = 0\} \cdot \mathbb{P}(y = 0|\boldsymbol{X} = \boldsymbol{x}) - \mathbf{1}\{f(\boldsymbol{X}) = 1\} \cdot \mathbb{P}(y = 1|\boldsymbol{X} = \boldsymbol{x}) \\
&= 1 - \mathbf{1}\{f(\boldsymbol{X}) = 0\} \cdot (1 - \eta(\boldsymbol{X})) - \mathbf{1}\{f(\boldsymbol{X}) = 1\} \cdot \eta(\boldsymbol{X}).
\end{aligned}$$

Hence

$$\begin{aligned}
&\mathbb{P}(f(\boldsymbol{X}) \neq y) - \mathbb{P}(f^*(\boldsymbol{X}) \neq y) \\
&= \eta(\boldsymbol{X}) \cdot (\mathbf{1}\{f^*(\boldsymbol{X}) = 1\} - \mathbf{1}\{f(\boldsymbol{X}) = 1\}) + (1 - \eta(\boldsymbol{X}))(\mathbf{1}\{f^*(\boldsymbol{X}) = 0\} - \mathbf{1}\{f(\boldsymbol{X}) = 0\}) \\
&= (2\eta(\boldsymbol{X}) - 1) \cdot (\mathbf{1}\{f^*(\boldsymbol{X}) = 1\} - \mathbf{1}\{f(\boldsymbol{X}) = 1\}) \\
&\geq 0.
\end{aligned}$$

In the problem of class imbalance, the classical classification paradigm may not be the ideal choice. If the dataset consists 90% of the majority class and 10% of the minority class, then a classifier which simply involves random guessing will satisfy $\mathbb{P}(f(\boldsymbol{X}) = y) = 0.9$. Furthermore, we decompose the overall classification $R(f)$ error in terms of

$$R(f) = \pi_0 R_0(f) + \pi_1 R_1(f),$$

where $R_0(f) = \mathbb{P}(f(\boldsymbol{X}) = 1|y = 0), R_1(f) = \mathbb{P}(f(\boldsymbol{X}) = 0|y = 1)$ are denoted as type (I), (II) error and π_0, π_1 are the true proportion of the population with class label $0, 1$, respectively.

Then in the imbalanced case where $\pi_0 << \pi_1$, minimizing $R(f)$ could sometimes result in a larger type (I) error [FZT21]. In the scenarios involve disease diagnosis where misclassifying a diseased person as healthy is more serious than misclassifying a healthy person as diseased, the CC-paradigm is also not optimal to serve the user's purpose.

As we have seen before, the cost of type (I) error is usually larger than the cost of type (II) error (for example, the disease diagnosis, spam email filtering, etc.). The goal of cost-sensitive paradigm is to assign a cost function $C(f(\boldsymbol{X}), y)$ to type (I) and (II) error. Let $C_0 = C(1,0), C_1 = C(0,1)$ be the cost of type (I) and (II) error, and we define $C(0,0) = C(1,1) = 0$. Then the CS-paradigm aims to minimize

$$\begin{aligned} R_c(f) &= \mathbb{E}[C(f(\boldsymbol{X}), y)] \\ &= C_0 \mathbb{P}(f(\boldsymbol{X}) = 1, y = 0) + C_1 \mathbb{P}(f(\boldsymbol{X}) = 0, y = 1) \\ &= C_0 \mathbb{P}(f(\boldsymbol{X}) = 1 | y = 0) \cdot \mathbb{P}(y = 0) + C_1 \mathbb{P}(f(\boldsymbol{X}) = 0 | y = 1) \cdot \mathbb{P}(y = 1) \\ &= C_0 \pi_0 R_0(f) + C_1 \pi_1 R_1(f). \end{aligned}$$

With the knowledge of C_0, C_1 at the population level, the optimal classifier is

$$f_C^* = \arg\min_f R_c(f) = \mathbf{1}\left\{\eta(\boldsymbol{X}) > \frac{C_0}{C_0 + C_1}\right\},$$

where $\eta(\boldsymbol{X}) = \mathbb{E}[y|\boldsymbol{X} = \boldsymbol{x}]$. And it reduces to the optimal classifier for CC-paradigm when we set $C_0 = C_1 = 1/2$. The drawback of CS-paradigm is obvious: Assigning the cost C_0, C_1 is not straightforward sometimes. On the other hand, despite minimizing $R_c(f)$, there is no guarantee on the probabilistic bounds on type (I) error under a pre-specified level [FZT21].

A third paradigm to control the asymmetric classification error is called the Neyman-Pearson (NP) paradigm (Tong et. al. [TFL18]). It aims to minimize type (II) error while controlling type (I) error under a certain level α:

$$f_\alpha^* := \arg\min_{f, R_0(f) \le \alpha} R_1(f),$$

which can be solved by the umbrella algorithm proposed by Tong et. al. [TFL18].

5.2 An Overview of Methods for Imbalanced Classification

In the problem of class imbalance, the accuracy of the classifier induced from the model would be not acceptable, and the tendency of the classification model is biased towards the majority class [KBGB21]. In order to tackle this issue, there are three commonly approach: Data level approach; Algorithm level approach and combing methods.

In the data level approach, the main objective is to balance the skewed dataset via resampling. It involves over sampling of the minority class or under sampling of the majority class. Common over sampling methods include random oversampling and the SMOTE algorithm [CBHK02]. Random oversampling simply duplicate the data points of the minority class at random. It generates new data points by selecting the existing data points from the minority class at random with replacement. The advantage of random oversampling is that, it is simple to implement; does not require complex algorithms and does not require any prior knowledge to the class distribution. However, random oversampling does not generate any new data points, so the oversampled data may be highly correlated and may increase the likelihood of

overfitting. The SMOTE (Synthetic Minority Over-sampling Technique) algorithm proposed by Chawla et. al. [CBHK02] is another oversampling technique which generates new data points for the minority class by interpolating the kth nearest neighbors. It first randomly selects a data point $\boldsymbol{X}$ from the minority class and compute its kth nearest neighbors (by default $k = 5$ [CBHK02]). It then selects one of the nearest neighbors $\boldsymbol{X}_j$ at random and a new data point is generated along the line segment within $\boldsymbol{X}\boldsymbol{X}_j$ at random (see Section 5.4 for more detail).

The under-sampling methods directly discard the data points from the majority class. Two common methods are random under-sampling and cluster-based under-sampling. Random under-sampling behaves in the same way as random over-sampling, where we randomly eliminate the data points from the majority class. Cluster-based sampling involves clustering algorithm, where the majority data points are divided into different clusters such that the number of clusters is equal to the number of data points in the minority class. Then one data point is chosen from each cluster at random. Under-sampling may lead to the lose of information since a large proportion of the majority class is discarded. In practice, when the class imbalance is huge, the combination of over-sampling and under-sampling are also used.

In the algorithm level approach, the main objective is to keep the training data invariable and adjust the classification algorithm to facilitate the task of learning specifically related to minority class [KBGB21]. Common methods include ensemble learning, thresholding and cost-sensitive learning.

5.3 Evaluation Metrics

For a given two-class classification and a classifier f, we define $y = 1$ as the positive class label and $y = 0$ as the negative class label. The classification result can be summarized into four groups: true positive (TP); true negative (TN); false positive (FP); false negative (FN). Assume the positive class has label 1 and negative class has label 0, then those quantities can be computed by

$$TP = \sum_i \mathbf{1}(f(\boldsymbol{X}_i) = 1, y_i = 1), \quad TN = \sum_i \mathbf{1}(f(\boldsymbol{X}_i) = 0, y_i = 0);$$

$$FP = \sum_i \mathbf{1}(f(\boldsymbol{X}_i) = 1, y_i = 0), \quad FN = \sum_i \mathbf{1}(f(\boldsymbol{X}_i) = 0, y_i = 1).$$

The misclassification rate (also known as the empirical risk) of f is given by

$$R(f) = \frac{FP + FN}{TP + TN + FP + FN},$$

which can be written as $R = \pi_0 R_0(f) + \pi_1 R_1(f)$. π_0, π_1 are the proportions of data with label $0, 1$ in the original data:

$$\pi_0 = \frac{TN + FP}{TP + FP + TN + FN}, \quad \pi_1 = \frac{TP + FN}{TP + TN + FP + FN},$$

and $R_0(f), R_1(f)$ are the risk of misclassifying a data with label $0, 1$:

$$R_0(f) = \frac{FN}{TN + FN}, \quad R_1(f) = \frac{FP}{TP + FP},$$

hence $R(f) = \pi_0 R_0(f) + \pi_1 R_1(f)$. For cost-sensitive learning, the misclassification rate can also be written as $R_c(f) = C_0 \pi_0 R_0(f) + C_1 \pi_1 R_1(f)$. The accuracy of f is simply given by $1 - R(f)$.

Another two quantities to evaulate classification results are called *recall* and *precision*. They are defined by

$$\text{Recall} = \frac{TP}{TP + FN}, \quad \text{Precision} = \frac{TP}{TP + FP}. \tag{5.2}$$

In words, recall is the proportion of data with true class label 1 that are correctly identified by f. A high recall value means the number of false negative (FN) is relatively small, meaning less data with original label 1 are misclassified. Precision is the proportion of data with true class label 1 among all data that are predicted as label 1 by f. A high precision value means the number of false positive (FP) is relatively small, meaning less data with original label 0 are misclassified. The $F1$ score is the harmonic mean of the precision and recall, and is defined by

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The $F1$ score uses harmonic mean rather than arithmetic mean to ensure both precision and recall are reasonably high. If either precision or recall is low, the $F1$ score drops significantly, highlighting poor model performance. To better visualize the performance of the classifier, several plot have been introduced, including the Receiver Operating Characteristic (ROC) curve and Precision-Recall (PR) curve. The ROC curve plots the relations between the false positive rate (FPR) and true positive rate (TPR). They are defined by

$$\text{TPR} = \frac{TP}{TP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN}. \tag{5.3}$$

Usually we set false positive rate be the x axis and true positive rate be the y axis, and the ideal point is $(0, 1)$. The area under curve (ROC-AUC) is the area enclosed by the ROC curve and the x (false positive) axis. It is a measure on how well a classifier ranks positives above negatives. However, when the data is highly imbalanced, the ROC-AUC curves can present an overly optimistic view of classifiers' performance. This is because in the imbalanced setting (assuming the minority class is positive), the number of true negative (TN) is huge, which makes FPR defined in (5.3) fairly small; Meanwhile, the true positive rate (TPR) can be fairly large within the minority class, resulting a point at the top-left corner in the ROC curve. Precision-recall (PR) curves and their AUC is used as an alternative measurement for the imbalanced data. Instead of plotting false positive rate versus true positive rate, it describes the relations between precision and recall defined in (5.2). Figure 21 illustrates a graphical example of the ROC and PR curve for one simulation.

5.4 The SMOTE Over-sampling Algorithm

The SMOTE algorithm stands for Synthetic Minority Over-sampling Technique, proposed by Chawla et. al. [CBHK02], it is an over-sampling technique where the minority class is over-sampled by creating "synthetic" examples rather than oversampling with replacement. The minority class is oversampled by randomly taking each minority class sample and generating synthetic examples along the line segments jointing its k minority class nearest neighbors.

According to Chawla et. al. [CBHK02], the detail of the generation process is described as follows: First randomly chosen a sample from the minority class, then compute its k nearest minority class neighbors (by default $k = 5$). Randomly chose one of its nearest minority class neighbors and take the difference between the feature vector (sample) and that neighbor. Finally generate a random number between 0 and 1 and multiply it by the difference, and add it

to the feature vector (sample). This causes the generation of a new synthetic sample along the line segment between two specific samples, which forces the decision region of the minority class to become more general. The SMOTE algorithm is given below:

Algorithm 7 SMOTE (Synthetic Minority Over-sampling Technique)

function SMOTE(Minority class $\{(\boldsymbol{x}_i, 1)\}_{i=1}^{m}$, Target minority class size N)
  Initialize MinoritySize $= m$.
  if MinoritySize $< N$ **then**
    $\boldsymbol{x} \leftarrow$ Chose one sample from the minority class at random
    $\mathcal{S}(k, \boldsymbol{x}) \leftarrow$ The k nearest minority class neighbors of $\boldsymbol{x}$
    $\boldsymbol{x}' \leftarrow$ Chose one sample from $\mathcal{S}(k, \boldsymbol{x})$ at random
    $\alpha \leftarrow Uniform(0, 1)$
    Update the synthetic sample: $\boldsymbol{x}_{\text{new}} \leftarrow \boldsymbol{x} + \alpha \cdot (\boldsymbol{x}' - \boldsymbol{x})$
    Minority class $= \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m\} \cup \{\boldsymbol{x}_{\text{new}}\}$, MinoritySize $\leftarrow m + 1$.
  end if
  Output the synthetic minority class $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$.
end function

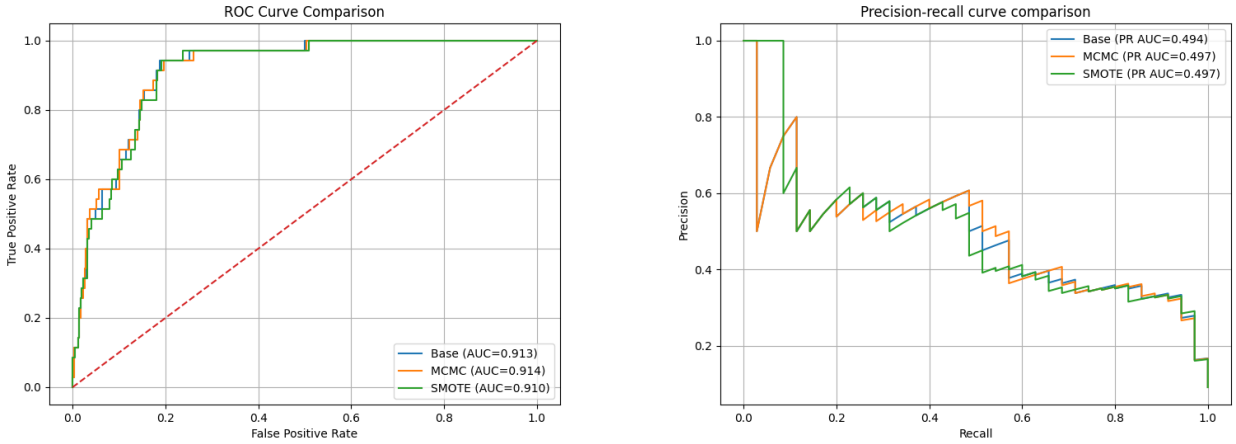

Figure 21: The ROC and PR curve for a logistic regression classifier based on three data sets: Base case (With class imbalance); MCMC (Simulate multiple data from the minority class via MCMC); SMOTE (Simulate multiple data from the minority class via SMOTE). The details of the simulation can be found in Section 6.

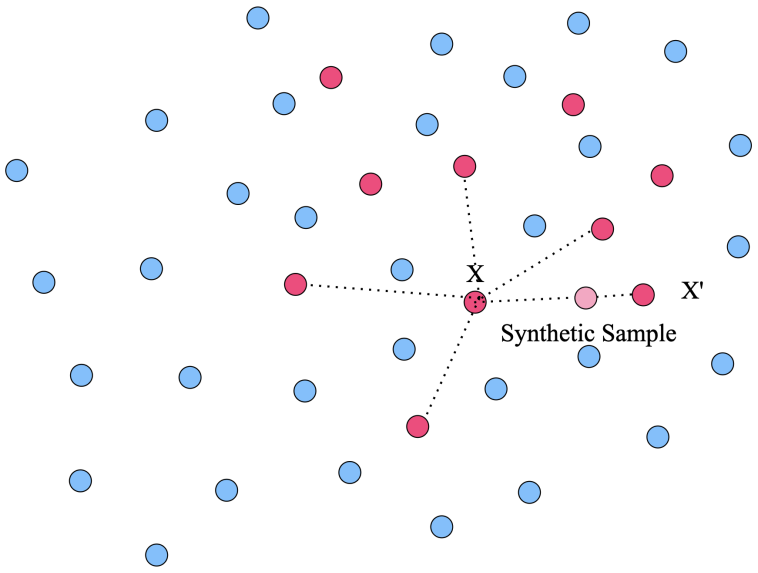

Figure 22: The graphical illustration of the SMOTE algorithm. Figure is inspired from Li et. al. [YYS+25].

Several variants of the SMOTE algorithm have been proposed, we briefly discuss Borderline-SMOTE (B-SMOTE, Han et. al. [HWM05]), enhanced SMOTE (ISMOTE, Li et. al. [YYS+25]) and Adaptive Synthetic Sampling (ADASYN, He et. al. [HBGL08]). Borderline-SMOTE aims to generate minority class samples near the borderline to better capture decision boundary and hence make better prediction. The samples on the borderline and the ones nearby are more apt to be misclassified than the ones far from the borderline, and thus more important for classification [HWM05]. ISMOTE algorithm expands the sample generation Space. Unlike SMOTE, which generates samples solely through linear interpolation between the line segments of two samples, ISMOTE allows new samples to be generated not only between existing samples but also around them. This approach effectively mitigates the issue of generating excessive samples in high-density regions, thereby reducing the risk of overfitting and improving the quality of the generated data [YYS+25]. ADASYN algorithm is based on idea of adaptively generating minority data samples according to their distributions, where more synthetic data is generated for minority class samples that are harder to learn compared to those that are easier to learn [HBGL08].

The procedure of the B-SMOTE algorithm is as follows: Let P, N denote the minority class and the majority class, respectively. Then for each $\boldsymbol{x} \in P$, we first find its k nearest neighbors from the entire data set. For each minority sample, we will classify it into three sets: Noise, Danger and Safe. A minority sample is classified into 'noise' if all its nearest neighbors belong to the majority class. Those samples are discarded by the algorithm and will not be considered. A minority sample is classified into 'Danger' if its majority class neighbors is larger than its minority class neighbors (given the minority class neighbors are non-zero), where in this case the sample has a higher chance of being misclassified. Finally a minority sample is classified into 'Safe' if its majority neighbors belong to the minority class. In this case, the sample is considered to be safe and they will not be used in the B-SMOTE algorithm. Below, 23 illustrates the B-SMOTE algorithm via a simulated data:

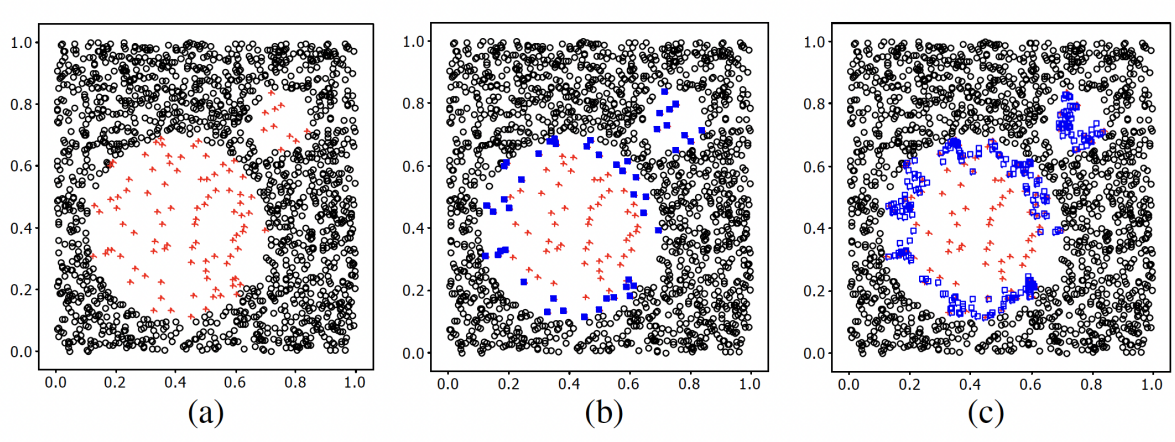

Figure 23: (a) The original data where black circles represent the majority class and red crosses represent the minority class. (b) The 'Danger' set from the minority class is represented in solid blue squares, which also indicates the borderline of the data. (c) The synthetic samples from the 'Danger' set is represented in hollow blue squares. The figure is taken from Han et. al. [HWM05]

The main difference between B-SMOTE and SMOTE is that, B-SMOTE algorithm generates synthetic data based on the minority class near the borderline, which in our case are the samples in the 'Danger' set. The rest of the B-SMOTE algorithm is almost identical to SMOTE, where a sample from 'Danger' set is randomly chosen and its nearest neighbors in the set are computed, and a new sample is generated along the line segment between that sample and a randomly chosen nearest neighbors.

The SMOTE algorithm restricts the synthetic data to be generated between the linear paths of minority samples, which may not accurately reflect the true distribution of the data. To address this limitation, the ISMOTE algorithm extends the spatial scope for generating new samples by adaptively adjusting their positions based on local density and distribution characteristics [YYS+25]. The procedure of ISMOTE is also similar to SMOTE: It first selects a random sample $\boldsymbol{x}$ from the minority class and find its k nearest minority class neighbors. Then it selects a random neighbor $\boldsymbol{x}'$ and generate a random number $\alpha \in (0,1)$ and produce the base synthetic sample $\boldsymbol{x}_s$ as in SMOTE. Then it examines the position of $\boldsymbol{x}_s$: When it is biased towards $\boldsymbol{x}'$, another random quantity is subtracted to generate a new sample near $\boldsymbol{x}$. Similarly, when $\boldsymbol{x}_s$ is closer to $\boldsymbol{x}$, the random quantity is added to generate a sample closer to $\boldsymbol{x}'$. That is, the new synthetic data $\boldsymbol{x}_{\text{new}}$ is given by

$$\boldsymbol{x}_{\text{new}} = \begin{cases} \boldsymbol{x} + \alpha \cdot (\boldsymbol{x}' - \boldsymbol{x}) - \beta \cdot (\boldsymbol{x}' - \boldsymbol{x}) & \text{if } \boldsymbol{x}_s \text{ is biased towards } \boldsymbol{x}' \\ \boldsymbol{x} + \alpha \cdot (\boldsymbol{x}' - \boldsymbol{x}) + \beta \cdot (\boldsymbol{x}' - \boldsymbol{x}) & \text{if } \boldsymbol{x}_s \text{ is biased towards } \boldsymbol{x} \end{cases},$$

where $\alpha, \beta \sim \text{Uniform}(0,1)$.

The procedure of the ADASYN algorithm is given as follow: Assume the minority class has size m, the total number of synthetic samples needed is G. first for each $\boldsymbol{x}_i$ belonging to the minority class, find its K nearest neighbors and compute

$$r_i = \frac{\Delta_i}{K}, \quad \forall i \in [m]$$

where Δ_i is the number of majority class samples belonging to K nearest neighbors of $\boldsymbol{x}_i$. It can be viewed as a measurement of weights for different minority class samples according to their level of difficulty in learning. A minority class sample surrounded by mostly the majority class has higher weights and thus more synthetic data needs to be generated. To better define the weight for each minority class sample, we truncate each r_i as

$$\widehat{r}_i = \frac{r_i}{\sum_{j=1}^{m} r_j}.$$

By doing so, $\widehat{r}_i$ is a probability mass function. Then at each $\boldsymbol{x}_i$, use SMOTE algorithm to generate g_i synthetic samples, where

$$g_i = \widehat{r}_i \cdot G.$$

Hence, the key idea of ADASYN algorithm is to use a probability mass function $\widehat{r}_i$ as a criterion to automatically decide the number of synthetic samples that needed to be generated for each minority data sample [HBGL08].

6 Simulation Study

6.1 Credit Card Fraud Detection

In this section, we investigate the `credit-card-fraud-detection` dataset from OpenML. The aim is to detect credit card fraud so the bank can prevent the transitions that customers did not make. In our implementation, the dataset is treated as a binary classification problem, where the target variable y is the fraud label and the remaining variables form the design matrix $\boldsymbol{X}$. The preprocessing step converts the target variable y into numeric $0/1$ (in our implementation, 1 is the minority class label) form and then creates a stratified train–test split so that the class imbalance is preserved in both subsets. The dataset contains $284,807$ observations, 29 predictor variables, and a highly imbalanced class distribution, with only 492 fraud cases against $284,315$ non-fraud cases. This means the minority class proportion is about 0.17%, making the dataset an appropriate benchmark for studying imbalanced classification.

The main challenge in this dataset is that standard classifiers trained directly on the original data tend to favor the majority class, which can lead to poor detection of fraudulent transactions. To address this issue, the analysis compares a baseline model with several oversampling methods, including regular SMOTE, Borderline-SMOTE, ADASYN, ISMOTE, and random oversampling.

To avoid overfitting, we split the dataset into 75% training (sample size $= 213,605$) and 25% testing (sample size $= 71,202$) where in each set the percentage of the minority class is roughly 0.17%. We will employ different oversampling methods to the training set only, and fit different classifiers to the synthetic training data. After fitting the classifiers, we will use them to test on the testing data. The performance is evaluated on the untouched test set using metrics suited to imbalanced learning, such as precision, recall, $F1$ score, ROC-AUC, PR-AUC. This setup allows us to assess whether synthetic minority oversampling can improve fraud detection performance without excessively increasing false positives. Our study comprises 3 parts:

- PART ONE: In this part, we analyze the performance of classification task using traditional classification techniques. The classification methods we implemented are Logistic regression (2.2), Linear discriminant analysis (LDA) (1.1); Quadratic discriminant analysis (QDA) (1.1); Random forest (3.4); AdaBoost (3.5.1) and Support vector machine (1.2, 2.1). We record the performance of different classifiers combined with over-sampling techniques as follow.

- PART TWO: In this part, we analyze the potential influence of high-dimensionality. We have implemented a random Gaussian noise feature, and examine the performance of those traditional classifiers introduced in PART ONE.

- PART THREE: In this part, we focus on the method using neural networks. We will employ a feed-forward, fully connected neural networks and analyze the resulting performances.

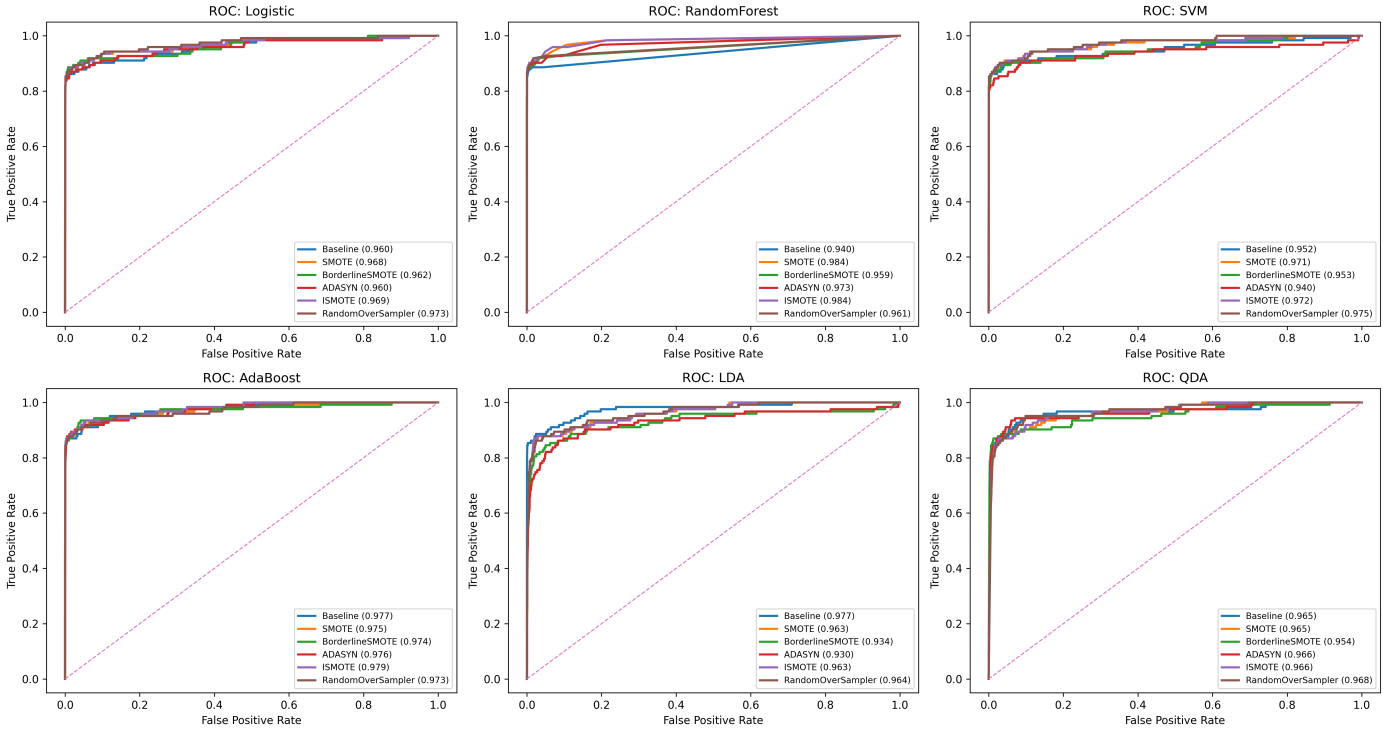

Figure 24: The ROC curve of different classifiers with over-sampling techniques.

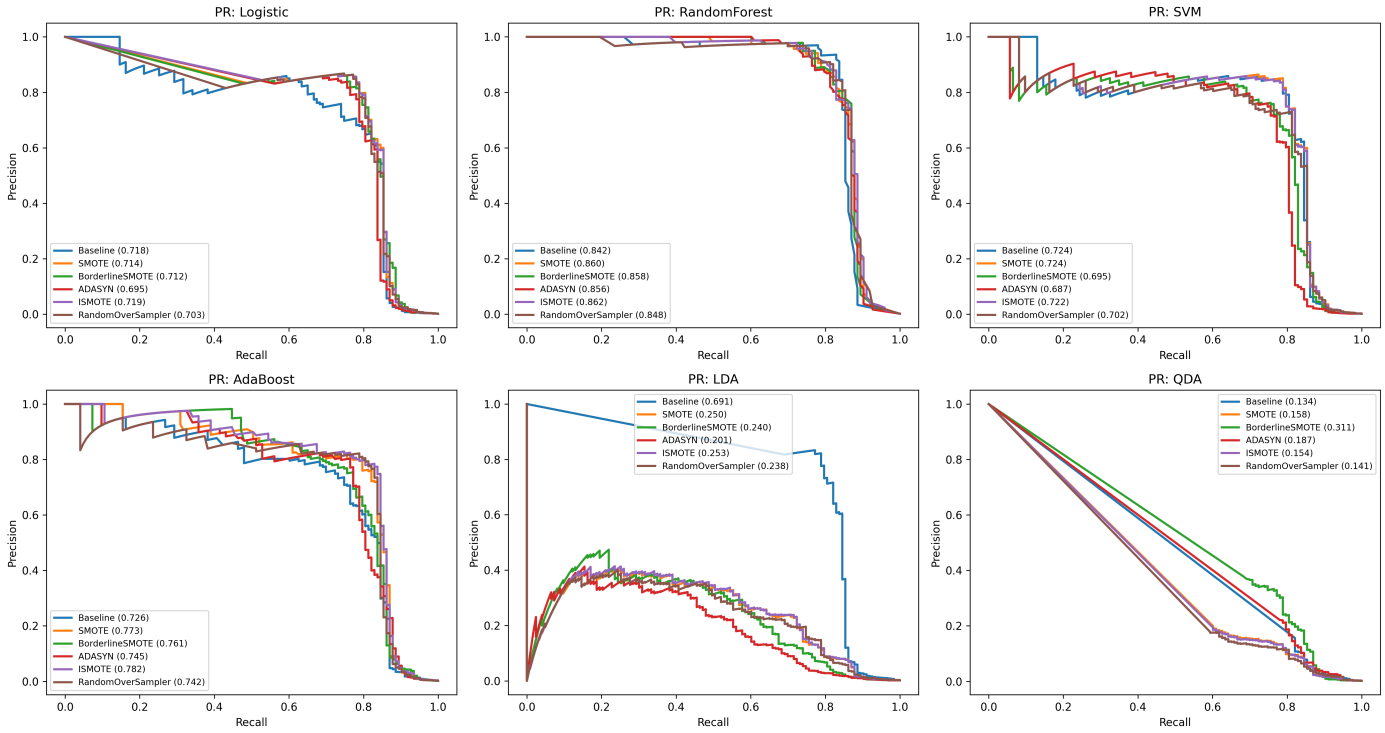

Figure 25: The PR curve of different classifiers with over-sampling techniques.

Classifier	Resampler	FP	FN	Precision	Recall	ROC_AUC	PR_AUC
AdaBoost	ISMOTE	1410	13	0.0724	0.8943	0.9786	0.7820
AdaBoost	Baseline	24	38	0.7798	0.6911	0.9767	0.7256
AdaBoost	ADASYN	2520	12	0.0422	0.9024	0.9758	0.7453
AdaBoost	SMOTE	1437	14	0.0705	0.8862	0.9754	0.7734
AdaBoost	BorderlineSMOTE	411	17	0.2050	0.8618	0.9735	0.7608
AdaBoost	RandomOverSampler	1192	15	0.0831	0.8780	0.9730	0.7421
LDA	Baseline	19	29	0.8319	0.7642	0.9766	0.6915
LDA	RandomOverSampler	892	26	0.0981	0.7886	0.9640	0.2378
LDA	SMOTE	882	25	0.1000	0.7967	0.9626	0.2498
LDA	ISMOTE	869	25	0.1013	0.7967	0.9626	0.2534
LDA	BorderlineSMOTE	1108	28	0.0790	0.7724	0.9336	0.2400
LDA	ADASYN	3383	25	0.0282	0.7967	0.9304	0.2005
Logistic	RandomOverSampler	1692	14	0.0605	0.8862	0.9725	0.7026
Logistic	ISMOTE	1313	14	0.0767	0.8862	0.9687	0.7191
Logistic	SMOTE	1293	14	0.0777	0.8862	0.9678	0.7142
Logistic	BorderlineSMOTE	445	16	0.1938	0.8699	0.9618	0.7118
Logistic	ADASYN	3200	15	0.0326	0.8780	0.9597	0.6951
Logistic	Baseline	14	47	0.8444	0.6179	0.9596	0.7181
QDA	RandomOverSampler	1631	19	0.0599	0.8455	0.9682	0.1412
QDA	ISMOTE	1514	19	0.0643	0.8455	0.9663	0.1535
QDA	ADASYN	2283	15	0.0452	0.8780	0.9655	0.1874
QDA	Baseline	1662	18	0.0594	0.8537	0.9654	0.1344
QDA	SMOTE	1502	19	0.0648	0.8455	0.9652	0.1578
QDA	BorderlineSMOTE	1229	16	0.0801	0.8699	0.9536	0.3109
RandomForest	ISMOTE	18	21	0.8500	0.8293	0.9837	0.8616
RandomForest	SMOTE	17	22	0.8559	0.8211	0.9836	0.8599
RandomForest	ADASYN	15	23	0.8696	0.8130	0.9731	0.8555
RandomForest	RandomOverSampler	4	31	0.9583	0.7480	0.9608	0.8483
RandomForest	BorderlineSMOTE	12	28	0.8879	0.7724	0.9590	0.8582
RandomForest	Baseline	5	26	0.9510	0.7886	0.9404	0.8424
SVM	RandomOverSampler	1400	14	0.0722	0.8862	0.9746	0.7016
SVM	ISMOTE	1072	15	0.0915	0.8780	0.9717	0.7223
SVM	SMOTE	1040	15	0.0941	0.8780	0.9706	0.7235
SVM	BorderlineSMOTE	462	18	0.1852	0.8537	0.9534	0.6947
SVM	Baseline	13	52	0.8452	0.5772	0.9518	0.7236
SVM	ADASYN	2866	18	0.0353	0.8537	0.9395	0.6874

Table 5: The performance comparison on the testing set of the `credit-card-fraud-detection` dataset.

The data has only 29 features, which is relatively small compared to the number of samples in each class. To examine the performance of different classifiers in high-dimensional setting, we have implemented a random Gaussian noise to the feature vector, where

$$\boldsymbol{\epsilon} \sim \mathcal{N}\Big(\mathbf{0}, \mathbb{I}_n\Big),$$

and $n = 20, 50, 100, 200$. The performance of different oversampling techniques with different noise levels have been recorded:

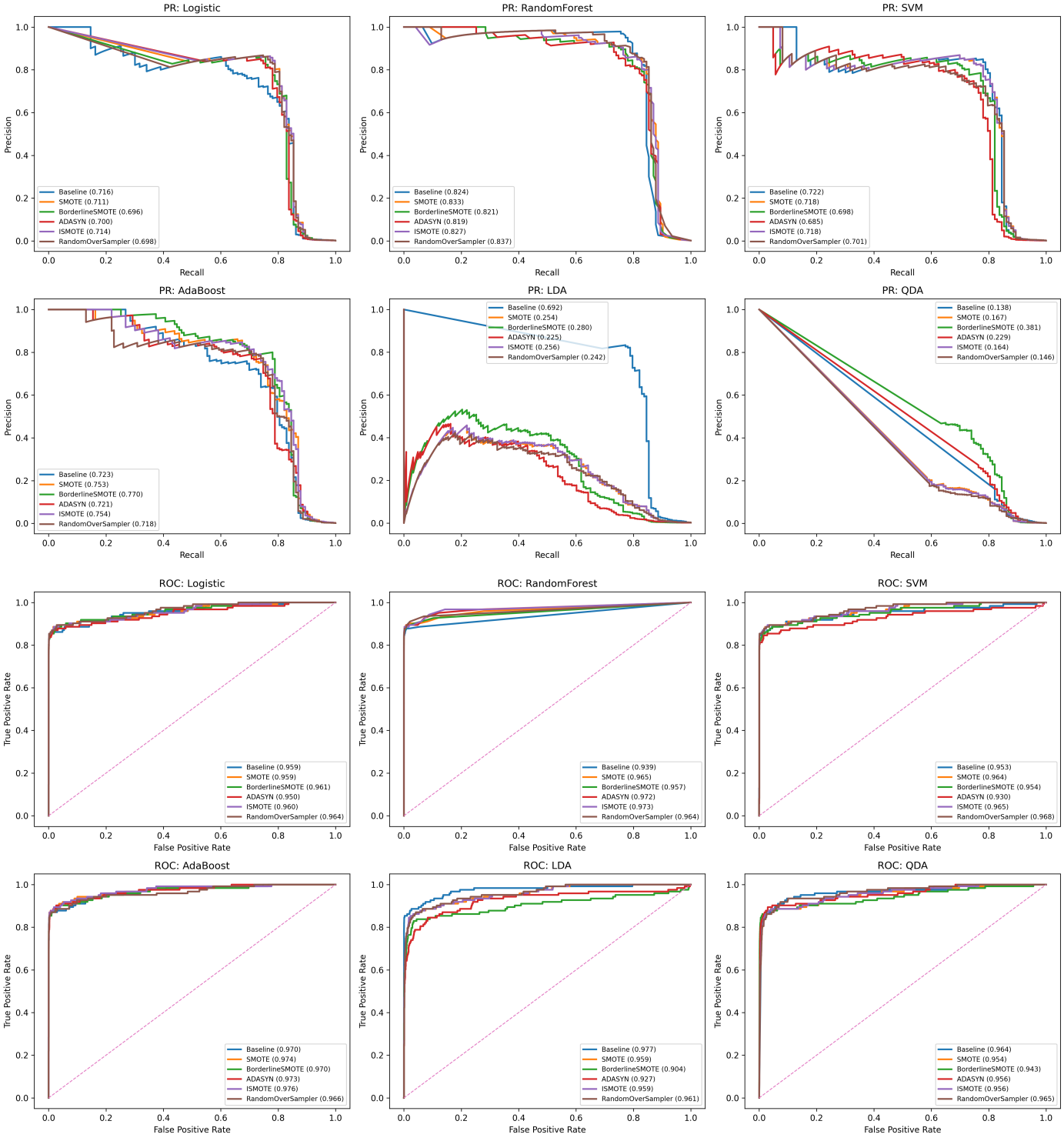

Figure 26: The PR_AUC and ROC_AUC curve when the number of Gaussian noise feature is 20.

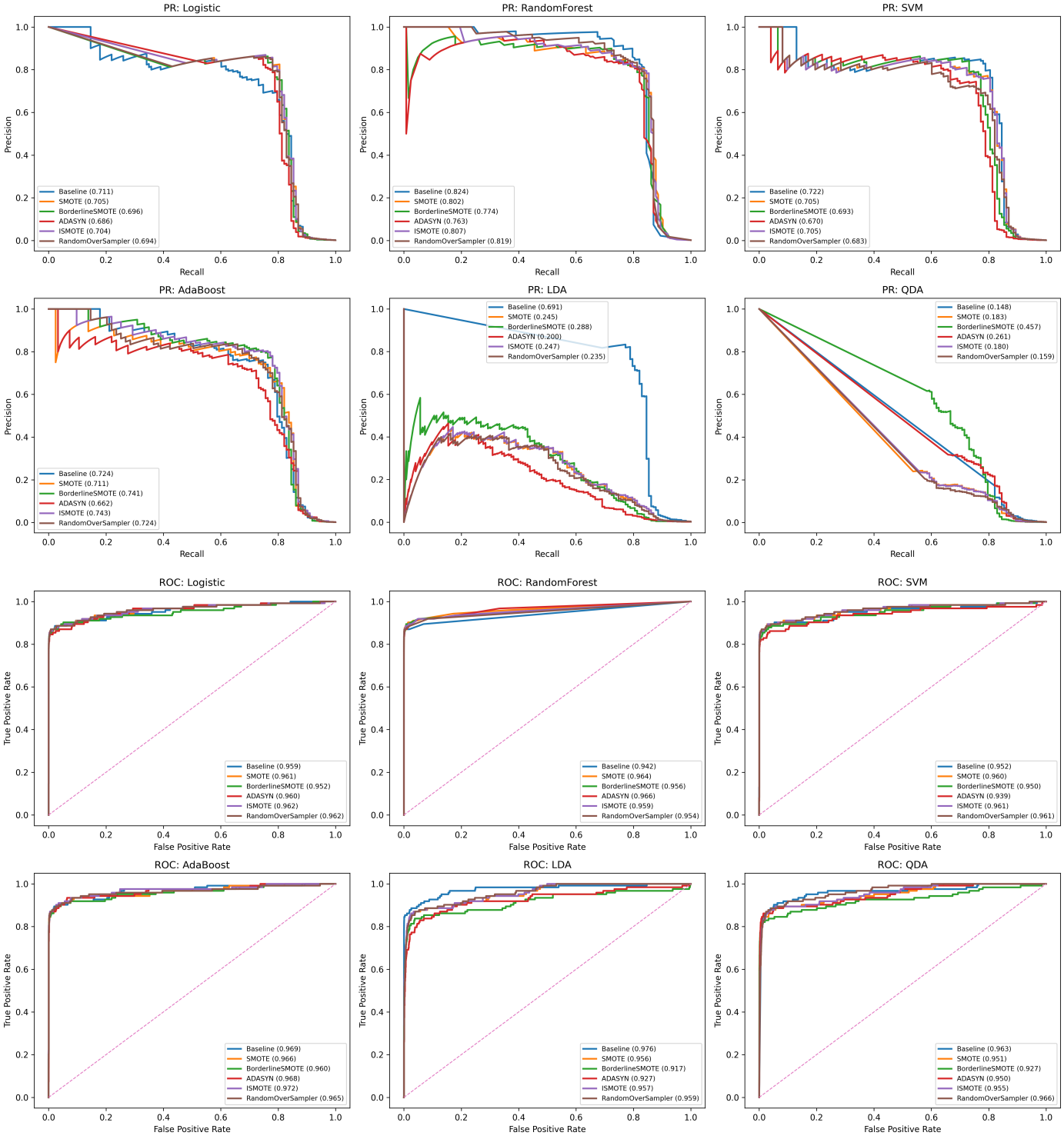

Figure 27: The PR_AUC and ROC_AUC curve when the number of Gaussian noise feature is 50.

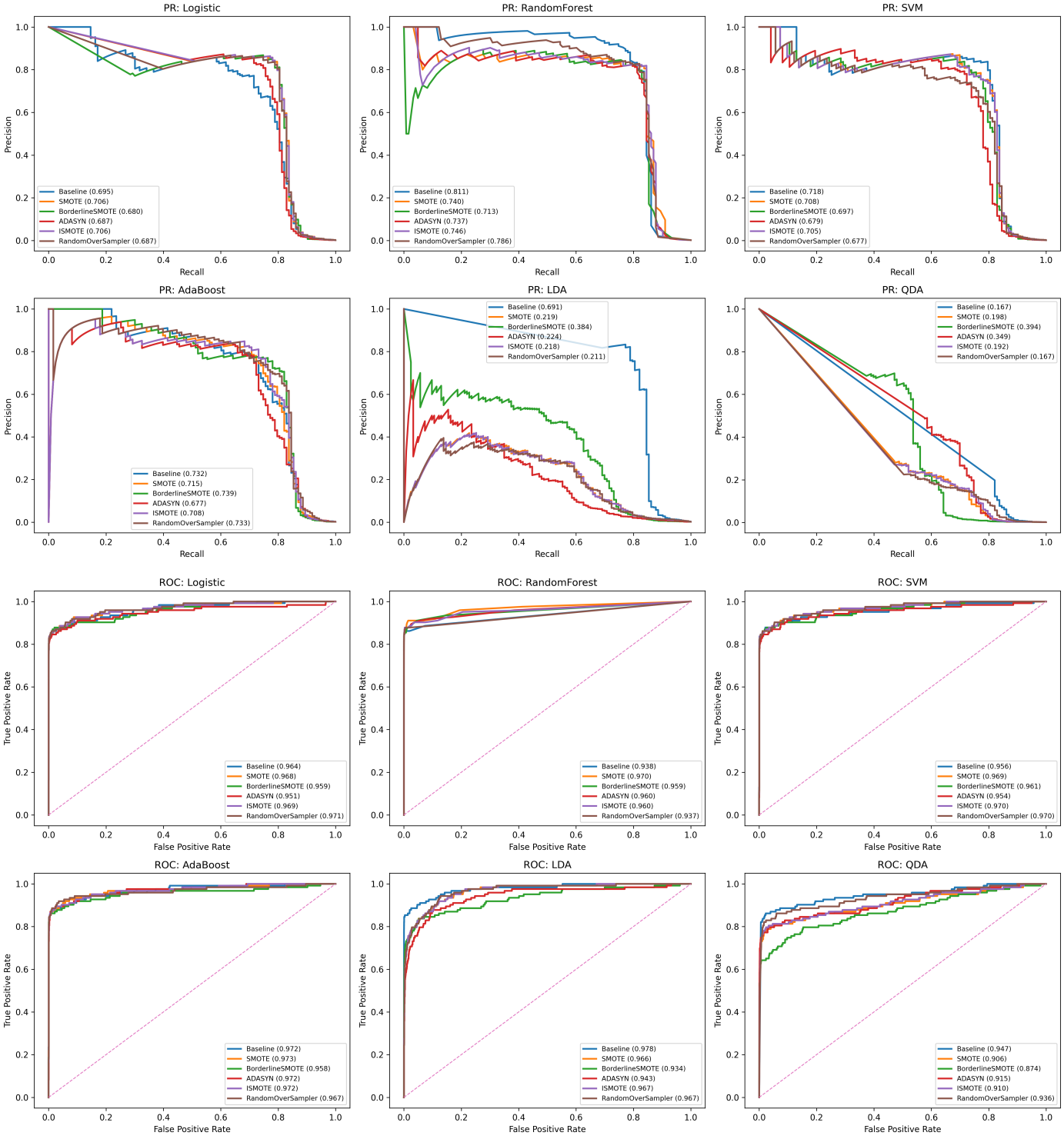

Figure 28: The PR_AUC and ROC_AUC curve when the number of Gaussian noise feature is 100.

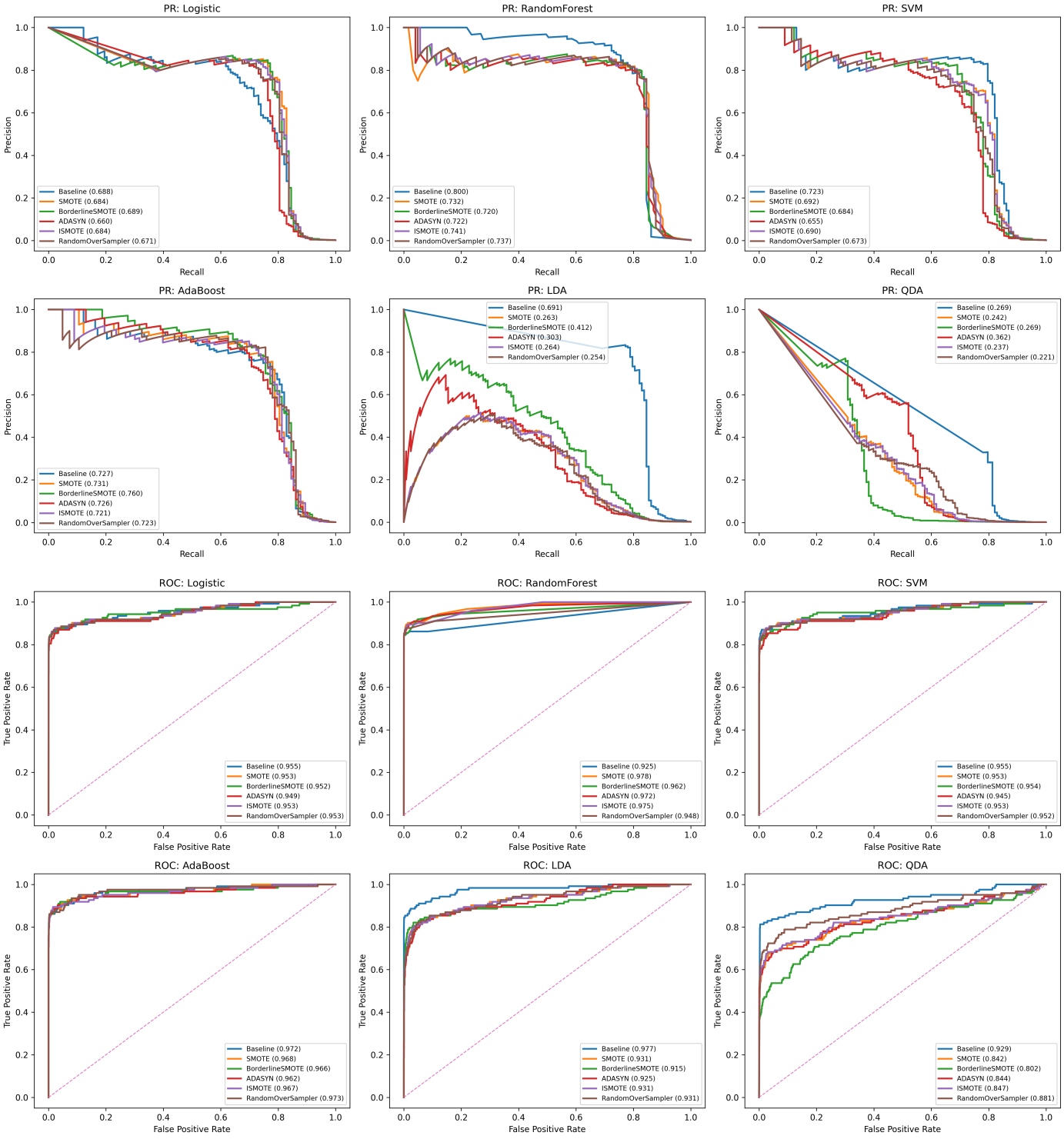

Figure 29: The PR_AUC and ROC_AUC curve when the number of Gaussian noise feature is 200.

Next, we examine the performance of classification using neural networks. The neural network has 2 hidden layers, with 64 and 32 neurons each, with ReLU activation function. We record the performance of different oversampling techniques against noise level as follow.


Recall vs NoiseFeatures for NeuralNet
Recall
Number of Gaussian noise features
ROC_AUC vs NoiseFeatures for NeuralNet
ROC_AUC
Number of Gaussian noise features
PR_AUC vs NoiseFeatures for NeuralNet
PR_AUC
Number of Gaussian noise features
Resampler
ADASYN
Baseline
BorderlineSMOTE
ISMOTE
RandomOverSampler
SMOTE

Figure 30

From Figure 30, we see that as noise level increases, various oversampling techniques behave poorly compared to baseline. Our guesses include the following:

- The neural network has over-fitted the training set when a huge amount of noise features are present due to its complex structure, so when applying the over-fitted network to our testing set, the performance becomes low.
- The resampling techniques generates redundant data points that may also lead to over-fitting.

Part IV
Acknowledgments

The original research was supervised by Professor Abbas Khalili at the department of Mathematics and Statistics at McGill University. This article summaries my independent research during the Winter 2026 semester. I would like to thank Professor Khalili for his professional guidance, without whose support this article would not have been possible.

Figure 31: Thank you!

Part V
References

7 Bibliography

[BB24] Christopher Bishop and Hugh Bishop. Deep learning: Foundations and concepts. 2024.

[BDL08] Gerard Biau, Luc Devroye, and Gabor Lugosi. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, 9:2015–2033, 2008.

[BFOS84] L. Breiman, J. Friedman, R.A. Olshen, and C.J. Stone. Classification and regression trees. 1984.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[Che20] Guangliang Chen. Linear discriminant analysis. *Math 253: Mathematical Methods for Data Visualization*, San José State University, 2020.

[Cra02] J.S Cramer. The origins of logistic regression. 2002.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoid function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55, 1997.

[FZT21] Yang Feng, Min Zhou, and Xin Tong. Imbalanced classification: A paradigm-based review. 2021.

[GT17] Paola Galdi and Roberto Tagliaferri. Data mining: Accuracy and error measures for classification and prediction. *University of Salerno*, 13, 2017.

[HBGL08] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. pages 1322–1328, 2008.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning, data mining, inference, and prediction. 2009.

[HWM05] H. Han, WY. Wang, and BH Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. *Advances in Intelligent Computing*, 3644:878–887, June 2005.

[JWHT21] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning, with applications in R. 2021.

[Kas80] G.V. Kass. An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society.*, 29:119–127, 1980.

[KBGB21] Pradeep Kumar, Roheet Bhatnagar, Kuntal Gaur, and Anurag Bhatnagar. Classification of imbalanced data: Review of methods and applications. 2021.

[Mak00] Ginny Mak. The implementation of support vector machines using the sequential minimal optimization algorithm. 2000.

[MJ24] Ibomoiye Domor Mienye and Nobert Jere. A survey of decision trees: Concepts, algorithms, and applications. *IEEE Access*, 12:86716–86727, 2024.

[MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. 2012.

[Pla98] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. (MSR-TR-98-14), April 1998.

[Qui85] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1985.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, June 1986.

[Rud17] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2017.

[SBV15] Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. Consistency of random forests. *The Annals of Statistics*, 43(4), August 2015.

[Sha15] Cosma Rohilla Shalizi. Classification and regression trees, chapter 13. 2015.

[Ste26] Russell Steele. Course notes for math 523: Generalized linear models. 2026.

[TFL18] Xin Tong, Yang Feng, and Jingyi Jessica Li. Neyman-pearson classification algorithms and np receiver operating characteristics. 2018.

[YYS+25] Li Ying, Yali Yang, Peihua Song, Lian Duan, and Rui Ren. An improved smote algorithm for enhanced imbalanced data classification by expanding sample generation space. *Scientific Reports*, 2025.