Machine Learning

Shuhei Watanabe

April 4, 2021

1 Introduction

The ultimate goal of machine learning (ML) is to automate the whole process of the learning and it is called **end-to-end learning**. The end-to-end learning is composed of the followings:

1. **Preprocessing**: Standardization, padding of missing values, removing outliers, checking imbalanced target
2. **Feature extraction and encoding**: feature engineering, i.e. to make new features from given data such as BMI from height and weight, and encoding such as one-hot encoding
3. **Feature selection**: To choose some variables and remove other variables. It is effective when there seems to be unrelated variables or the dimension is too high.
4. **Evaluation and model selection**: To select suitable algorithms, tune hyperparameter and generalize the model not only for past data but also future instances.
5. **Postprocessing**: To ensure fairness and remove the dicriminational prediction (e.g. class imbalance, cancer test)

Note that current deep learning algorithms are often called end-to-end learning. We use the following notations throughout the notebook.

Definition 1

- **weight vector** $\boldsymbol{w} \in \mathbb{R}^K$*: the internal parameters learned through algorithms*
- **feature vector** $X \in \mathbb{R}^{N \times D}$*: the given input data*
- **labels** $Y \in \mathbb{R}^N$*: the corresponding answer of the given data*
- **Objective function** $f : \mathbb{R}^D \to \mathbb{R}$*: The function, which algorithms learn. In this example, it takes an input and returns the corresponding prediction*
- **Loss function** $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$*: The function, which measures the loss of the current model state*

2 Probabilistic Interpretation

In this section, we introduce the probabilistic interpretation of regression models.

2.1 Generative Model

The regression models assume the existence of $f(\boldsymbol{x})$ and predict the function in the form of $f(\boldsymbol{x}; \boldsymbol{w})$. Probabilistic models assume that such functions $f(\boldsymbol{x})$ follows normal distribution at each given point.

Assumption 1
At given point $\boldsymbol{x}$, the function follows the normal distribution, i.e.

$$f(\boldsymbol{x}) \sim \mathcal{N}(f(\boldsymbol{x}; \boldsymbol{w}), \sigma^2)$$

2.2 Maximum Likelihood Estimation

When we would like to infer the parametric model $P(y|\boldsymbol{x}, \boldsymbol{w})$ of a given task [1]. Under this condition, the likelihood is defined as follows:

$$L(\boldsymbol{w}|\mathcal{D}) = \prod_{i=1}^{N} P(y_i|\boldsymbol{x}_i, \boldsymbol{w}) \tag{1}$$

where $\mathcal{D}$ is the given data to train. As seen in Eq (1), the likelihood is defined as the product of the probabilities that each data can happen with the given parametric model. Therefore, we can obtain the model, which well approximates the training data by maximizing the likelihood given in Eq (1). Conventionally, the parametric model $P(y_i|\boldsymbol{x}_i, \boldsymbol{w})$ belongs to the exponential family. For this reason, we take the logarithm of likelihood $\log L(\boldsymbol{w}|\mathcal{D}) = \sum_{i=1}^{N} \log P(y_i|\boldsymbol{x}_i, \boldsymbol{w})$ when we optimize the likelihood. The derivative of the log likelihood is the following:

$$\frac{\partial \log L(\boldsymbol{w}|\mathcal{D})}{\partial \boldsymbol{w}} = \sum_{i=1}^{N} \frac{\partial Q(\boldsymbol{x}_i, y_i, \boldsymbol{w})}{\partial \boldsymbol{w}} \tag{2}$$

where $\log P(\boldsymbol{x}_i, y_i, \boldsymbol{w}) = \log e^{Q(\boldsymbol{x}_i, y_i, \boldsymbol{w})} = Q(\boldsymbol{x}_i, y_i, \boldsymbol{w})$. We can achieve the maximization of log likelihood when the derivative is zero.

3 Simple non-parametric Methods

Roughly speaking, Non-parametric models refer to models that have a fixed number of parameters. There can be another definition, but we use the definition in this notebook.

3.1 K-nearest neighbors method

This method checks the K-nearest neighbors and pick the label which happens the most in the neighbors. The advantage of this method is **the simplicity**. On the other hand, since this method decides the class label based on the neighbors, if **covariate-shift** happens between training and test dataset, it cannot handle the tasks well. Furthermore, **the parameter selection of** K can seriously affect the performance. Another disadvantage of KNN is **the time complexity** $O(DN)$ for each test case where N is the number of training samples and D is the dimension of each sample. The imbalanced data can also damage the performance.

[1] The naïve examples are binomial distribution, e.g. the probability that you will get a head of a coin by a coin toss and normal distribution, which we have to infer the mean vector and covariance matrix.

Algorithm 1 K-nearest neighbors method

$\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}, \mathcal{D}_{\text{test}} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M\}$
p, K ▷ the parameter of p-norm and the number of neighbors

```
function KNN(D_train, x_test, K)
    data = []
    for i = 0, 1, ..., N do
        d = distance(p, x_i, x_test)        ▷ Define the p-norm
        data.append([d, y_i])
    data.sort(axis=0)        ▷ Sort the data with respect to distance from x_test
    return the most frequent class in data[:K]
```

3.2 Naïve Bayes

Assumption 2
Suppose all the input variables $x_1, x_2, ..., x_D$ are conditionally independent i.e. $P(x_i|x_j) = P(x_i)$ for all i, j.

Under this assumption, Naïve Bayes can calculate the arbitrary conditional probability as follows:

$$P(y = C|x_1, x_2, ..., x_D) = \frac{P(y = C)}{P(x_1, x_2, ..., x_D)} \prod_{d=1}^{D} P(x_d|y = C) \tag{3}$$

where C is the choice of class and $P(x_1, x_2, ..., x_n) = \sum_{C=1}^{N_c} P(y = C) \prod_{d=1}^{D} P(x_d|y = C)$. The benefits of Naïve Bayes are the followings:

- Simple and easy to code
- Computationally cheap and feasible even when the data size is large
- Not affected seriously by trivial features ($\because$ Assumption 2)

On the other hand, the Assumption 2 does not hold in most real-world problems and this is the major barrier of this method.

4 Linear models

4.1 Linear Discriminant Analysis (LDA)

LDA is a linear classification method and this algorithm assumes the followings:

Assumption 3
Given data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^D, y_i \in \{1, 2, \cdots, K\}$,

1. *$\forall i, p(x|y = i) \sim \mathcal{N}(\boldsymbol{m}_i, \boldsymbol{\Sigma}_i)$*
2. *The covariance matrix of each distribution is identical, i.e., $\forall i, j, \boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$*

where $\boldsymbol{m}_i = \sum_{j:y_j \in \mathcal{C}_i} \boldsymbol{x}_j / n_i$, $\boldsymbol{\Sigma}_i = \sum_{j:y_j \in \mathcal{C}_i} (\boldsymbol{x}_j - \boldsymbol{m}_i)(\boldsymbol{x}_j - \boldsymbol{m}_i)^\top / (n_i - 1)$, n_i is the number of samples which belong to the i-th class $\mathcal{C}_i$. Using these notations, we define the followings:

Definition 2
Total within-class covariance matrix

$$\boldsymbol{\Sigma}_W = \sum_{i=1}^{K} \boldsymbol{\Sigma}_i \tag{4}$$

Between-class covariance matrix (the weighted sum of the case of the i-th class and the others)

$$\boldsymbol{\Sigma}_B = \sum_{i=1}^{K} n_i (\boldsymbol{m}_i - \boldsymbol{m}_{-i})(\boldsymbol{m}_i - \boldsymbol{m}_{-i})^\top \tag{5}$$

where $\boldsymbol{m}_{-i}$ is the mean vector of all the classes except the i-th class.

When considering the decision boundary of two gaussian distributions with an identical covariance matrix, the line clearly passes the centeral point between $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$. However, the slope is unknown; therefore, we discuss the derivation of the slope. When the mapping of covariance matrix to a hyperplane [2] perpendicular to the decision boundary, the overlap of the mapped distributions should be minimized. The singed distance from the decision boundary $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ is computed as $f(\boldsymbol{x})/||\boldsymbol{w}||$ [3]. Therefore, we minimize the variance of distance from the decision boundary. When mapping the mean vector $\boldsymbol{m}$ and the covariance matrix $\boldsymbol{\Sigma}$, those two will be transformed as follows:

$$\begin{aligned}
\boldsymbol{m}_{i,\text{mapped}} &= \frac{1}{n_i} \sum_{y_j \in \mathcal{C}_i} \frac{f(\boldsymbol{x}_j)}{||\boldsymbol{w}||} = \frac{1}{n_i} \sum_{y_j \in \mathcal{C}_i} \frac{\boldsymbol{w}^\top \boldsymbol{x}_j + b}{||\boldsymbol{w}||} = \frac{\boldsymbol{w}^\top \boldsymbol{m}_i + b}{||\boldsymbol{w}||} \\
\boldsymbol{\Sigma}_{i,\text{mapped}} &= \frac{1}{n_i - 1} \sum_{y_j \in \mathcal{C}_i} (f(\boldsymbol{x}_j)/||\boldsymbol{w}|| - \boldsymbol{m}_{i,\text{mapped}})(f(\boldsymbol{x}_j)/||\boldsymbol{w}|| - \boldsymbol{m}_{i,\text{mapped}})^\top \\
&= \frac{1}{(n_i - 1)||\boldsymbol{w}||^2} \sum_{y_j \in \mathcal{C}_i} (\boldsymbol{w}^\top \boldsymbol{x}_j - \boldsymbol{w}^\top \boldsymbol{m}_i)(\boldsymbol{w}^\top \boldsymbol{x}_j - \boldsymbol{w}^\top \boldsymbol{m}_i)^\top \\
&= \frac{1}{||\boldsymbol{w}||^2} \boldsymbol{w}^\top \boldsymbol{\Sigma}_i \boldsymbol{w}
\end{aligned} \tag{6}$$

This maximization for the two-class tasks is performed using the following Fisher's criterion:

Definition 3
Fisher's criterion measures the separation between two distribution via the ratio of between-class variance and within-class variance, i.e.,

$$C = \frac{\sigma_B^2}{\sigma_W^2} \tag{7}$$

The maximization of this equation leads to larger separation.

Those two are computed as $\sigma_B^2 = n(\boldsymbol{m}_{i,\text{mapped}} - \boldsymbol{m}_{j,\text{mapped}})(\boldsymbol{m}_{i,\text{mapped}} - \boldsymbol{m}_{j,\text{mapped}})^\top$, $\sigma_W^2 = \sum_{i=1}^{K} \boldsymbol{\Sigma}_{i,\text{mapped}}$. Therefore, the criterion between class i and j is the following:

$$C = \frac{n(\boldsymbol{w} \cdot \boldsymbol{m}_i - \boldsymbol{w} \cdot \boldsymbol{m}_j)^2}{\boldsymbol{w}^\top \boldsymbol{\Sigma}_i \boldsymbol{w} + \boldsymbol{w}^\top \boldsymbol{\Sigma}_j \boldsymbol{w}} \propto \frac{||\boldsymbol{w}||^2 (\boldsymbol{m}_i - \boldsymbol{m}_j)^2}{\boldsymbol{w}^\top (\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j) \boldsymbol{w}} \tag{8}$$

[2] Since the decision boundary is always $D-1$ dimension, this hyperplane is also $D-1$ dimension.
[3] The mapping of $\boldsymbol{x}$ to $\boldsymbol{w}$ ($\boldsymbol{w}$ is perpendicular to the decision boundary, e.g. $\boldsymbol{w} \cdot (b/w_1, , -b/w_2, 0, \cdots, 0) = 0$) is $\boldsymbol{w}^\top \boldsymbol{x}/||\boldsymbol{w}||$ and the distance between the origin and the decision boundary is $||b||/||\boldsymbol{w}||$.

By diffrentiating the equation with respect to $\boldsymbol{w}$ and taking this derivative as zero, we obtain the following equation:

$$(\boldsymbol{w}^\top \boldsymbol{\Sigma}_B \boldsymbol{w})\boldsymbol{\Sigma}_W \boldsymbol{w} = (\boldsymbol{w}^\top \boldsymbol{\Sigma}_W \boldsymbol{w})\boldsymbol{\Sigma}_B \boldsymbol{w} \tag{9}$$

Since both $\boldsymbol{w}^\top \boldsymbol{\Sigma}_B \boldsymbol{w}$ and $\boldsymbol{w}^\top \boldsymbol{\Sigma}_W \boldsymbol{w}$ are the scalar factor, we can drop them and obtain the following equation:

$$\begin{aligned} \boldsymbol{\Sigma}_W \boldsymbol{w} &\propto \boldsymbol{\Sigma}_B \boldsymbol{w} \\ \boldsymbol{w} &\propto \boldsymbol{\Sigma}_W^{-1} \boldsymbol{\Sigma}_B \boldsymbol{w} \propto \boldsymbol{\Sigma}_W^{-1} (\boldsymbol{m}_i - \boldsymbol{m}_j) \\ \boldsymbol{w} &= \boldsymbol{\Sigma}_W^{-1} (\boldsymbol{m}_i - \boldsymbol{m}_j) \end{aligned} \tag{10}$$

where $(\boldsymbol{m}_i - \boldsymbol{m}_j)^\top \boldsymbol{w}$ is also a scalar factor. The bias term b can be introduced from the fact that the decision boundary must pass the center point of two mean vector. Therefore, b follows this equation:

$$f\left(\frac{\boldsymbol{m}_i + \boldsymbol{m}_j}{2}\right) = 0 \iff b = -\frac{\boldsymbol{w}^\top (\boldsymbol{m}_i + \boldsymbol{m}_j)}{2} \tag{11}$$

The advantages of LDA are the closed-form solution and non-parametetric. Additionally, it can be used for multiple-class separation as follows:

1. **one-against-rest**
   It requires K discriminators. There can be large regions where the class is ambiguous.
2. **one-against-one**
   It requires $K(K-1)/2$ discriminators. This algorithm can also have the class-undefined region, but better than one-against-rest.
3. **inherent multi-class formulation**
   It requires K discriminators. First, we prepare $f_i(\boldsymbol{x}) = \boldsymbol{w}_i \boldsymbol{x} + b_i$ for all i. A given point $\boldsymbol{x}$ belongs to $\mathcal{C}_i$ if and only if it satisfies $f_i(\boldsymbol{x}) > f_j(\boldsymbol{x})$ for all j. This algorithm does not create any ambiguous regions.

On the other hand, the major issues are that the assumption of Gaussian distribution for each class does not hold in real-world problems and it cannot separate non-linearly. Note that when we use non-linear mapping of the features, we can separate non-linearly. Furthermore, the total complexity is $O(nD^2 + D^3)$ [4], so it can be computationally expensive for high dimensional problems.

4.2 Logistic Regression

Logistic regression is a supervised learning for classification tasks. This method returns the probability that a given input vector belongs to class $\mathcal{C}_1$. This probabilistical post-processing is computed as $h(\boldsymbol{x}; \boldsymbol{w}) = 1/(1 + e^{-\boldsymbol{x}^\top \boldsymbol{w}})$ where $\boldsymbol{x}$ is an augmented vector again and is known as **logistic calibration**. This allows us to yield $[0, 1]$ bounded probability-like real numbers. Plus, **isotonic calibration** is also known as a post-processing method. This method takes the percentil of samples and map to given distribution according to the percentile information. From the Bayesian perspective, it is formulated as follows:

$$\begin{aligned} p(\mathcal{C}_1|\boldsymbol{x}) &= \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \\ &= \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)(1 + e^{-a})} \\ &= \frac{1}{1 + e^{-a}} = \sigma(a) \end{aligned} \tag{12}$$

where we set $a = \boldsymbol{w}^\top \boldsymbol{x} + b = \log \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$ and it is called **log-odds**. We approximate the optimal log-odds in the formulation.

[4] The computation of the variance and the inverse matrix are dominant.

The major drawback of this method is that it does **not have the analytical solution**. This is because this method uses **non-linear transformation** to obtain the probability. However, the loss function is differentiable, so it is possible to optimize the weight vector $\boldsymbol{w}$ using gradient descent.

4.2.1 Gradient descent

Let f be the function $f : \mathbb{R}^D \to \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^D$ be the input vector, x_i be the i-th element of vector $\boldsymbol{x}$, the objective be the minimization of f. The gradient of the function $f(\boldsymbol{x})$ is defined as follows:

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} \tag{13}$$

Since gradient tells us if the function value will increase or decrease when we perturb the function, the update of the gradient descent is performed as follows:

$$x_i \leftarrow x_i - \alpha \frac{\partial f(\boldsymbol{x})}{\partial x_i} \tag{14}$$

where α is the learning rate. Note that when a given function is **continually differentiable** and **convex** [5], it is guaranteed to converge. The **cross entropy function** satisfies those two properties.

4.2.2 The gradient of cross entropy in Logistic Regression

The cross entropy function is $J(\boldsymbol{w}) = -\sum_{i=1}^{n} \Big(y_i \log\, \sigma(\boldsymbol{x}_i; \boldsymbol{w}) + (1 - y_i) \log(1 - \sigma(\boldsymbol{x}_i; \boldsymbol{w})) \Big)$ and the derivative of sigmoid function is $\partial \sigma(\boldsymbol{x}; \boldsymbol{w}) / \partial \boldsymbol{w} = x_i \sigma(\boldsymbol{x}; \boldsymbol{w})(1 - \sigma(\boldsymbol{x}; \boldsymbol{w}))$. Therefore, the gradient of the cross entropy is the following:

$$\frac{\partial J(\boldsymbol{w})}{\partial w_i} = -\sum_{j=1}^{n} (x_{j,i}(y_j - \sigma(\boldsymbol{x}_j; \boldsymbol{w})) \tag{15}$$

where each term can be derived as follows:

$$\begin{aligned} \frac{\partial}{\partial w_i}(1 - y_j)\log(1 - \sigma(\boldsymbol{x}_j; \boldsymbol{w})) &= \frac{-x_{j,i}(1 - y_j)}{1 - \sigma(\boldsymbol{x}_j; \boldsymbol{w})} \sigma(\boldsymbol{x}_j; \boldsymbol{w})(1 - \sigma(\boldsymbol{x}_j; \boldsymbol{w})) = -x_{j,i}(1 - y_j)\sigma(\boldsymbol{x}_j; \boldsymbol{w}) \\ \frac{\partial}{\partial w_i} y_n \log\, \sigma(\boldsymbol{x}_n; \boldsymbol{w}) &= \frac{x_{j,i} y_j}{\sigma(\boldsymbol{x}_j; \boldsymbol{w})} \sigma(\boldsymbol{x}_j; \boldsymbol{w})(1 - \sigma(\boldsymbol{x}_j; \boldsymbol{w})) = x_{j,i} y_n (1 - \sigma(\boldsymbol{x}_j; \boldsymbol{w})) \end{aligned} \tag{16}$$

Therefore, the update of the gradient descent can be computed as:

$$w_i \leftarrow w_i - \alpha \sum_{j=1}^{n} (x_{j,i}(\sigma(\boldsymbol{x}_j; \boldsymbol{w}) - y_j)) \tag{17}$$

Since the termination criterion is not given, the total complexity is unknown. However, the weight update is $O(nD)$ and the query is $O(D)$.

4.3 Linear Regression

Linear Regression is a supervised learning to model $f : \mathbb{R}^D \to \mathbb{R}$. and described as $f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + \sum_{d=1}^{D} w_d x_d$ or $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{x}^\top \boldsymbol{w}$ [6]. The assumptions on linear regression model are the followings:

[5] $f(\lambda \boldsymbol{x}_1 + (1 - \lambda)\boldsymbol{x}_2) \leq \lambda f(\boldsymbol{x}_1) + (1 - \lambda) f(\boldsymbol{x}_2)$ for $\lambda \in [0, 1]$. Hessian is quite famous related to convex.

[6] This $\boldsymbol{x} \in \mathbb{R}^{D+1}$ is an augumented vector and includes 1 at the head of the vector.

Assumption 4
Linear Regression performs well if a given dataset satisfies the following assumptions.

1. *The expected value of the residual errors is zero ($\mathbb{E}(\epsilon_i) = 0$)*
2. *The residual errors are independent and each variance is identical ($\mathbb{V}(\epsilon_i) = \sigma^2$)*
3. *The residual errors follow a normal distribution, i.e., $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$)*

In other words, $\epsilon_1, \cdots, \epsilon_n$ is i.i.d (Independent and identically distributed random variables) and **The heteroscedasctic data** (σ^2 is not constant over all the data) and **outliers** potentially damage the accuracy of the model. If data include outliers, it is better to use **L1 loss** instead of L2 loss.

4.3.1 The derivation of the analytical form of Linear Regression

In general, the loss function of linear regression is the square least error function as follows:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} ||y_i - f(\boldsymbol{x}_i; \boldsymbol{w})||^2 \tag{18}$$

By minimizing this function, we obtain the optimal $\boldsymbol{w}^\star$. First, we transform Eq (18) as follows:

$$\begin{aligned} J(\boldsymbol{w}) &= \sum_{i=1}^{n} ||y_i - \boldsymbol{x}_i^\top \boldsymbol{w}||^2 \\ J(\boldsymbol{w}) &= ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) \end{aligned} \tag{19}$$

where $\boldsymbol{X} \in \mathbb{R}^{n \times (D+1)}$ is the given feature vectors and $\boldsymbol{y} \in \mathbb{R}^n$ is the corresponding value vector for each feature vectors. Next, we will take the derivative with respect to the weight vector.

$$\begin{aligned} \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} &= \frac{\partial}{\partial \boldsymbol{w}} \left(\boldsymbol{y}^\top \boldsymbol{y} - (\boldsymbol{X}\boldsymbol{w})^\top \boldsymbol{y} - \boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{w} + (\boldsymbol{X}\boldsymbol{w})^\top (\boldsymbol{X}\boldsymbol{w}) \right) \\ &= \frac{\partial}{\partial \boldsymbol{w}} \left(-2\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^\top (\boldsymbol{X}^\top \boldsymbol{X}) \boldsymbol{w} \right) \left(\because \boldsymbol{a}^\top \cdot \boldsymbol{b} = \boldsymbol{b}^\top \cdot \boldsymbol{a} \right) \end{aligned} \tag{20}$$

The optimal $\boldsymbol{w}^\star$ is obtained when the derivative is equal to 0. Therefore, the following equation can be derived using $\partial(\boldsymbol{w}^\top A \boldsymbol{w})/\partial \boldsymbol{w} = \boldsymbol{w}^\top (A + A^\top)$:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{w}} \left(-2\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{w} + \boldsymbol{w}^\top (\boldsymbol{X}^\top \boldsymbol{X}) \boldsymbol{w} \right) &= \boldsymbol{0} \\ -2\boldsymbol{y}^\top \boldsymbol{X} + \boldsymbol{w}^\top (\boldsymbol{X}^\top \boldsymbol{X} + (\boldsymbol{X}^\top \boldsymbol{X})^\top) &= \boldsymbol{0} \\ \boldsymbol{w} &= (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \end{aligned} \tag{21}$$

Table 1: The comparison of LDA and Logistic Regression

-	Logistic Regression	LDA
Estimation	gradient descent	maximum log likelihood (closed form)
Likelihood	the output	requires extra processing
Data Distribution	no assumption	desirably gaussian and same covariance
Outliers	not so sensitive	sensitive
Variables	continuous, categorical	only continuous

where **the determinant** of $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ must be **non-zero**. If it is zero, we use gradient descent and the convergence is guaranteed due to the convex objective. This condition of non-zero determinant are met if and only if $D+1 \leq n$ and $\mathrm{Rank}(X) = D+1$ [7]. The total complexity of the parameter inference is $O(nD^2 + D^3)$, the multiplication and the computation of the inverse of $\boldsymbol{X}^\top \boldsymbol{X}$ and the approximation for a new point is $O(D)$. Since the weight vector $\boldsymbol{w}$ is the partial derivative of f, the absolute value of w_i represents **the importance of the dimension** x_i. Note that we can use analytic form $\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda I)^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ in the case of L2 loss, but there is no closed form for L1 loss.

5 Overfitting and underfitting

The goal of ML models is to find out features behind given data and generalize such features. However, ML models often yield data-specific features or **noise** on training data and this leads to less generalization, i.e. **overfitting**. On the other hand, the scarcity of data or the lack of the flexibility of learning models leads to poor performance, i.e. **underfitting**. Therefore, we have to pay much attention not only to the training accuracy, but also the discrepancy between the training and the test performance called **generalization gap**.

5.1 Bias-variance trade-off

Bias is the gap between the distribution of the prediction and the true value and **variance** is the deviation of prediction on different data distribution. Those two are in the trade-off relationship and we need to balance them. For example, when the bias of models is large, the models do not predict the objective function well on given data. On the other hand, when the bias of models is small, the models predicts well. For variance, if the variance is small, prediction is similar to each other regardless of given data distribution. On the other hand, if the variance is large, ML models overfit specific features or noise in given data. For this reason, the models predict well on given training data, but potentially not on other data distribution. Those terms are appeared in the cost function. The **expected test error** is represented as follows:

$$\mathbb{E}_{(\boldsymbol{X},\boldsymbol{Y})\sim\mathcal{D},\mathcal{D}\sim\mathcal{P}^n}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \boldsymbol{Y})^2] = \int_{\boldsymbol{x}}\int_{y}\int_{\mathcal{D}} (\hat{f}(\boldsymbol{x};\mathcal{D}) - y)^2 p(\boldsymbol{x},y)p(\mathcal{D}) d\boldsymbol{x} dy d\mathcal{D} \tag{22}$$

where $\mathcal{D}$ is a dataset sampled from data distribution $\mathcal{P}^n$ and $|\mathcal{D}| = n$. For the simplicity, we write the left hand side as $\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \boldsymbol{Y})^2]$. By transforming Eq (22), we will obtain the following equations:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \boldsymbol{Y})^2] &= \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}) + \bar{f}(\boldsymbol{X}) - \boldsymbol{Y})^2] \\
&= \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))^2 + (\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})^2 \\
&\quad + 2(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})] \\
&= \mathbb{E}_{\boldsymbol{X},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))^2] + \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})^2] \\
&\quad \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[2(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})]
\end{aligned} \tag{23}$$

where $\bar{f}(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}[\hat{f}(\boldsymbol{x};\mathcal{D})]$. The last line $\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[2(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})]$ can be canceled out by transforming as follows:

$$\begin{aligned}
2\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})] &= 2\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})]] \\
&= 2\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})\mathbb{E}_{\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))]] \\
&= 2\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})(\bar{f}(\boldsymbol{X}) - \bar{f}(\boldsymbol{X}))] \\
&= 0
\end{aligned} \tag{24}$$

[7] If $n < D+1$, the rank of $\boldsymbol{X}^\top \boldsymbol{X}$ is less than $D+1$.

Table 2: Bias-variance trade-off

-	bias	variance
High	underfitting	overfitting
Low	well-fitting	generalization

Furthermore, we transform $\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})^2]$ and yield the followings:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \boldsymbol{Y})^2] &= \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \bar{\boldsymbol{Y}}(\boldsymbol{X}) + \bar{\boldsymbol{Y}}(\boldsymbol{X}) - \boldsymbol{Y})^2] \\ &= \mathbb{E}_{\boldsymbol{X}}[(\bar{f}(\boldsymbol{X}) - \bar{\boldsymbol{Y}}(\boldsymbol{X}))^2] + \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{\boldsymbol{Y}}(\boldsymbol{X}) - \boldsymbol{Y})^2] \\ &\quad + 2\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \bar{\boldsymbol{Y}}(\boldsymbol{X}))(\bar{\boldsymbol{Y}}(\boldsymbol{X}) - \boldsymbol{Y})]\end{aligned} \tag{25}$$

where $\bar{y}(\boldsymbol{x}) = \int y\ p(y|\boldsymbol{x})dy$. The last line $\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \bar{\boldsymbol{Y}}(\boldsymbol{X}))(\bar{\boldsymbol{Y}}(\boldsymbol{X}) - \boldsymbol{Y})]$ can be canceled out by transforming as follows:

$$\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{f}(\boldsymbol{X}) - \bar{\boldsymbol{Y}}(\boldsymbol{X}))(\bar{\boldsymbol{Y}}(\boldsymbol{X}) - \boldsymbol{Y})] = \mathbb{E}_{\boldsymbol{X}}[(\bar{f}(\boldsymbol{X}) - \bar{\boldsymbol{Y}}(\boldsymbol{X}))\mathbb{E}_{\boldsymbol{Y}|\boldsymbol{X}}[\bar{\boldsymbol{Y}}(\boldsymbol{X}) - \boldsymbol{Y}]] = 0 \tag{26}$$

The last transformation uses the definition of $\bar{y}$. Therefore, as a whole, we obtain the following equation:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \boldsymbol{Y})^2] &= \underbrace{\mathbb{E}_{\boldsymbol{X},\mathcal{D}}[(\hat{f}(\boldsymbol{X};\mathcal{D}) - \bar{f}(\boldsymbol{X}))^2]}_{\text{Variance}} \\ &\quad + \underbrace{\mathbb{E}_{\boldsymbol{X}}[(\bar{f}(\boldsymbol{X}) - \bar{\boldsymbol{Y}}(\boldsymbol{X}))^2]}_{\text{Bias}} \\ &\quad + \underbrace{\mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}[(\bar{\boldsymbol{Y}}(\boldsymbol{X}) - \boldsymbol{Y})^2]}_{\text{Noise}}\end{aligned} \tag{27}$$

The properties of each state are listed in Table 2. Note that it is effective to add informative features especially for tabular data or increase the model capacity to deal with underfitting. The informative features can be created via feature engineering or domain knowledge. However, it is less effective when we have huge dataset or use end-to-end learning models.

5.2 Regularization

To balance the bias-variance trade-off, we often introduce so-called **a regularization term** to the loss function to prevent models from overfitting as follows:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\Omega(\boldsymbol{w}) \tag{28}$$

where $\Omega(\boldsymbol{w})$ is called a regularization term and λ is the coefficient to control the size of each weight value. Note that we do not include bias terms in the regularization. When the weight values grow, this is punished by the regularization term. The following two are often used as a regularization term:

1. **L1 regularization** $||\boldsymbol{w}||$: promotes **sparsity** and most trivial weight values go to zero
2. **L2 regularization** $||\boldsymbol{w}||^2$: alleviates **colinearity**, because it treats each feature evenly

Since the absolute weight values can be considered as the magnitude of contribution of the corresponding feature, it implies the regularization suppresses the contribution from each feature. In the case of L2 regularization, the regularization puts penalty on the features that contribute much. Therefore, these features cannot have too much influence on the model.

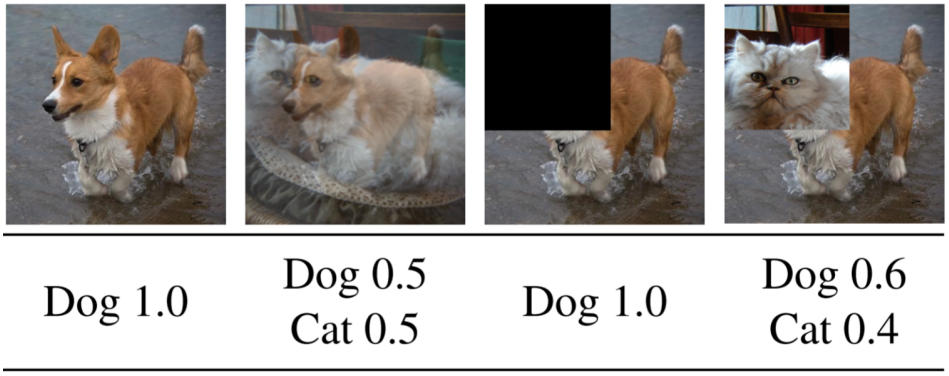

Figure 1: From the left side, the original, mixup, cutout, and cutmix image.

5.3 Miscellaneous regularization techniques

There are many methods that handle the overfitting and underfitting such as residual blocks, batch normalization, dropout, NAS, weight decay, data augmentation, early stopping and model averaging. Since we already discuss most methods in DL lecture, we do not discuss them. We focus on shake shake and data augmentation.

First, shake shake is the method to take linear interpolation of two parallel residual blocks. The coefficient for the interpolation is different for both forward and backward pass, so it achieves better generalization.

Second, the data augmentation is a great way to increase the data amount. The visualization of each data augmentation method is shown in Fig 1. Mixup is to take the linear intepolation of two images and the corresponding labels. Cutout is to drop randomly selected $k \times k$ pixels for image and k columns for tabular data. Cut mix is to replace randomly selected $k \times k$ pixels with another image. Since mixup and cutmix take non-deterministic labels, it can genelize more. Additionally, there is a technique called **trainable data augmentation**. This method learns RNN that suggests the data augmentation methods minimizing the validation loss and the RNN is updated using reinforcement learning on the validation loss. The optimization of the selection of such regularization methods is called Combined Algorithm Selection and Hyperparameter (CASH) optimization problem.

5.4 VC-dimension

In supervised learning, the choice of **function class** for decision boundaries or regression model is strongly related to the performance. Basically, the function with high capacity are likely to overfit the training dataset, so the following **VC-dimension** has been proposed to estimate the capacity of a function class:

Definition 4
VC-dimension is defined as the largest number D, such that there exists a set of D data points which the function class can shatter.

Roughly speaking, if there are more than $D+1$ data points, we can create problems which cannot be solved by a given function. For example, if there are four points in a 2D plane, linear functions sometimes cannot separate four points perfectly, e.g. XOR by logistic regression. However, linear functions can separate up to three points in a 2D plane with arbitrary configurations. Therefore, the VC-dimension for linear functions in a 2D space is $D = 3$. Additionally, the upper bound of the

potential test error is represented using the VC-dimension as follows:

$$\underbrace{R[f]}_{\text{Test Error}} \leq \underbrace{\hat{R}[f]}_{\text{Training Error}} + \underbrace{\sqrt{\frac{D(\log\frac{2n}{D}+1)-\log\frac{\delta}{4}}{n}}}_{\text{Variance}} \tag{29}$$

where $D \ll n$ and n is the number of data points in the training data. Roughly speaking, larger D leads to larger variance and lower bias. Therefore, we should avoid high capacity functions if we can achieve the same training error with lower capacity functions. Note that the concept of VC-dimension in **probably approximately correct (PAC)** framework inspired the SVMs.

6 Support Vector Machine (SVM)

In this section, we discuss hard-margin, soft-margin SVM with or without the kernel method. The hard-margin is the basic SVM, which assumes given data points can be perfectly separable and the soft-margin allows some violations. SVM is preferred over the preceding methods, because of no strong assumption about data distribution and **less damage by high dimensional inputs**. Additionally, soft margin allows more robust classification and support vectors can help to understand the problem better [8]. Non-linear decision boundaries can be drawn by using the kernel method. While soft-margin kernel SVM achieves great performance, kernel computation can be expensive with larger dataset and kernel SVM is hard to understand compared to the linear method. As a common property, SVM works poorly on imbalanced data in general[9].

6.1 Margin maximization

The basic formulation of SVM is to classify a given dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \{-1, 1\}$ by maximizing the margin between two classes as follows:

$$\text{argmax}_{\boldsymbol{w},b}\text{min}\{||\boldsymbol{x}-\boldsymbol{x}_i|| \mid y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 \text{ for all } i = 1, \cdots, n\} \tag{30}$$

Since the singed distance between $\boldsymbol{x}_i$ and $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ is computed as $\frac{\boldsymbol{w}^\top \boldsymbol{x}_i}{||\boldsymbol{w}||}$ and the minimum margin satisfies $\boldsymbol{w}^\top \boldsymbol{x}^{(1)} + b = 1$ and $\boldsymbol{w}^\top \boldsymbol{x}^{(-1)} + b = -1$ for each class. The margin between the decision boundary and class 1 or -1 are $\min\frac{\boldsymbol{w}^\top \boldsymbol{x}_i}{||\boldsymbol{w}||} = \frac{1-b}{||\boldsymbol{w}||}$ and $\max\frac{\boldsymbol{w}^\top \boldsymbol{x}_i}{||\boldsymbol{w}||} = \frac{-1-b}{||\boldsymbol{w}||}$ [10] respectively. Therefore, Eq (30) can be rewritten in the following form:

$$\text{max}_{\boldsymbol{w},b}\left(\frac{1-b}{||\boldsymbol{w}||} - \frac{-1-b}{||\boldsymbol{w}||}\right) = \text{max}_{\boldsymbol{w}}\frac{2}{||\boldsymbol{w}||} = \text{min}_{\boldsymbol{w}}\frac{||\boldsymbol{w}||^2}{2} \tag{31}$$

In summary, SVM solves the following optimization problem:

$$\text{min}_{\boldsymbol{w}}\frac{||\boldsymbol{w}||^2}{2} \text{ subject to } y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 \text{ for all } i = 1, \cdots, n \tag{32}$$

The constraints are called **inequality constraints**.

[8] As another usage, one-class SVM is known. It constructs the possible smallest hypersphere and if a given data point is outside of the hypersphere, it is an outlier.

[9] If we use class-weighted SVM, which penalizes the misclassification of the scarce class more, SVMs can handle with imbalanced datasets as well.

[10] This distance is negative, so the maximization of negative distance is equivalent to the minimization of absolute distance.

6.2 Dual representation

The optimization of the objective under given constraints can be solved using the **Lagrangian** as follows:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{||\boldsymbol{w}||^2}{2} - \sum_{i=1}^{n} \alpha_i \underbrace{(y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1)}_{\text{margin maximization}} \tag{33}$$

where $\forall i, \alpha_i \geq 0$ and α_i is called **dual variables**[11]. The KKT condition [12] for this equation are the followings:

$$\begin{aligned}
&(1).\ \textbf{Stationarity}: \ \frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i = 0, \ \frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \\
&(2).\ \textbf{Primal feasibility}: \ y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 \geq 0 \\
&(3).\ \textbf{Dual feasibility}: \ \alpha_i \geq 0 \\
&(4).\ \textbf{Complementary slackness}: \ \alpha_i(y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1) = 0
\end{aligned} \tag{34}$$

Note that the inequality of condition (2) holds for all the data points except the closest points and the condition (4) implies that if α_i is non-zero, $\boldsymbol{x}_i$ is one of the closest points. From this equation, the optimal weight vector is $\boldsymbol{w}^\star = \sum_{i=1} \alpha_i y_i \boldsymbol{x}_i$. By replacing $\boldsymbol{w}$ with the optimal $\boldsymbol{w}^\star$, the dual optimization problem can be yielded as follows:

$$\begin{aligned}
L^d(\boldsymbol{\alpha}) &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j - \sum_{i=1}^{n} \alpha_i \left(y_i \left(\sum_{j=1}^{n} \alpha_j y_j \boldsymbol{x}_j^\top \boldsymbol{x}_i + b \right) - 1 \right) \\
&= -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + \sum_{i=1}^{n} \alpha_i - b \underbrace{\sum_{i=1}^{n} \alpha_i y_i}_{=0} \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j
\end{aligned} \tag{35}$$

The relationship between the solutions for the primal and the dual problems is known as **weak duality** shown below:

Theorem 1
The optimal value of L^d is, by definition, the best lower bound on L^p that can be obtained from the Lagrange dual function. In particular, $\sup(L^d) \leq \inf(L^p)$ *holds even if the original problem is not convex.*

In general, the Lagrange primal function is $L^p(\boldsymbol{w}, b) = \sup_{\boldsymbol{\alpha} \geq 0} L(\boldsymbol{w}, b, \boldsymbol{\alpha})$ and the Lagrange dual function is $L^d(\boldsymbol{\alpha}) = \inf_{\boldsymbol{w}, b} L(\boldsymbol{w}, b, \boldsymbol{\alpha})$. To solve the mimization of L^p is called **primal problem** and the maximization of L^d is called **dual problem**. It is known that when the strong duality holds, the solutions for the primal problem and the dual problem are identical and the solution gives us the optimal objective [13] under given constraints. By definition, $L^d(\boldsymbol{\alpha}) \leq L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \leq L^p(\boldsymbol{w}, b)$ can be

[11] $\boldsymbol{w}, b$ are called primal variables.
[12] If the objective takes the optimal value, the KKT condition must be fulfilled.
[13] I will stress that the goal is to not minimize the Lagrange, but minimize the objective function. This can be achieved by optimizing the primal or the dual function.

easily derived. Therefore, the dual optimization problem [14] of the primal one is the following:

$$\begin{aligned} &\text{argmax}_{\boldsymbol{\alpha}}\left(\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j\right) \\ &\text{subject to } \alpha_i \geq 0 \text{ for all } i = 1, \cdots, n \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0 \end{aligned} \tag{36}$$

Note that this problem can be solved using numerical quadratic programming and the convergence to the global optima is guaranteed. Since SVM requires users to store training data points, it is called an **instance-based learning** and **the data points that have non-zero** α_i are called **support vectors** [15]. As most α_i goes to zero, we have to **store only a fraction of training data points**. Once the optimization problem is solved, the predicted label of a given data can be computed as follows:

$$\text{sign}(f(\boldsymbol{x})) = \sum_{i=1}^{n}\alpha_i y_i \boldsymbol{x}^\top \boldsymbol{x}_i + b \tag{37}$$

Suppose the set of support vectors is $\mathcal{S}$, then the derivation of b is trivially derived from the definition of the support vectors as follows:

$$\begin{aligned} 1 &= y_i\left(\sum_{j=1}^{n}\alpha_j y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j + b\right) \text{ (for all } i\text{s.t. } \boldsymbol{x}_i \in \mathcal{S}) \\ b &= \frac{1}{|\mathcal{S}|}\sum_{\boldsymbol{x}_i \in \mathcal{S}}\left(y_i - \sum_{\boldsymbol{x}_j \in \mathcal{S}}\alpha_j y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j\right) \end{aligned} \tag{38}$$

Note that we take the average over all the support vectors for the numerical statbility.

6.3 Soft margin SVM

Soft margin SVM does not require linear separability of the given dataset. The formulation of soft margin SVM is as follows:

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{||\boldsymbol{w}||^2}{2} + C\sum_{i=1}^{n}\xi_i \\ &\text{subject to } y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i \text{ for all } i = 1, \cdots, n \end{aligned} \tag{39}$$

where $\xi_i \geq 0$ alleviates the margin between data points and the decision boundary. $\xi_i = 0$ is same as the normal SVM. $0 < \xi_i \leq 1$ is the state where data points can be classified properly, but it is inside the margin. $\xi_i > 1$ will be misclassified. In other words, we have to **store the data points that satisfies** $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \leq 1$ **as support vectors**. Basically, we would like to take smaller ξ_i and this is reflected in the formulation (the minimization of $C\sum_{i=1}^{n}\xi_i$). Note that since the constraints $y_i f(\boldsymbol{x}_i) \geq 1 - \xi_i$ must be satisfied, we can also use the **hinge loss** $\max(0, 1 - y_i f(\boldsymbol{x}_i))$ instead of ξ_i. In the same way as the hard margin SVM, the dual problem of soft margin SVM is as follows:

$$\begin{aligned} &\text{argmax}_{\boldsymbol{\alpha}}\left(\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^\top \boldsymbol{x}_j\right) \\ &\text{subject to } 0 \leq \alpha_i \leq C \text{ for all } i = 1, \cdots, n \text{ and } \sum_{i=1}^{n}\alpha_i y_i = 0 \end{aligned} \tag{40}$$

[14] In the case of SVM, the strong duality holds.

[15] Models such as neural networks are called **model-based learning**.

Both primal and dual problems of soft-margin SVMs are convex optimizatin and the global solution is guaranteed. We often use SGD for the primal problem and coordinate search for the dual problem.

Additionally, ν-SVM is known as the reformulation of soft margin SVM. ν upper-bounds the percentage of training data points which can violate the hard margin, i.e. $\xi_i > 1$. This formulation does not make any difference in terms of the optimization result. However, users can treat the hyperparameter ν more intuitively. Note that $\nu = 0$ and $C = \infty$ correspond to hard margin SVM and $\nu = 1$ and $C = 0$ correspond to highly soft margin SVM. Since soft margin generalizes more, it is recommended to use soft margin even when we can apply hard margin SVM.

6.4 Kernel and Kernel Trick

The dot product is known as one of the similarity measures and the generalized concept of the dot product is **kernel function** that measures a similarity between two given points. The definition of kernel function is as follows:

Definition 5
kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a similarity measure of given points $\boldsymbol{x}, \boldsymbol{x}'$. This function must satisfy the following properties:

1. **Symmetric***: $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$*
2. **Semi-positive definite***: $\forall n \in \mathbb{N}_{\geq 1}, \forall \boldsymbol{a} \in \mathbb{R}^n, \sum_{i=1}^n \sum_{j=1}^n, a_i a_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$*

This kernel function is the transformation of the following equation:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{K} \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{x}') \tag{41}$$

where $\phi : \mathcal{X} \to \mathbb{R}$ is a mapping and K is the number of mapping in the dot product space. Since this is also an inner product, it can also measure the similarity. Additionally, we can introduce non-linear mapping of the feature in the dot product space and that is why the kernel function has more flexible representation and higher capacity. However, it is usually hard for users to **define suitable mapping for each problem** and it takes $O(Kd)$ **for the computation of the inner product**. For this reason, the similarity measure using kernel function so-called **kernel trick** has been invented and kernel trick allows us to avoid those two problems. Typical kernel functions are the followings:

1. **Polynomial kernel** (d): $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}')^d$
2. **Gaussian Radial Basis function (RBF) kernel** [16] (σ): $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\sigma^2}\right)$
3. **Sigmoid kernel**(a, b): $k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(a(\boldsymbol{x}^\top \boldsymbol{x}') + b)$

where the parameters after the kernel name show the hyperparameter of each kernel. Note that every linear algorithm, which can be expressed by dot product operations, can be reformulated using the kernel function.

6.5 Multiclass SVMs

Multiclass classifications are performed by either **one-vs-rest** or **one-vs-one** approach. As in LDA, one-vs-rest takes K SVMs where K is the number of classes. The major issues of one-vs-rest are overlapping regions and unbalanced class. Although the classifiers score $f_k(\boldsymbol{x})$ is often biased especially with independent K learnings, overlapping regions can be solved by taking $y = \text{argmax}_k f_k(\boldsymbol{x})$. The

[16] It is often used since this function is defined in the dot product space with the infinite number of mapping.

unblanced class can be alleviated by **scaling the margin on both side by the inverse of the number of training samples**. One-vs-one takes $\frac{K(K-1)}{2}$ SVMs. This method also has two issues. One is the class-undefined regions and the other is the more classifiers. The first issue is addressed by taking the summation of score over each class and choosing the best class. The second issue is not serious because each classifier has fewer support vectors.

As seen in the both cases, the independent training can cause the biased choices in the class-undefined regions. Therefore, there is a generalized version of SVMs, i.e. joint optimization of all the classifiers:

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{||\boldsymbol{w}||^2}{2} + C\sum_{i=1}^{N}\left(\max_{y\neq y_i}\max(0, 1 - f_{y_i}(\boldsymbol{x}) + f_y(\boldsymbol{x}))\right) \\ &\text{subject to } \forall y \neq y_i,\ f_{y_i}(\boldsymbol{x}) - f_y(\boldsymbol{x}) \geq 1 \end{aligned} \tag{42}$$

where $f_k(\boldsymbol{x}) = \boldsymbol{w}_k^\top \boldsymbol{x} + b_k$. This formulation promotes the larger margin between true and false classes and puts a penalty on the most confusing class.

7 Decision tree

Decision tree is a method that divides the space by several rules. Since the decision tree divides each data point using some splitting rules, the result has **nice interpretability** and it can draw flexible boundaries. On the other hand, if we allow deep splitting, it can easily **overfit**. Further benefits of this algorithm is that the computational complexity of the whole splitting is $O(nd\log n)$ and it is **scalable** with respect to training data size n. Other advantages are **the handle of categorical parameters** and **no influence from unimportant dimensions** [17] and **the flexible framework**, i.e. we can choose splitting criterion or the computation in each partition freely. Since this algorithm is **deterministic**, the decision tree is not suitable for a naïve ensemble. We will see how to ensemble this method in Random Forest (a bagging method).

7.1 Computation of the prediction

Given data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x} \subset \mathcal{X} \in \mathbb{R}^d, y \in \mathbb{R}$, the decision tree $\mathcal{T} : \mathbb{R}^d \to \mathbb{R}$ is the model which has partitions $\mathcal{P}_i \in \mathcal{X}$: satisfying the following properties:

Definition 6
Suppose the model has p partitions, the set of partition has to satisfy the following two properties:

1. $\forall i, j (1 \leq i < j \leq p, i, j \in \mathbb{N}), \mathcal{P}_i \cap \mathcal{P}_j = \emptyset$
2. $\bigcup_{i=1}^p \mathcal{P}_i = \mathcal{X}$

Roughly speaking, each partition does not have any overlap with the others and the union of all the partitions covers the whole feature space. After decision tree takes an input, input will be assigned to one of the partitions. Then, the output will be computed in the partition or just return the value assigned to the partition. In most cases, the output will be **the mean value of a partition** for regression and **the most frequent class in a partition** for classification tasks. However, we can substitute these operation with $f(\mathcal{D}_{\mathcal{P}_i})$ where $\mathcal{D}_{\mathcal{P}_i}$ is the set of $(\boldsymbol{x}_j, y_j)$ belonging to the partition $\mathcal{P}_i$. For example, when we know that there is class imbalance, we can put more weights on class with scarce samples.

The partitions will be determined using the indicator called **impurity** $H(\mathcal{D})$. This is **variance** for regression and **gini** or **cross entropy** for classification tasks. Decision tree iteratively split data points

[17] Because it handles each dimension separately.

into two partitions which **maximize** so-called **gain** $\mathcal{G}(B(\boldsymbol{x}))$ [18] where $B(\boldsymbol{x})$ is the splitting boundary [19]. Let $\mathcal{D}^{(L)}, \mathcal{D}^{(R)}$ be the data divided by the boundary $B(\boldsymbol{x})$. Then the gain $\mathcal{G}(B(\boldsymbol{x}))$ is computed as follows:

$$\mathcal{G}(B(\boldsymbol{x})) = nH(\mathcal{D}) - (n^{(L)}H(\mathcal{D}^{(L)}) + n^{(R)}H(\mathcal{D}^{(R)})) \tag{43}$$

where $n = |\mathcal{D}|, n^{(L)} = |\mathcal{D}^{(L)}|, n^{(R)} = |\mathcal{D}^{(R)}|$. As mentioned previously, $H(\mathcal{D})$ is

$$\begin{aligned} H(\mathcal{D}) &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \\ \bar{y} &= \frac{1}{n}\sum_{i=1}^{n} y_i \\ H(\mathcal{D}) &= -\sum_{c=1}^{K} p_c \log_2 p_c \end{aligned} \tag{44}$$

for regression and K-class classification tasks respectively where p_c is the occurrence of class c out of the number of data points in the leaf. The Gini index defined below is also used in classification tasks:

$$H(\mathcal{D}) = 1 - \sum_{c=1}^{K} p_c^2 \tag{45}$$

In decision tree, we split the given data to maximize the gain in Eq (43). In the case of cross entropy $H(\mathcal{D})$, another interpretation of the optimization of the information gain are the KL divergence as follows:

$$\begin{aligned} D_{\mathrm{KL}}(P||Q) &= \int P(y|\mathcal{P}_i)\log\frac{P(y|\mathcal{P}_i)}{Q(y)}dy \\ Q(y) &= \frac{1}{K} \text{ (for all } c = 1, 2, \cdots, K \text{)} \end{aligned} \tag{46}$$

The uniform distribution $Q(y)$ is the worst case, so the formulation of impurity can be interpreted as the maximization of the KL-divergence. This is equivalent to the **minimization of the impurity or variance**. The ppliting procedure is performed in a recursive manner and each boundary divides the i-th dimension in $x_i < v$ and $x_i \geq v$. Before implementing the algorithm, we convert categorical parameters into some integers. In the recursion, the data split function will be called n times at most and searching for the best splitting takes $O(nd)$. This is because there are n candidates $\{x_i\}_{i=1}^n$, i.e. $n-1$ possible splits, in each dimension. Additionally, each split can be computed by $O(1)$ except the first split in each dimension, which takes $O(n)$, if we sort each dimension. Therefore, the total complexity is $O(nd)$ [20]. The complexity of the whole algorithm takes $O(n^2 d)$ and the prediction takes $O(n)$ in the worst case [21]. On the other hand, the worst case does not happen often and the whole split takes $O(nd\log n)$ and the prediction takes $O(\log n)$ in the best case [22]. Note that decision tree can draw non-linear boundaries as well, but we do not handle the topic here.

7.2 Important hyperparameters of decision tree

1. **min leaf**: This controls the minimum number of data points for each partition. Since the larger number prevents partitions from being occupied by only one data point, it is not likely to have specific rules to classify exceptional data. Therefore, the larger number leads to generalization.

[18] if the $H(\mathcal{D})$ is cross entropy, it is called **information gain**

[19] In most cases, we divide in two, but we can also divide in K, but this can be achieved by a sequence of binary splits. Additionally, binary splits are faster and often yield better performance.

[20] If the splitting boundary is multi dimension or non-linear, we cannot compute each split in $O(1)$.

[21] The case where each partition has only one data point.

[22] The case where each splitting divides given data perfectly in half.

Algorithm 2 CART

$\mathcal{D}_{\text{cur}}(\text{data}), d_{\text{cur}}(\text{current depth}), v_{\text{cur}}(\text{current assinged value})$ ▷ Input
Max depth $d_{\max}$, min data points per leaf δ ▷ Hyperparameters
function DATA SPLIT($\mathcal{D}_{\text{cur}}, d_{\text{cur}}, v_{\text{cur}}$)
  if $|\mathcal{D}_{\text{cur}}| > \delta \cap d_{\text{cur}} \leq d_{\max}$ **then**
    (j, v) = Search Best Split($\mathcal{D}_{\text{cur}}$) ▷ The complexity is $O(nd)$
    Compute the assigned value for left v_L and right v_R
    $\mathcal{D}^{(L)} = \{(\boldsymbol{x}, y) \in \mathcal{D}_{\text{cur}} | \boldsymbol{x}_j \leq v\}$
    DataSplit($\mathcal{D}^{(L)}, d_{\text{cur}} + 1, v_L$)
    $\mathcal{D}^{(R)} = \{(\boldsymbol{x}, y) \in \mathcal{D}_{\text{cur}} | \boldsymbol{x}_j > v\}$
    DataSplit($\mathcal{D}^{(R)}, d_{\text{cur}} + 1, v_R$)
  else
    Assign v_{cur} to the current partition

2. **max depth**: Deeper trees have higher representational capacity and higher variance while shallower trees can generalize, but have higher bias.
3. **number of nodes**: The maximum number of nodes by the depth x is 2^x. However, it does not happen in most cases, so we can just specify the maximum number of nodes using this variable.
4. **leaf model**: $f(\mathcal{D}_{\mathcal{P}_i})$ can control the flexibility of the model
5. **split criterion**: It also changes the model

8 Ensembles

8.1 Bagging

Bagging is to ensemble models trained on bootstrapped data.

8.1.1 General settings

Expected test error is decomposed as $(\text{Variance}) + (\text{Bias})^2 + (\text{Noise})$ and the variance can be computed as

$$\mathbb{E}_{\boldsymbol{x}, \mathcal{D}}[\hat{f}(\boldsymbol{x}; \mathcal{D}) - \bar{f}(\boldsymbol{x})^2] \tag{47}$$

where $\bar{f}(\boldsymbol{x}) = \int \hat{f}(\boldsymbol{x}; \mathcal{D}) p(\mathcal{D}) d\mathcal{D}$ Therefore, when we take the expected value of $\hat{f}(\boldsymbol{x}, \mathcal{D})$ for various training dataset $\mathcal{D}$ sampled from i.i.d, the variance can be reduced. More formally, we take subsets $\mathcal{D}_k = \{(\boldsymbol{x}_{i_{k,j}}, y_{i_{k,j}})\}_{j=1}^n$ of the whole training data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ where $\forall k (1 \leq k \leq K), \boldsymbol{i}_k$ is a subset of $\{1, 2, \cdots, N\}$ with the length of n. Then the empirical formulation of $\hat{f}$ is the following:

$$\hat{f}(\boldsymbol{x}; \mathcal{D}) = \frac{1}{K} \sum_{k=1}^{K} \hat{f}_k(\boldsymbol{x}; \mathcal{D}_k) \tag{48}$$

This ensemble technique is called **Bagging**. Since the bootstrapped data share some data points with some other data, $\forall k, \mathcal{D}_k$ are not i.i.d and Eq (48) does not correspond with $\bar{f}$. It is important to **reduce the overlaps** between each data $\mathcal{D}_k$ and **uncorrelate each model** $\hat{f}_k$. Suppose we would like to solve a regression task and each model yields the expected squared error of $v = \mathbb{E}_{\boldsymbol{x}}[\epsilon_i^2]$ and any pairs of two models yield the covariance of errors of $c = \mathbb{E}_{\boldsymbol{x}}[\epsilon_i \epsilon_j]$. Then, the overall variance is the

Algorithm 3 Random Forests

function RANDOM FORESTS
  for $k = 1, \dots, K$ **do**
    $\mathcal{D}_k \sim \mathcal{D}$ ▷ Bootstrapping
    Select d dimensions from D dimensions
    Obtain $\hat{f}_k$ by performing CART on the selected d dimensions
  return $\frac{1}{K}\sum_{k=1}^{K} \hat{f}_k$

following:

$$\begin{aligned}\mathbb{E}\left[\left(\frac{1}{K}\sum_{k=1}^{K}\epsilon_k\right)^2\right] &= \frac{1}{K^2}\mathbb{E}\left[\sum_{k=1}^{K}\epsilon_k^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right] \\ &= \frac{K}{K^2}v + \frac{K(K-1)}{K^2}c \\ &= \frac{1}{K}v + \underbrace{\frac{K-1}{K}c}_{\text{Correlation term}}\end{aligned} \tag{49}$$

When there is strong correlation, c goes to v and the overall variance goes to v. On the other hand, when the correlation is smaller, the second term goes to zero and the overall variance goes to v/K.

8.1.2 Random Forests

Bagging requires **low-biased** and **uncorrelated** models. Low bias can be achieved by decision tree. Therefore, **bagged trees** has been invented. However, since each tree can be correlated, **random forests** introduces the random spliting shown in Algorithm 3 to decrease the correlation between each tree. The random forests can evaluate the uncertainty of an ensemble as follows:

$$\sigma(\boldsymbol{x}) = \frac{1}{K}\sum_{k=1}^{K}\left(\hat{f}_k(\boldsymbol{x};\mathcal{D}_k) - \hat{f}(\boldsymbol{x};\mathcal{D})\right)^2 \tag{50}$$

When each model yields similar outputs, the overall confidence becomes higher and the variance approaches zero. Additionally, the test error of the model can be computed using the following **out-of-bag training error**:

$$\frac{1}{N}\sum_{i=1}^{N} = L\left(y_i, \frac{1}{|B_i^{\text{out}}|}\sum_{k\in B_i^{\text{out}}}\hat{f}_k(\boldsymbol{x}_i)\right) \tag{51}$$

where B_i^{out} is the set of all integers k s.t. $(\boldsymbol{x}_i, y_i) \notin \mathcal{D}_k$.

8.2 Boosting

Another ensemble method is boosting and this method adds new models iteratively to reduce the loss. The main goal of **Boosting** is to obtain a low-biased ensemble by aggregating high-biased models. The aggregation is performed in a **sequential** manner. The final ensemble is the weighted sum of each model $F_K(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}) = \sum_{k=1}^{K}\alpha_k\hat{f}_k(\boldsymbol{x})$. In other words, we add the $(k+1)$-th model $\alpha_{k+1}\hat{f}_{k+1}$ to the fixed cummulated model $F_k(\boldsymbol{x})$ and train only the $(k+1)$-th model. Note that the selection of α_k is typically uniform over each base model and it is a learnable parameter in the case of AdaBoost.

Algorithm 4 Gradient boosting

$\forall i \in \{1, \cdots, N\}, F_0(\boldsymbol{x}_i) = 0$ ▷ objective function

function GRADIENT BOOSTING

  for $k = 0, 1, \cdots, K-1$ **do**

    $\forall i \in \{1, \cdots, N\}, g_{k,i} = \left(\frac{\partial L(y_i, F)}{\partial F}\right)_{F=F_{k,i}}$

    Train the $(k+1)$-th model $\hat{f}_{k+1} := \mathrm{argmin}_f \sum_{i=1}^N (f(\boldsymbol{x}_i) + g_{k,i})^2$

    Update $F_{k+1} = F_k + \alpha_{k+1}\hat{f}_{k+1}$ ▷ α_{k+1} depends on the inference

  return F_K

8.2.1 Gradient boosting

Using the Taylor approximation, each term can be reformulated as follows:

$$\begin{aligned} L(y_i, F_{k,i} + \alpha_{k+1}\hat{f}_{k+1,i}) &\simeq L(y_i, F_{k,i}) + \alpha_{k+1}\left(\frac{\partial L}{\partial F}\right)_{F=F_{k,i}} \hat{f}_{k+1,i} \\ \sum_{i=1}^N L(y_i, F_{k,i} + \alpha_{k+1}\hat{f}_{k+1,i}) &\simeq \sum_{i=1}^N L(y_i, F_{k,i}) + \alpha_{k+1} \underbrace{\sum_{i=1}^N g_{k,i}\hat{f}_{k+1,i}}_{\text{minimize}} \end{aligned} \tag{52}$$

where $g_{k,i} = \left(\frac{\partial L(y_i, F)}{\partial F}\right)_{F=F_{k,i}}$, $F_{k,i} = F_k(\boldsymbol{x}_i)$ and $\hat{f}_{k+1,i} = \hat{f}_{k+1}(\boldsymbol{x}_i)$. Since we would like to minimize the second term, the optimization problem is equivalent to the following:

$$\begin{aligned} \hat{f}_{k+1} &= \mathrm{argmin}_f \sum_{i=1}^N g_{k,i} f(\boldsymbol{x}_i) \\ &= \mathrm{argmin}_f \sum_{i=1}^N (2 g_{k,i} f(\boldsymbol{x}_i) + g_{k,i}^2) \\ &= \mathrm{argmin}_f \sum_{i=1}^N (f(\boldsymbol{x}_i) + g_{k,i})^2 \end{aligned} \tag{53}$$

Note that the last transformation assumes that $\sum_{i=1}^N \hat{f}_{k+1,i}^2 = \text{const}$. This optimization takes over the following opmality:

$$\begin{aligned} f(x + \Delta x) &\simeq f(x) + \frac{df}{dx}\Delta x \\ f(x + \Delta x) &\simeq f(x) - \alpha\left(\frac{df}{dx}\right)^2 \leq f(x) \end{aligned} \tag{54}$$

where we take $\Delta x = -\alpha \frac{df}{dx}$.

8.2.2 AdaBoost

AdaBoost is a binary classification method using boosting. Both labels and predictions are either 1 or -1. Predictions are made by the following weighted majority vote:

$$\begin{aligned} F_k(\boldsymbol{x}) &= \mathrm{sign}\left(\sum_{i=1}^{k} \alpha_i f_i(\boldsymbol{x})\right) \\ \alpha_k &= \ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right) \\ \epsilon_k &= \frac{1}{N}\sum_{i=1}^{N} \frac{w_{k,i}\mathbb{1}(y_i \neq f_k(\boldsymbol{x}_i))}{\sum_{j=1}^{N} w_{k,j}} \end{aligned} \tag{55}$$

where $w_{i,k}$ is the weight

$$w_{k,i} = \begin{cases} w_{k-1,i}e^{\alpha_k} \ (\text{if } y_i \neq f_k(\boldsymbol{x}_i)) \\ w_{k-1,i} \ (\text{otherwise}) \end{cases} \tag{56}$$

Note that the priority $w_{i,k}$ goes up significantly when the k-th model is more accurate and the model predicts the i-th instance correctly. In other words, the more the i-th instance is difficult, the larger the priority of correct answers to the i-th instance becomes. The optimization of each model is formulated as follows:

$$\begin{aligned} f_k &= \mathrm{argmin}_f \ \epsilon_k \\ &\simeq \mathrm{argmin}_f \sum_{i=1}^{N} w_{k,i} L(y_i, f(\boldsymbol{x}_i)) \\ &\simeq \mathrm{argmin}_f \underbrace{\sum_{i=1}^{N} w_{k,i} e^{-y_i f(\boldsymbol{x}_i)}}_{\text{surrogate loss}} \end{aligned} \tag{57}$$

AdaBoost iteratively complements the ensemble model by adding an additional model solving more difficult instances. In the end, AdaBoost returns the following:

$$F_K(\boldsymbol{x}) = \mathrm{sign}\left(\sum_{k=1}^{K} \alpha_k f_k(\boldsymbol{x})\right) \tag{58}$$

Note that models that have larger α_k can solve harder instances.

8.3 Gradient boosting decision trees (GBDT)

GBDT combines gradient boosting and decision trees. Let p be the number of partitions one tree has and $t(\boldsymbol{x}_i)$ be the index of the partition that $\boldsymbol{x}_i$ belongs to, i.e. $\boldsymbol{x}_i \in \mathcal{P}_{t(\boldsymbol{x}_i)}$. Addtionally, f_i be the output value of the i-th partition. Then the objective function is the following:

$$\begin{aligned} &\mathrm{argmin}_{f^{(k)}} \left(\sum_{i=1}^{n} L^{(k)}(y_i, \hat{y}_i^{(k-1)} + f^{(k)}(\boldsymbol{x}_i)) + \Omega(f^{(k)})\right) \\ &\text{where } \hat{y}^{(k)}(\boldsymbol{x}_j) = \sum_{i=1}^{k} f^{(i)}(\boldsymbol{x}_j), \\ &\Omega(f^{(k)}) = \gamma p + \frac{\lambda}{2}\sum_{i=1}^{p} ||f_i^{(k)}||^2, \end{aligned} \tag{59}$$

γ, λ are the hyperparameters to control the regularization effect. The first term of the regularization term helps to reduce the number of partitions for less important variables and the second term prevents the model from fitting outliers and moderates influences from one leaf. Using the second-order Taylor approximation with respect to $f^{(k)}$, we obtain the following:

$$\begin{aligned} L^{(k)}(y_i, \hat{y}_i^{(k-1)} + f^{(k)}(\boldsymbol{x}_i)) &= L_i^{(k)} \\ &= L_i^{(k-1)} + \underbrace{\frac{\partial L_i^{(k-1)}}{\partial \hat{y}_i^{(k-1)}}}_{=g_i^{(k-1)}} f_i^{(k)} + \underbrace{\frac{\partial^2 L_i^{(k-1)}}{\partial (\hat{y}_i^{(k-1)})^2}}_{=h_i^{(k-1)}} \frac{(f_i^{(k)})^2}{2} \\ &= L_i^{(k-1)} + g_i^{(k-1)} f_i^{(k)} + h_i^{(k-1)} \frac{(f_i^{(k)})^2}{2} \end{aligned} \tag{60}$$

where g, h are the gradient and Hessian, respectively. Since $L_i^{(k-1)}$ is constant with respect to $f^{(k)}$ and the optimization with respect to $f^{(k)}$ is equivalent to that with respect to $\forall i, f_i^{(k)}$, the objective is the following:

$$\begin{aligned} &\mathrm{argmin}_{f_1^{(k)}, \dots, f_p^{(k)}} \left(\sum_{i=1}^{n} \left(g_i^{(k-1)} f_i^{(k)} + h_i^{(k-1)} \frac{(f_i^{(k)})^2}{2} \right) + \gamma p + \frac{\lambda}{2} \sum_{i=1}^{p} ||f_i^{(k)}||^2 \right) \\ &\mathrm{argmin}_{f_1^{(k)}, \dots, f_p^{(k)}} \sum_{i=1}^{p} \left(\left(\sum_{j \in \mathcal{P}_i} g_j^{(k-1)} \right) f_i^{(k)} + \left(\sum_{j \in \mathcal{P}_i} h_j^{(k-1)} + \lambda \right) \frac{(f_i^{(k)})^2}{2} + \gamma \right) \end{aligned} \tag{61}$$

When we define $G_i = \sum_{j \in \mathcal{P}_i} g_j^{(k-1)}$ and $H_i = \sum_{j \in \mathcal{P}_i} (h_j^{(k-1)} + \lambda)$, the optimal value for each partition is the following:

$$\mathrm{argmin}_{f_1^{(k)}, \dots, f_p^{(k)}} \sum_{i=1}^{p} \Bigg(\underbrace{G_i f_i^{(k)} + H_i \frac{(f_i^{(k)})^2}{2}}_{\text{optimal at } f_i^{(k)} = -\frac{G_i}{H_i}} + \gamma \Bigg) \tag{62}$$

Therefore, the objective of the i-th partition can be computed as of the following $\mathcal{O}(\boldsymbol{x} \in \mathcal{P}_i) = -\frac{G_i^2}{2H_i} + \gamma$ and when spliting the i-th partition in two, the gain is the following:

$$\begin{aligned} \mathcal{G}(B(\boldsymbol{x})) &= \mathcal{O}(\boldsymbol{x} \in \mathcal{P}_i) - \left(\mathcal{O}(\boldsymbol{x} \in \mathcal{P}_{i_{\text{left}}}) + \mathcal{O}(\boldsymbol{x} \in \mathcal{P}_{i_{\text{right}}}) \right) \\ &= \frac{1}{2} \left(-\frac{G_i^2}{H_i} + \frac{G_{i_{\text{left}}}^2}{H_{i_{\text{left}}}} + \frac{G_{i_{\text{right}}}^2}{H_{i_{\text{right}}}} \right) - \gamma \end{aligned} \tag{63}$$

The objective is to maximize the gain and $\mathcal{G}(B(\boldsymbol{x})) \leq 0$ is termination criterion. Note that the factor n in decision tree is the coefficient to match the scale and we do not need scale factor in this formulation. Since the gradient and Hessian can be pre-computed, the searching for the optimal split can be performed by $O(nd)$ and the total complexity of the training is $O(knd\log p)$ in the case of balanced trees and k trees ensemble. The prediction can be computed by $O(k\log p)$ on average. It can naturally handle categorical parameters and it is widely used for tabular data, because this method is low-biased and has relatively low variance.

9 Hyperparameter optimization (HPO)

9.1 General settings

The hyperparameter configurations of ML can significantly affect the performance of the model. For this reason, the HPO is essential. In the typical settings, we first train a model with a hyperparameter

configuration θ and evaluate the performance via the result on the validation dataset. We repeat such procedure and take the configuration that has the best performance on the validation dataset in the end. In other words, we would like to solve the following:

$$\theta^\star \in \mathrm{argmin}_\theta L(\theta, \mathcal{D}_{\mathrm{train}}, \mathcal{D}_{\mathrm{valid}}) = \mathrm{argmin}_\theta f(\theta) \tag{64}$$

where we assume supervised learning in this case and $f(\theta)$ is the objective function. The performance of HPO itself is measured via the performance on the test dataset after taking the best configuration on the validation dataset. Since the searching space determines the possible global solutions, it is important to pay attention to which hyperparameters to tune and in what ranges we optimize. Meta-learning can guide these settings. When we would like to generalize the performance more, we often take the average of the loss from **k-fold cross validation**. Cross validation first splits the dataset into k parts. Then, we train the model on $k-1$ parts and validate on the rest. Since there are k possible selections of the validation dataset, we can yield the k-th validation loss. In the cross validation, we take the average of all the loss and report the average as the performance. Note that the case of $k=1$ is called hold-out validation and the case where k is equal to the number of data points is called leave-one-out cross validation. Leave-one-out cross validation is used when there are not sufficient training data points. Note that one advantage of cross validation ($k \geq 2$) is to use all the possible data points for training as a whole.

9.2 Optimization methods

The most naïve methods are grid search and random search. Since random search handles the **low intrinsic dimensionality** better, it is preferred especially in the high dimension. Additionally, since grid search takes the grids over searching space, it cannot stop the optimization although **random search can stop the optimization anytime**. However, both methods are typically inefficient because we do not make use of the observations.

One example of more efficient method is **evolutionary algorithms (EA)**. This method improves the quality of solutions by evolution strategies. General scheme follows:

1. **initialize population / evaluate population**: sample λ candidates randomly and evaluate each of them [23]
2. **parents selection**: choose parents for the next generation such as random selection , fitness-proportional selection and tournament selection [24]
3. **variation**: choose parents and apply crossover and mutation
4. **survival selection**:

The major advantage of EA is the customization of each component. On the other hand, this customization is necessary for complex problems and it is hard to set it properly. It is often hard to balance exploration and exploitation. Additionally, there are not many theories and it often requires more evaluations for the convergence.

Note that those three methods can trivially evaluate configurations in parallel and **handle all the parameter types**.

9.3 Sequential model-based optimization

The most naïve HPO methods are random search and grid search. However, since both methods do not use the observations, they are not efficient. To make use of the previous knowledge, sequential model-based optimization (SMBO) has been invented.

[23] EA employs restarts with new randomly generated populations to reduce the influence from the initialization.
[24] Choose K candidates from the pool and pick the best one. Repeat this procedure λ times.

Table 3: The advantages and Disadvantages of each surrogate model. When it says "poor", it means it is poor compared to the other models.

Surrogate models	Advantages	Disadvantages
GP	smoothness	requires $O(n^3)$
	reliable uncertainty estimates	curse of dimensionality
	flexible kernel design	sensitivity to the choice of kernel [25]
	better sample efficiency	
Random Forest	cheaper training cost	poor extrapolation ability[26]
	scalability to dimensions	poor uncertainty estimate
	scalability to # of data points	hard to incorporate prior
	handling of diverse parameters	
	robust to its own hyperparameters	
BNN	scalability to # of data points	many meta-design decision
	smooth uncertainty estimates	requires more data
	handling of diverse parameters[27]	poor uncertainty estimates

9.3.1 Surrogate models

Since the objective function for HPO is expensive in most cases, we use a surrogate model to yield promising configurations. For surrogate models, we need to achieve **accurate predictions** and **the model uncertainty representation**. Additionally, we need to consider **cost of training**, **scalability to the number of data points and dimensions** and **the handling of categorical and conditional parameters.**

Typically, we employ a machine learning model and here we discuss gaussian process (GP) regression model. GP is the stochastic process assuming that the objective function $f(\theta)$ follows the gaussian distribution with an arbitrary configuration as follows:

$$f(\theta) \sim \mathcal{N}(\mu(\theta), \sigma^2(\theta)) \tag{65}$$

where the mean and the variance are approximated by the following:

$$\begin{aligned} \mu(\theta) &= \boldsymbol{k}^\top (\boldsymbol{K} + \lambda^2 \boldsymbol{I})^{-1} \boldsymbol{f} \\ \sigma^2(\theta) &= k(\theta, \theta) - \boldsymbol{k}^\top (\boldsymbol{K} + \lambda^2 \boldsymbol{I})^{-1} \boldsymbol{k} \end{aligned} \tag{66}$$

where λ is the coefficient of L2 regularization term, the observation $\boldsymbol{f} = [f(\theta_1), \cdots, f(\theta_n)]$, $\boldsymbol{k} = [k(\theta, \theta_1), \cdots, k(\theta, \theta_n)]$ and $\boldsymbol{I}$ is an n-dimensional identity matrix. Additionally, $k : \Theta \times \Theta \to \mathbb{R}$ is the kernel function and $\boldsymbol{K}$ is the covariance with the (i, j)-th element $k(\theta_i, \theta_j)$. More formally, the following holds:

$$\tilde{\boldsymbol{f}} \sim \mathcal{N}(\boldsymbol{f}, \boldsymbol{K}) \tag{67}$$

Other examples are random forest and Bayesian neural networks (BNN). The typical examples of BNN are DNGO and BOHAMIANN. The advantages and the shortcomings are listed in Table 3. Note that the random forest estimates uncertainty using empirical variance across tree predictors. This is not exactly a surrogate model, but Tree-structured parzen estimator (TPE) is also a major method in HPO. The computational complexity of this model is $O(nd)$ and it handles conditional and categorical parameters. Additionally, since this model is a sampling-based model, it can effectively evaluate multiple configurations in parallel. However, TPE is less sample-efficient than GP in low dimensions.

[27] Therefore, we often employ mariginalization of its own hyperparameters to make it more robust.

[27] This is because it does not have splits outside in nature and it returns a constant value.

[27] Random forest can handle conditional parameters, but BNN cannot handle conditional parameters. However, both can handle categorical parameters.

9.3.2 Acquisition functions

In the SMBO, while we would like to exploit the knowledge, we would like to explore the unfamiliar region to avoid falling into the local minimum. The typical method is the expected improvement (EI) as follows:

$$\begin{aligned} \mathrm{EI}_{f^\star}(\theta) &= \int_{-\infty}^{f^\star} (f^\star - f) p(f|\theta) df \\ &= \sigma(\theta)\Big(z(\theta)\Phi(z(\theta)) + \phi(z(\theta)) \Big) \end{aligned} \tag{68}$$

where $z(\theta) = \frac{f^\star - \mu(\theta)}{\sigma(\theta)}$, ϕ is the standard normal distribution and Φ is the cumulative distribution of the standard normal distribution. Other acquisition functions are lower confidence bound (LCB) and upper confidence bound (UCB) as follows:

$$\begin{aligned} \mathrm{UCB}(\theta) &= \mu(\theta) + \beta\sigma(\theta) \\ \mathrm{LCB}(\theta) &= \mu(\theta) - \beta\sigma(\theta) \end{aligned} \tag{69}$$

In most cases, we use UCB for maximization problems and LCB for minimization problems.

Another example is the following probability of improvement (PI):

$$\mathrm{PI}_{f^\star}(\theta) = \int_{-\infty}^{f^\star} p(f|\theta) df \tag{70}$$

The Thompson sampling samples the surrogate model $f(\theta)$ and maximize such sampled function:

$$\begin{aligned} \tilde{f}(\theta) &\sim p(f|\theta) \\ \theta^\star &\in \mathrm{argmin}_\theta \tilde{f}(\theta) \end{aligned} \tag{71}$$

Since the Thompson sampling is a stochastic method, it is robust to the parallel evaluation settings.

The entropy search evaluates from the location where we can reduce the uncertainty about the location of $\theta^\star$ by minimizing:

$$\theta^\star \in \mathrm{argmax}_{\theta'} \left(H(p(\theta'|\mathcal{D})) - \int H(p(\theta'|\mathcal{D} \cup \{(\theta, f)\})) p(f|\theta) df \right) \tag{72}$$

where $p(\theta|\mathcal{D}) = p(\theta \in \mathrm{argmin}_{\theta'} f(\theta'|\mathcal{D}))$.

The knowledge gradient is the optimization of the expected improvement of the mean value after the evaluation:

$$\theta^\star \in \mathrm{argmax}_{\theta} \left(\min_{\theta'} \mu(\theta'|\mathcal{D}) - \int \min_{\theta'} \mu(\theta'|\mathcal{D} \cup \{(\theta, f)\}) p(f|\theta) df \right) \tag{73}$$

9.3.3 Extensions of Bayesian optimization

Since one of the drawbacks of GP is the curse of dimensionality, two major solutions have been introduced. One is to map the hyperparameter space to the lower dimensional space using a randomly generated matrix. The other is to fit models to a subset of dimensions.

Another drawback is the handling of categorical and conditional parameters. To address this issue, one-hot encoding, hamming distance and entity embedding have been invented. For conditional parameters, some kernels have been introduced, but it is still actively studied.

Algorithm 5 Successive halving

$R, n, \eta(>1)$ ▷ Running cost and the number of configurations to be evaluated and cut-off ratio
function SUCCESSIVE HALVING
  $N = \lfloor \log_2 n \rfloor$, Sample $[\theta_1, \cdots, \theta_n]$
  for $i = 0, 1, \ldots, N-1$ **do**
    Train the survived configurations with the budget of $\lfloor \eta^{i-N+1} R \rfloor$
    $n = \lfloor \eta^{-i} n \rfloor$, pick the top n configurations
  return $\theta^\star$

Algorithm 6 Hyperband

function HYPERBAND
$R, \eta(>1)$
  $s_{\max} = \lfloor \log_\eta R \rfloor, B = (s_{\max}+1)R$
  for $s = s_{\max}, \ldots, 0$ **do**
    $n = \lceil \eta^s \frac{s_{\max}+1}{s+1} \rceil$
    $\theta =$ successive halving(n, R, η)
  return $\theta^\star$

9.4 Multi-fidelity optimization

Since the evaluation of HPO is expensive, we often use multi-fidelity optimization to accelerate HPO. The major example is **learning curve prediction** and the other examples are **successive halving** and **hyperband**. Successive halving is cutting off poor configurations and putting much more budget on better configurations. The major issue of the algorithm is the trade-off between the performance and budget. Basically, when we take large n, it leads to aggressive cut-off and poor performance. On the other hand, while smaller n leads to good performance, it leads to less reduction of the computational time. Therefore, Hyperband controls such trade-off by changing the parameter iteratively.

10 Error metrics

10.1 Binary classification

In real-world problems, the class distribution is not necessarily uniform. Furthermore, when the dataset includes 99% of label 0 and 1% of label 1, we can easily achieve the 99% accuracy by predicting 0 for every input. Therefore, it is important to divide the result into the four patterns listed in Table 4. Using these results, we define the following metrics:

1. **True positive rate (TPR or recall)**: $\frac{\mathrm{TP}}{\mathrm{TP + FN}} = \frac{\text{\# of True Positive}}{\text{\# of Positive labels}}$, prefer FP to FN such as cancer test
2. **False positive rate (FPR)**: $\frac{\mathrm{FP}}{\mathrm{TN + FP}} = \frac{\text{\# of False Positive}}{\text{\# of Negative labels}}$, prefer FN to FP
3. **precision**: $\frac{\mathrm{TP}}{\mathrm{TP + FP}} = \frac{\text{\# of True Positive}}{\text{\# of Positive predictions}}$, prefer FN to FP such as optimal coupon allocation
4. **F1 score**: $2(\frac{1}{\text{precision}} + \frac{1}{\text{recall}})^{-1}$, trade-off of precision and recall

False positive rate and true positive rate are used in receiver operating characteristic (ROC). ROC takes FPR for the horizontal axis and TPR for the vertical axis as shown in Figure 2. In the normal setting, the trained model predicts label 1 when the output $f(\boldsymbol{x}) \geq \epsilon = \frac{1}{2}$. ROC is plotted by observing the TPR and FPR for arbitrary thresholds ϵ. In the best case, TPR is close to 1 and FPR is close

Table 4: More detailed results used for imbalanced datasets.

label / prediction	Positive	Negative
Positive	True positive	False negative
Nagative	False positive	True negative

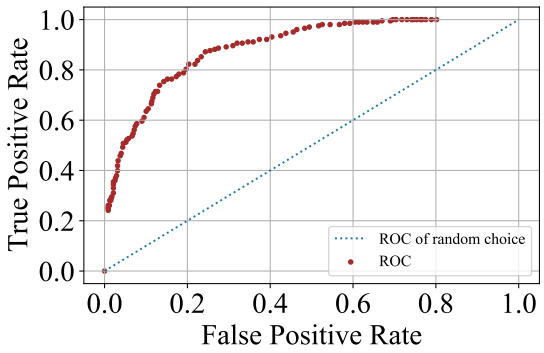

Figure 2: ROC of the case of positive : negative = 0.2 : 0.8 The optimal threshold is the one that achieves the maximum margin between ROC and ROC of random choice.

to 0. This is realized by maximizing so-called AUC (area under the ROC curve). AUC is maximized through the following pairwise rank proxy:

$$\mathrm{argmax}_{\boldsymbol{w}}\mathrm{AUC}(\boldsymbol{y}, \hat{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{w})) \simeq \mathrm{argmax}_{\boldsymbol{w}} \sum_{i|y_i=0} \sum_{j|y_j=1} \sigma(f(\boldsymbol{x}_i; \boldsymbol{w}) - f(\boldsymbol{x}_j; \boldsymbol{w})) \tag{74}$$

where σ is the sigmoid function. Note that the sigmoid function is introduced to ignore the pairs with larger values and intensively optimize the pairs with smaller values.

Another metric is F1 score. F1 score balances the trade-off between precision and recall. Since this metric is non-differentiable and non-decomposable [28], the optimization is performed via the following practical heuristic:

1. oversample the minority class
2. train the model with a proxy of the misclassification rate (e.g. logistic loss)
3. tune all the hyperparameters including the threshold ϵ with respect to the validation F1 score

10.2 Regresion task

In the most cases, root mean square error is used. However, when we would like to robustify the model against outliers, we often use **mean absolute error**. Additionally, the following **mean absolute percentage error** can optimize independently from the scale:

$$\frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{75}$$

[28] This is because recall and precision are defined on mini-batch.

The major issue of this metric happens when the label y is close to zero. Therefore, the alternative metric is the following **mean absolute scaled error**:

$$\frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i-\hat{y}_i}{\frac{1}{N-1}\sum_{j=2}^{N}|y_j-y_{j-1}|}\right| \tag{76}$$

The advantage of this metric is the stability. On the other hand, we cannot interpret the metric as percentage anymore.

10.3 Ranking

In this section, we assume that our problem is to rank based on the relevance of articles given a query. Suppose we have a dataset $\mathcal{D}=\{(\boldsymbol{x}_i,y_i)\}_{i=1}^{N}$ where $\boldsymbol{x}_i$ is the feature vector and y_i is the relevance. The goal is to learn $f(\boldsymbol{x})$ that predicts the relevance.

10.3.1 Discounted cumulative gain (DCG) and normalized DCG (NDCG)

The DCG until the K-th best is computed as follows:

$$\sum_{i=1}^{K}\frac{2^{y_i}-1}{\log(i+1)} \tag{77}$$

where y_i is sorted according to the estimated relevance $f(\boldsymbol{x}_i;\boldsymbol{w})$. In other words, $f(\boldsymbol{x}_i;\boldsymbol{w})\geq f(\boldsymbol{x}_j;\boldsymbol{w})$ holds for $\forall i,j$ s.t. $i<j$. The K-th NDCG is computed as the ratio of the K-th DCG and the K-th ideal DCG, i.e. DCG computed under the condition where $y_1\geq y_2\geq\cdots\geq y_N$.

10.4 Pairwise rank loss

Given a ranking order among all articles of query q:

$$i<_q j \text{ iff } y_i>y_j \tag{78}$$

Then we estimate the probability that a pair is correctly ranked as:

$$\hat{p}_{i,j}=\hat{p}(i<_q j)=\frac{1}{1+e^{-(f(\boldsymbol{x}_i;\boldsymbol{w})-f(\boldsymbol{x}_j;\boldsymbol{w}))}} \tag{79}$$

The loss of a pair i,j is the following:

$$L_{i,j}=-p_{i,j}\log\hat{p}_{i,j}-(1-p_{i,j})\log(1-\hat{p}_{i,j}) \tag{80}$$

Using the chain rule, we can derive the derivative of this equation.

10.4.1 LambdaRank heuristic

LambdaRank heuristic puts more weights on the change which leads to more increse in the NDCG. Roughly speaking, this method prioritizes the change in the higher ranks. Such importance is measured by the following:

$$\lambda_{i,j}\simeq-\frac{|\Delta\text{NDCG}_{i,j}|}{1+e^{f(\boldsymbol{x}_i;\boldsymbol{w})-f(\boldsymbol{x}_j;\boldsymbol{w})}} \tag{81}$$

where $\Delta\text{NDCG}_{i,j}$ is the increase of NDCG by swapping the ranking positions of i,j. The update of the weights are performed by the following:

$$\begin{aligned}\boldsymbol{w}&=\boldsymbol{w}+\alpha\sum_{i=1}^{N}\lambda_i\frac{\partial f(\boldsymbol{x}_i;\boldsymbol{w})}{\partial\boldsymbol{w}}\\ \lambda_i&=\sum_{j\neq i}\lambda_{i,j}-\sum_{j\neq i}\lambda_{j,i}\end{aligned} \tag{82}$$