Overfitting, Cross Validation, and Regularization

Statistical Machine Learning (ENGG*6600*02)

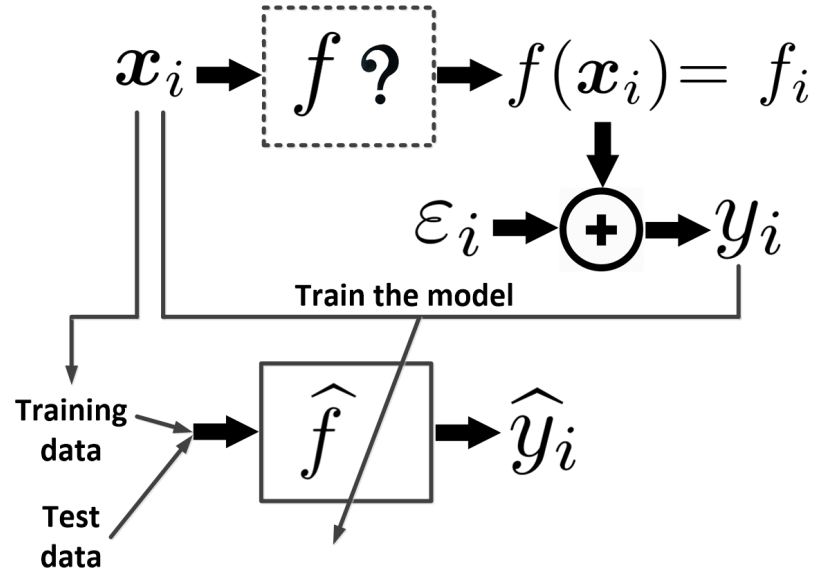
School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Summer 2023

Measures for a Model

Learning Model

- Assume we have a function f which gets the i-th input $\boldsymbol{x}_i$ and outputs $f_i = f(\boldsymbol{x}_i)$. This figure shows this function and its input and output:



- We wish to know the function which we call it the **true model** but we do not have access to it as it is unknown.
- Also, the pure outputs (true observations), f_i's, are not available. The output may be corrupted with an additive noise ε_i:

$$y_i = f_i + \varepsilon_i, \tag{1}$$

where the noise is $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$.

Learning Model

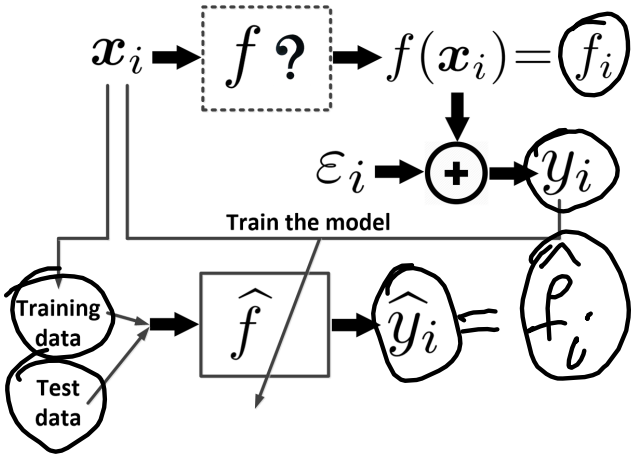
- We have:

$$y_i = f_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, \sigma^2) \Longrightarrow \mathbb{E}(\varepsilon_i) = 0, \quad \mathbb{E}(\varepsilon_i^2) = \mathbb{V}\text{ar}(\varepsilon_i) + (\mathbb{E}(\varepsilon_i))^2 = \sigma^2, \tag{2}$$

- The true observation f_i is not random, thus:

$$\mathbb{E}(f_i) = f_i. \tag{3}$$

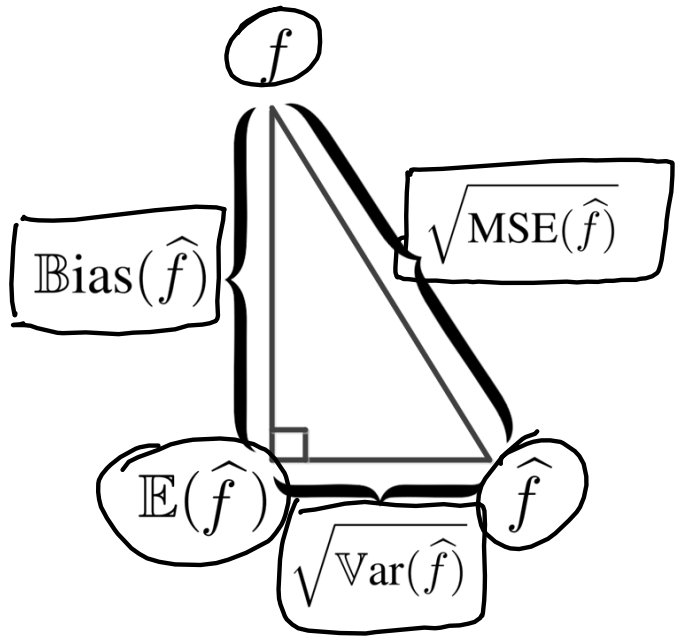
- The input training data $\{\boldsymbol{x}_i\}_{i=1}^n$ and their corrupted observations $\{y_i\}_{i=1}^n$ are available to us. We would like to approximate (estimate) the true model by a **model** $\widehat{f}$ in order to estimate the observations $\{y_i\}_{i=1}^n$ from the input $\{\boldsymbol{x}_i\}_{i=1}^n$.
- Calling the estimated observations by $\{\widehat{y}_i\}_{i=1}^n$, we want the $\{\widehat{y}_i\}_{i=1}^n$ to be as close as possible to $\{y_i\}_{i=1}^n$ for the training input data $\{\boldsymbol{x}_i\}_{i=1}^n$.
- We train the model using the training data in order to estimate the true model.
- After training the model, it can be used to estimate the output of the model for both the training input $\{\boldsymbol{x}_i\}_{i=1}^n$ and the unseen test input $\{\boldsymbol{x}_i\}_{i=1}^m$ to have the estimates $\{\widehat{y}_i\}_{i=1}^n$ and $\{\widehat{y}_i\}_{i=1}^m$, respectively.



Learning Model

- Here, we denote the estimation of the observation of the i-th instance with either $\widehat{y_i}$ or $\widehat{f_i}$.
- The model can be a **regression (prediction)** or **classification** model. In regression, the model's estimation is continuous while in classification, the estimation is a member of a discrete set of possible observations.
- The definitions of variance, bias, and MSE can also be used for the estimation $\widehat{f_i}$ of the true model f_i.

$$\mathsf{MSE}(\widehat{f}) = \mathbb{V}\mathsf{ar}(\widehat{f}) + (\mathbb{B}\mathsf{ias}(\widehat{f}))^2 \Longrightarrow \left(\sqrt{\mathsf{MSE}(\widehat{f})}\right)^2 = \left(\sqrt{\mathbb{V}\mathsf{ar}(\widehat{f})}\right)^2 + (\mathbb{B}\mathsf{ias}(\widehat{f}))^2. \quad (4)$$



Mean Squared Error of the Estimation of Observations

Mean Squared Error of the Estimation of Observations

- Suppose we have an instance $(\boldsymbol{x}_0, y_0)$. This instance can be either a training or test/validation instance. We will cover both cases.
- According to Eq. (1), the observation y_0 is:

$$y_0 = f_0 + \varepsilon_0. \tag{5}$$

- Assume the model's estimation of y_0 is $\widehat{f}_0$. The MSE of the estimation is:

$$\begin{aligned}\mathbb{E}\big((\widehat{f}_0 - y_0)^2\big) &\overset{(5)}{=} \mathbb{E}\big((\widehat{f}_0 - f_0 - \varepsilon_0)^2\big) = \mathbb{E}\big((\widehat{f}_0 - f_0)^2 + \varepsilon_0^2 - 2\,\varepsilon_0(\widehat{f}_0 - f_0)\big) \\ &= \mathbb{E}\big((\widehat{f}_0 - f_0)^2\big) + \mathbb{E}(\varepsilon_0^2) - 2\,\mathbb{E}\big(\varepsilon_0(\widehat{f}_0 - f_0)\big) \\ &\overset{(2)}{=} \mathbb{E}\big((\widehat{f}_0 - f_0)^2\big) + \sigma^2 - 2\,\mathbb{E}\big(\varepsilon_0(\widehat{f}_0 - f_0)\big).\end{aligned} \tag{6}$$

- The last term is:

$$\mathbb{E}\big(\varepsilon_0(\widehat{f}_0 - f_0)\big) \overset{(5)}{=} \mathbb{E}\big((y_0 - f_0)(\widehat{f}_0 - f_0)\big). \tag{7}$$

- For calculation of this term, we have two cases: (I) whether the instance $(\boldsymbol{x}_0, y_0)$ is in the training set or (II) not in the training set. In other words, whether the instance was used to train the model (estimator) or not.

Case I: Instance not in the Training Set

- Assume the instance $(\boldsymbol{x}_0, y_0)$ was not in the training set, i.e., it was not used for training the model. In other words, we have $y_0 \notin \mathcal{T}$.
- This means that the estimation $\widehat{f}_0$ is independent of the observation y_0 because the observation was not used to train the model but the estimation is obtained from the model. Therefore:

$$\therefore \; y_0 \perp\!\!\!\perp \widehat{f}_0 \implies (y_0 - f_0) \perp\!\!\!\perp (\widehat{f}_0 - f_0)$$

$$\implies \mathbb{E}\big((y_0 - f_0)(\widehat{f}_0 - f_0)\big) \overset{(a)}{=} \mathbb{E}\big((y_0 - f_0)\big)\,\mathbb{E}\big((\widehat{f}_0 - f_0)\big) \overset{(b)}{=} 0 \times \mathbb{E}\big((\widehat{f}_0 - f_0)\big) = 0,$$

where (a) is because $(y_0 - f_0) \perp\!\!\!\perp (\widehat{f}_0 - f_0)$ and (b) is because:

$$\mathbb{E}\big((y_0 - f_0)\big) = \mathbb{E}(y_0) - \mathbb{E}(f_0) \overset{(c)}{=} f_0 - f_0 = 0,$$

where (c) is because of Eq. (3) and:

$$y_0 = f_0 + \varepsilon_0 \implies \mathbb{E}(y_0) \overset{(5)}{=} \mathbb{E}(f_0) + \mathbb{E}(\varepsilon_0) = f_0 + 0 = f_0.$$

Therefore, in this case, the last term in Eq. (6) is zero. Thus:

$$\text{MSE} \rightarrow \boxed{\mathbb{E}\big((\widehat{f}_0 - y_0)^2\big) = \mathbb{E}\big((\widehat{f}_0 - f_0)^2\big) + \sigma^2} \tag{8}$$

Case I: Instance not in the Training Set

- We found:

$$\mathbb{E}((\widehat{f}_0 - y_0)^2) = \mathbb{E}((\widehat{f}_0 - f_0)^2) + \sigma^2$$

- Suppose the number of instances which are not in the training set is m. By Monte Carlo approximation of the expectation terms, we have:

$$\frac{1}{m}\sum_{i=1}^{m}(\widehat{f}_i - y_i)^2 = \frac{1}{m}\sum_{i=1}^{m}(\widehat{f}_i - f_i)^2 + \sigma^2 \xrightarrow{\times m} \sum_{i=1}^{m}(\widehat{f}_i - y_i)^2 = \sum_{i=1}^{m}(\widehat{f}_i - f_i)^2 + m\sigma^2. \quad (9)$$

- The term $\sum_{i=1}^{m}(\widehat{f}_i - y_i)^2$ is the error between the predicted output and the label in the dataset. So, it is the **empirical error**, denoted by **err**.
- The term $\sum_{i=1}^{m}(\widehat{f}_i - f_i)^2$ is the error between the predicted output and true unknown label. This error is referred to as **true error**, denoted by **Err**. Therefore:

$$\mathbf{err} = \mathbf{Err} + m\sigma^2 \Longrightarrow \mathbf{Err} = \mathbf{err} - m\sigma^2. \quad (10)$$

- The term $m\sigma^2$ is a constant and can be ignored. Hence, in this case, the empirical error is a good estimation of the true error. Thus, we can minimize the empirical error in order to properly minimize the true error.

Case II: Instance in the Training Set

- In case 2, the instance is in the training set. For this case, we need to use a mathematical formula named SURE, introduced in the following.
- Consider a multivariate random variable $\mathbb{R}^d \ni \boldsymbol{z} = [z_1, \ldots, z_d]^\top$ whose components are independent random variables with normal distribution, i.e., $z_i \sim \mathcal{N}(\mu_i, \sigma)$.
- Take $\mathbb{R}^d \ni \boldsymbol{\mu} = [\mu_1, \ldots, \mu_d]^\top$ and let $\mathbb{R}^d \ni \boldsymbol{g}(\boldsymbol{z}) = [g_1, \ldots, g_d]^\top$ be a function of the random variable $\boldsymbol{z}$ with $\boldsymbol{g}(\boldsymbol{z}) : \mathbb{R}^d \to \mathbb{R}^d$.
- There exists a lemma, named **Stein's Lemma**, which states:

$$\mathbb{E}\big((\boldsymbol{z} - \boldsymbol{\mu})^\top \boldsymbol{g}(\boldsymbol{z})\big) = \sigma^2 \sum_{i=1}^{d} \mathbb{E}\big(\frac{\partial g_i}{\partial z_i}\big), \tag{11}$$

  which is used in **Stein's Unbiased Risk Estimate (SURE)** [1]. See our tutorial [2] for the proof of Eq. (11).
- If the random variable is a univariate variable, the Stein's lemma becomes:

$$\mathbb{E}\big((z - \mu)\, g(z)\big) = \sigma^2\, \mathbb{E}\big(\frac{\partial g(z)}{\partial z}\big). \tag{12}$$

Case II: Instance in the Training Set

$y_0 = f_0 + \varepsilon_0$

- SURE for univariate variable:

$$\mathbb{E}\big((z-\mu)\,g(z)\big) = \sigma^2\,\mathbb{E}\Big(\frac{\partial g(z)}{\partial z}\Big).$$

- In the SURE formula for univariate variable, we take ε_0, 0, and $\widehat{f}_0 - f_0$ as the z, μ, and $g(z)$, respectively. We do this to make Eq. (7).
- Using Eq. (12), the last term in Eq. (6) is:

$$\mathbb{E}\big((\varepsilon_0 - 0)(\widehat{f}_0 - f_0)\big) = \sigma^2\,\mathbb{E}\Big(\frac{\partial(\widehat{f}_0 - f_0)}{\partial \varepsilon_0}\Big) = \sigma^2\,\mathbb{E}\Big(\frac{\partial \widehat{f}_0}{\partial \varepsilon_0} - \frac{\partial f_0}{\partial \varepsilon_0}\Big) \overset{(a)}{=} \sigma^2\,\mathbb{E}\Big(\frac{\partial \widehat{f}_0}{\partial \varepsilon_0}\Big)$$

$$\overset{(b)}{=} \sigma^2\,\mathbb{E}\Big(\frac{\partial \widehat{f}_0}{\partial y_0} \times \frac{\partial y_0}{\partial \varepsilon_0}\Big) \overset{(c)}{=} \sigma^2\,\mathbb{E}\Big(\frac{\partial \widehat{f}_0}{\partial y_0}\Big),$$

3rd term in MSE

where (a) is because the true model f is not dependent on the noise, (b) is because of the chain rule in derivative, and (c) is because:

$$y_0 \overset{(5)}{=} f_0 + \varepsilon_0 \Longrightarrow \frac{\partial y_0}{\partial \varepsilon_0} = 1.$$

- Therefore, in this case, the Eq. (6) is:

MSE →

$$\mathbb{E}\big((\widehat{f}_0 - y_0)^2\big) = \mathbb{E}\big((\widehat{f}_0 - f_0)^2\big) + \sigma^2 - 2\sigma^2\mathbb{E}\Big(\frac{\partial \widehat{f}_0}{\partial y_0}\Big). \tag{13}$$

Case II: Instance in the Training Set

- We had:

$$\mathbb{E}((\widehat{f}_0 - y_0)^2) = \mathbb{E}((\widehat{f}_0 - f_0)^2) + \sigma^2 - 2\sigma^2 \mathbb{E}(\frac{\partial \widehat{f}_0}{\partial y_0}).$$

- Suppose the number of training instances is n. By Monte Carlo approximation of the expectation terms, we have:

$$\frac{1}{n}\sum_{i=1}^{n}(\widehat{f}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\widehat{f}_i - f_i)^2 + \sigma^2 - 2\sigma^2 \frac{1}{n}\sum_{i=1}^{n}\frac{\partial \widehat{f}_i}{\partial y_i} \quad \Longrightarrow \times n$$

$$\sum_{i=1}^{n}(\widehat{f}_i - y_i)^2 = \sum_{i=1}^{n}(\widehat{f}_i - f_i)^2 + n\sigma^2 - 2\sigma^2 \sum_{i=1}^{n}\frac{\partial \widehat{f}_i}{\partial y_i}. \tag{14}$$

- The term $\sum_{i=1}^{m}(\widehat{f}_i - y_i)^2$ is the error between the predicted output and the label in the dataset. So, it is the **empirical error**, denoted by **err**.
- The term $\sum_{i=1}^{m}(\widehat{f}_i - f_i)^2$ is the error between the predicted output and true unknown label. This error is referred to as **true error**, denoted by **Err**. Therefore:

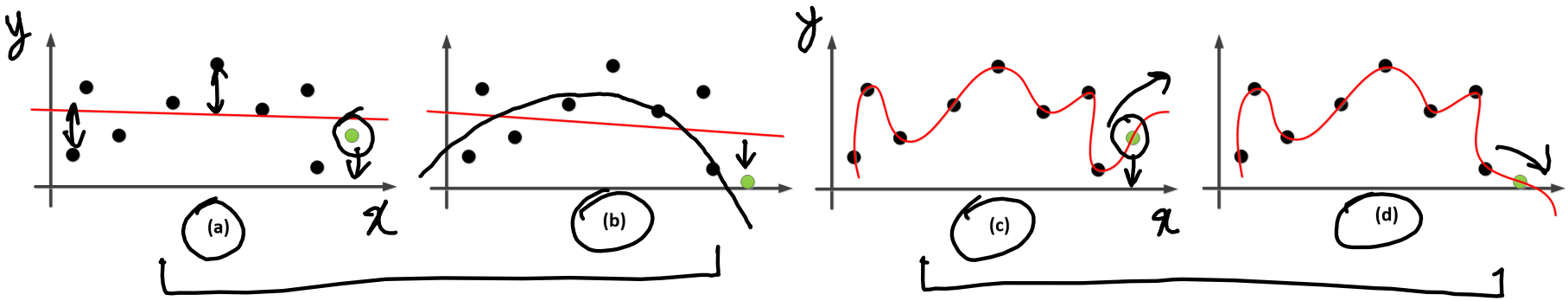
$$\mathbf{err} = \mathbf{Err} + n\sigma^2 - 2\sigma^2 \sum_{i=1}^{n}\frac{\partial \widehat{f}_i}{\partial y_i} \Longrightarrow \mathbf{Err} = \mathbf{err} - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n}\frac{\partial \widehat{f}_i}{\partial y_i}. \tag{15}$$

Case II: Instance in the Training Set

- We had:

$$\textbf{Err} = \textbf{err} - n\,\sigma^2 + 2\,\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i},$$

- The last term in this equation is a measure of **complexity** (or **overfitting**) of the model. Note that $\partial \widehat{f}_i / \partial y_i$ means if we move the i-th training instance, how much the model's estimation of that instance will change? This shows how much the model is complex or overfitted.
- For better understanding, suppose a line regressing a training set via least squares problem. If we change a point, the line will not change significantly because the model is not complex (is **underfitted**). On the other hand, consider a regression model passing through "all" the points. If we move a training point, the regressing curve changes noticeably which is because the model is very complex (**overfitted**).



Case II: Instance in the Training Set

not in training set: $Err = err - n\sigma^2$

- We had:

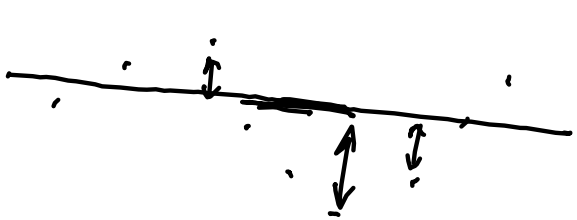
in training set: ★ true error / empirical error

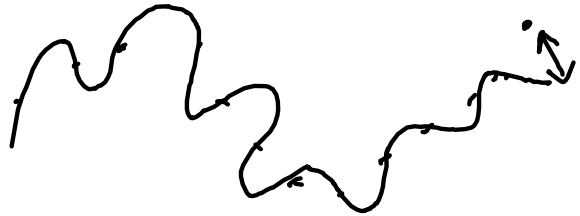
$$Err = err - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i},$$

- According to this equation, in the case where the instance is in the training set, the empirical error is not a good estimation of the true error.
- The reason is that minimization of **err** usually increases the complexity of the model, cancelling out the minimization of **Err** after some level of training.

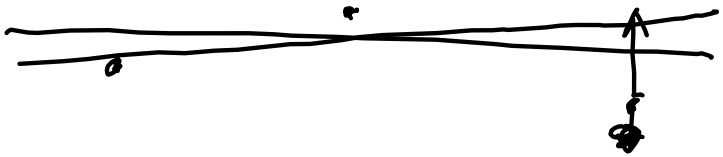
Overfitting, Underfitting, and Generalization

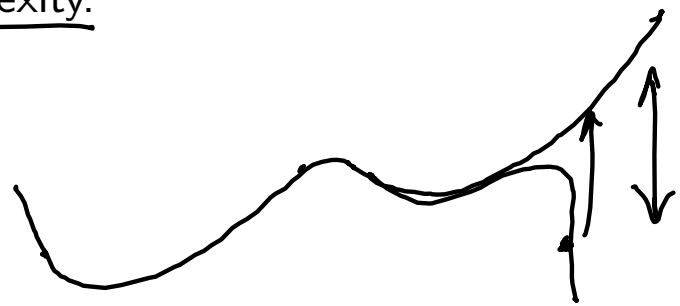
Overfitting, Underfitting, and Generalization





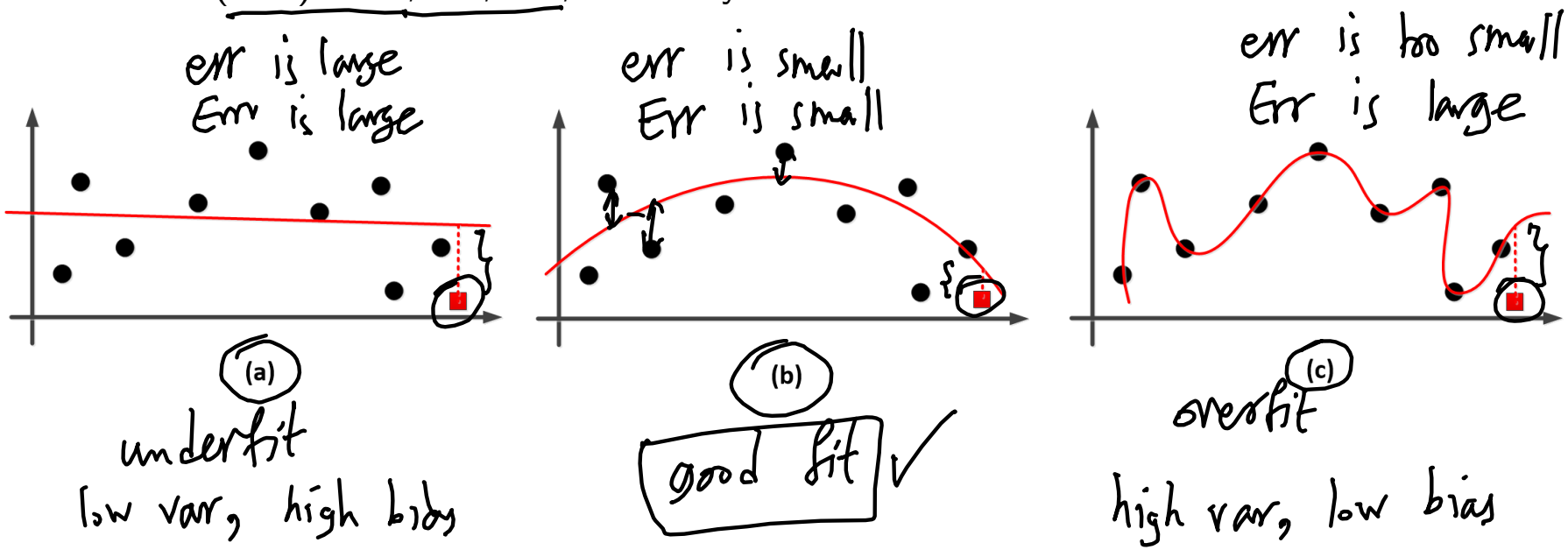
- If the model is trained in an extremely simple way so that its estimation has low variance but high bias, we have **underfitting**. Note that underfitting is also referred to as **over-generalization**.
- On the other hand, if the model is trained in an extremely complex way so that its estimation has high variance but low bias, we have **overfitting**.
- To summarize:
  - in underfitting: low variance, high bias, and low complexity.
  - in overfitting: high variance, low bias, high complexity.





Overfitting, Underfitting, and Generalization

- An example for underfitting, good fit, and overfitting is illustrated in this figure.
- As this figure shows, in both underfitting and overfitting, the estimation of a test instance might be very weak while in a good fit, the test instance, which was not seen in the training phase, is estimated well enough with smaller error.
- The ability of the model to estimate the unseen test (out-of-sample) data is referred to as **generalization**.
- The lack of generalization is the reason why both overfitting and underfitting, especially overfitting, is not acceptable. In overfitting, the training error, i.e., **err**, is very small while the test (true) error, i.e., **Err**, is usually awful!



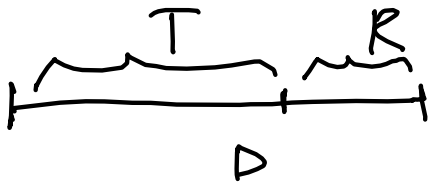
Cross Validation

Cross Validation

- In order to either (I) find out until which complexity we should train the model or (II) tune the parameters of the model, we should use **cross validation** [3].
- In cross validation, we divide the dataset $\mathcal{D}$ into two partitions, i.e., **training set** denoted by $\mathcal{T}$ and **test set** denoted by $\mathcal{R}$ where the union of these two subsets is the whole dataset and the intersection of them is the empty set:

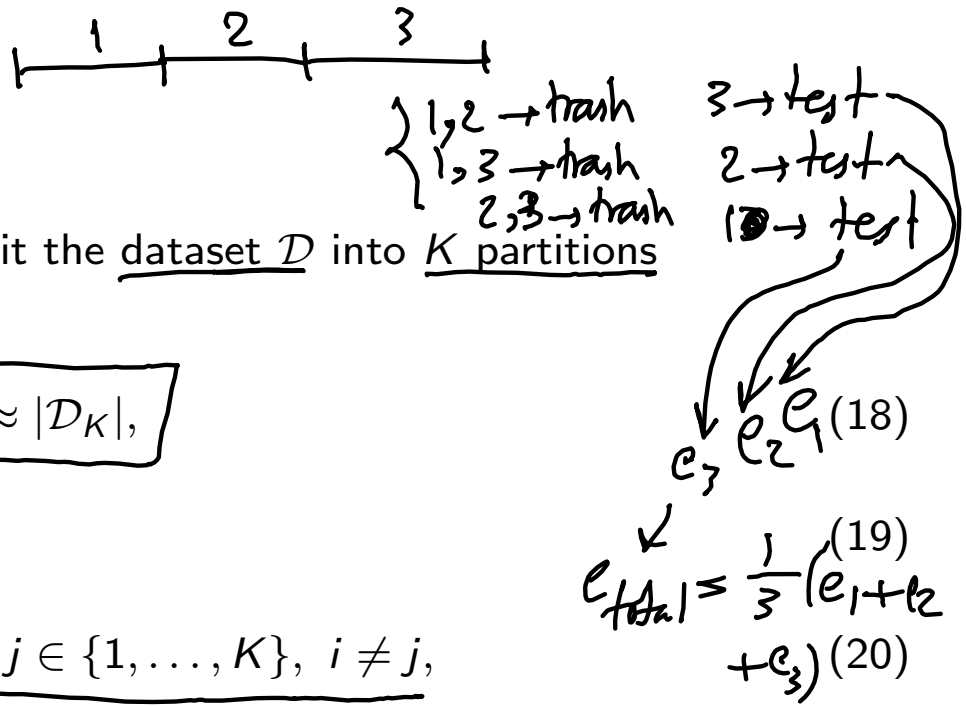
$$\mathcal{T} \cup \mathcal{R} = \mathcal{D}, \tag{16}$$

$$\mathcal{T} \cap \mathcal{R} = \varnothing. \tag{17}$$



- The $\mathcal{T}$ is used for training the model. After the model is trained, the $\mathcal{R}$ is used for testing the performance of the model.
- We have different methods for cross validation. Two of the most well-known methods for cross validation are K-**fold cross validation** and **Leave-One-Out Cross Validation (LOOCV)**.

K-fold Cross Validation



- In K-**fold cross validation**, we randomly split the dataset $\mathcal{D}$ into K partitions $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ where:

$$|\mathcal{D}_1| \approx |\mathcal{D}_2| \approx \cdots \approx |\mathcal{D}_K|,$$

$$\bigcup_{i=1}^{K} \mathcal{D}_i = \mathcal{D},$$

$$\mathcal{D}_i \cap \mathcal{D}_j = \varnothing, \quad \forall i, j \in \{1, \ldots, K\},\ i \neq j,$$

  where $|.|$ denoted the cardinality of set.
- Sometimes, the dataset $\mathcal{D}$ is shuffled before the cross validation for better randomization.
- Moreover, both simple random sampling without replacement and stratified sampling [4, 5] can be used for this splitting.
- The K-fold cross validation includes K iterations, where in each of them, one of the partitions is used as the test set and the rest of data is used for training. The overall estimation error is the average test error of iterations.
- We usually have $K = 2, 5, 10$ in the literature but $K = 10$ is the most common.

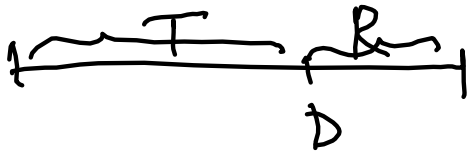
K-fold Cross Validation

- The algorithm of K-fold cross validation is shown in the following.

1 Randomly split $\mathcal{D}$ into K partitions with almost equal sizes.
2 **for** k *from* 1 *to* K **do**
3 $\quad \mathcal{R} \leftarrow$ Partition k from $\mathcal{D}$.
4 $\quad \mathcal{T} \leftarrow \mathcal{D} \setminus \mathcal{R}$.
5 $\quad$ Use $\mathcal{T}$ to train the model.
6 $\quad \mathbf{Err}_k \leftarrow$ Use the trained model to predict $\mathcal{R}$.
7 $\mathbf{Err} \leftarrow \frac{1}{K} \sum_{k=1}^{K} \mathbf{Err}_k$

Algorithm 1: K-fold Cross Validation

Leave-One-Out Cross Validation



- In **Leave-One-Out Cross Validation (LOOCV)**, we iterate for $|\mathcal{D}| = N$ times and in each iteration, we take one instance as the $\mathcal{R}$ (so that $|\mathcal{R}| = 1$) and the rest of instances as the training set.
- The overall estimation error is the average test error of iterations.
- Usually, when the size of dataset is small, LOOCV is used in order to use the most of dataset for training and then test the model properly.
- The algorithm of LOOCV is shown in the following.

1 **for** k *from* 1 *to* $|\mathcal{D}| = N$ **do**
2 $\quad \mathcal{R} \leftarrow$ Take the k-th instance from $\mathcal{D}$.
3 $\quad \mathcal{T} \leftarrow \mathcal{D} \setminus \mathcal{R}$.
4 $\quad$ Use $\mathcal{T}$ to train the model.
5 $\quad \mathbf{Err}_k \leftarrow$ Use the trained model to predict $\mathcal{R}$.
6 $\mathbf{Err} \leftarrow \frac{1}{|\mathcal{D}|} \sum_{k=1}^{|\mathcal{D}|} \mathbf{Err}_k$

Algorithm 2: Leave-One-Out Cross Validation

Cheating #1 in Machine Learning

- The test set and the training set should be disjoint, i.e., $\mathcal{T} \cap \mathcal{R} = \varnothing$; otherwise, we are introducing the whole or a part of the test instances to the model to learn them.
- Of course, in that way, the model will learn to estimate the test instances easier and better; however, in the real-world applications, the test data is not available at the time of training. Therefore, if we mistakenly have $\mathcal{T} \cap \mathcal{R} \neq \varnothing$, it is referred to as **cheating** in machine learning (we call it **cheating #1** here).

Validation Set

$\alpha \rightarrow \{0, 0.5, 1\}$
$\beta \rightarrow \{100, 200\}$
$\longrightarrow$ for $\alpha = [0, 0.5, 1]$:
train model ($\mathcal{T}$)
eval model ($\mathcal{V}$)

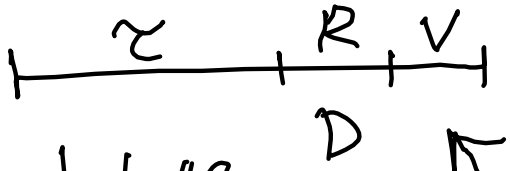
for $\alpha = [0, 0.5, 0]$:
for $\beta = [100, 200]$:
train model ($\mathcal{T}$)
eval model ($\mathcal{V}$)

- In some cases, the model has some parameters which need to be determined. In this case, we split the data $\mathcal{D}$ to three subsets, i.e., training set $\mathcal{T}$, test set $\mathcal{R}$, and **validation set** $\mathcal{V}$.
- Usually, we have $|\mathcal{T}| > |\mathcal{R}|$ and $|\mathcal{T}| > |\mathcal{V}|$.
- First, we want to find the best parameters. For this, the training set is used to train the model with different values of parameters. For every value of parameter(s), after the model is trained, it is tested on the validation set. This is performed for all desired values of parameters. The parameter value resulting in the best estimation performance on the validation set is selected to be the value of parameter(s).
- After finding the values of parameters, the model is trained using the training set (where the found parameter value is used). Then, the model is tested on the test set and the estimation performance is the average test set over the cross validation iterations.

choose α, β with best performance on $\mathcal{V} \Longrightarrow \alpha_{Best}, \beta_{best}$

train model ($\mathcal{T}$; $\alpha_{best}, \beta_{best}$)
eval model ($\mathcal{R}$) $\longrightarrow$ report this

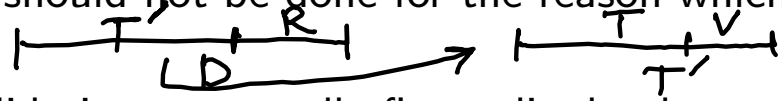
Cheating #2 in Machine Learning



- In cross validation with validation set, we have:

$$\mathcal{T} \cap \mathcal{R} = \varnothing, \quad \mathcal{T} \cap \mathcal{V} = \varnothing, \quad \mathcal{V} \cap \mathcal{R} = \varnothing. \tag{21}$$

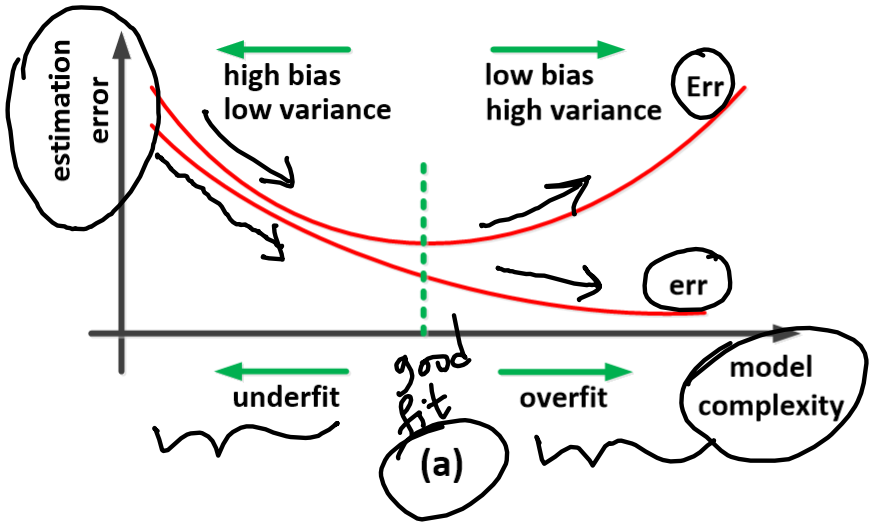
- The validation and test sets should be disjoint because the parameters of the model should not be optimized by testing on the test set. In other words, in real-world applications, the training and validation sets are available but the test set is not available yet. If we mistakenly have $\mathcal{V} \cap \mathcal{R} \neq \varnothing$, it is referred to as **cheating** in machine learning (we call it **cheating #2** here).
- This kind of mistake is very common in the literature unfortunately, where some people optimize the parameters by testing on the test set without having a validation set.
- Moreover, the training and test sets should be disjoint as explained beforehand; otherwise, that would be another kind of **cheating** in machine learning (introduced before as **cheating #1**).
- On the other hand, the training and validation sets should be disjoint. Although having $\mathcal{T} \cap \mathcal{V} \neq \varnothing$ is not cheating but it should not be done for the reason which will be explained later in this section.

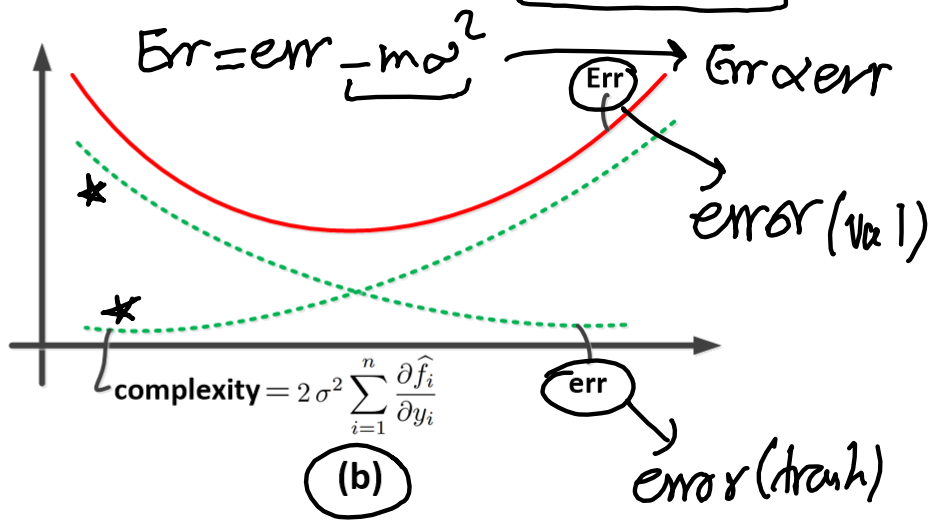


- To have validation set in cross validation, we usually first split the dataset $\mathcal{D}$ into $\mathcal{T}'$ and $\mathcal{R}$ where $\mathcal{T}' \cup \mathcal{R} = \mathcal{D}$ and $\mathcal{T}' \cap \mathcal{R} = \varnothing$. Then, we split the set $\mathcal{T}'$ into the training and validation sets, i.e., $\mathcal{T} \cup \mathcal{V} = \mathcal{T}'$ and $\mathcal{T} \cap \mathcal{V} = \varnothing$ and usually $|\mathcal{T}| > |\mathcal{V}|$.
- The algorithms of K-fold cross validation and LOOCV can be modified accordingly to include the validation set. In LOOCV, we usually have $|\mathcal{V}| = 1$.

Justification of Overfitting

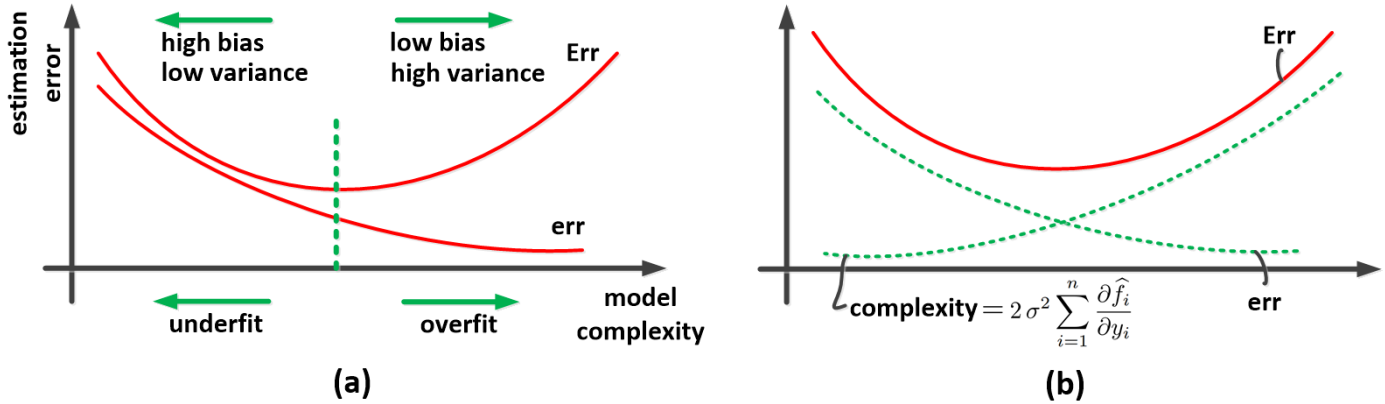
Justification of Overfitting





- When the instance is not in the training set, the true error, **Err**, and the test error, **err**, behave differently as shown in Fig. (a).
- At the first stages of training, the **err** and **Err** both decrease; however, after some training, the model becomes more complex and goes toward overfitting. In that stage, the **Err** starts to increase.
- We should end the training when the **Err** starts to increase because that stage is the good fit. Usually, in order to find out when to stop training, we train the model for one stage (e.g., iteration) and then test the trained model on the validation set where the error is named **Err**. This is commonly used in training neural networks [6] where **Err** is measured after every epoch for example.

Justification of Overfitting



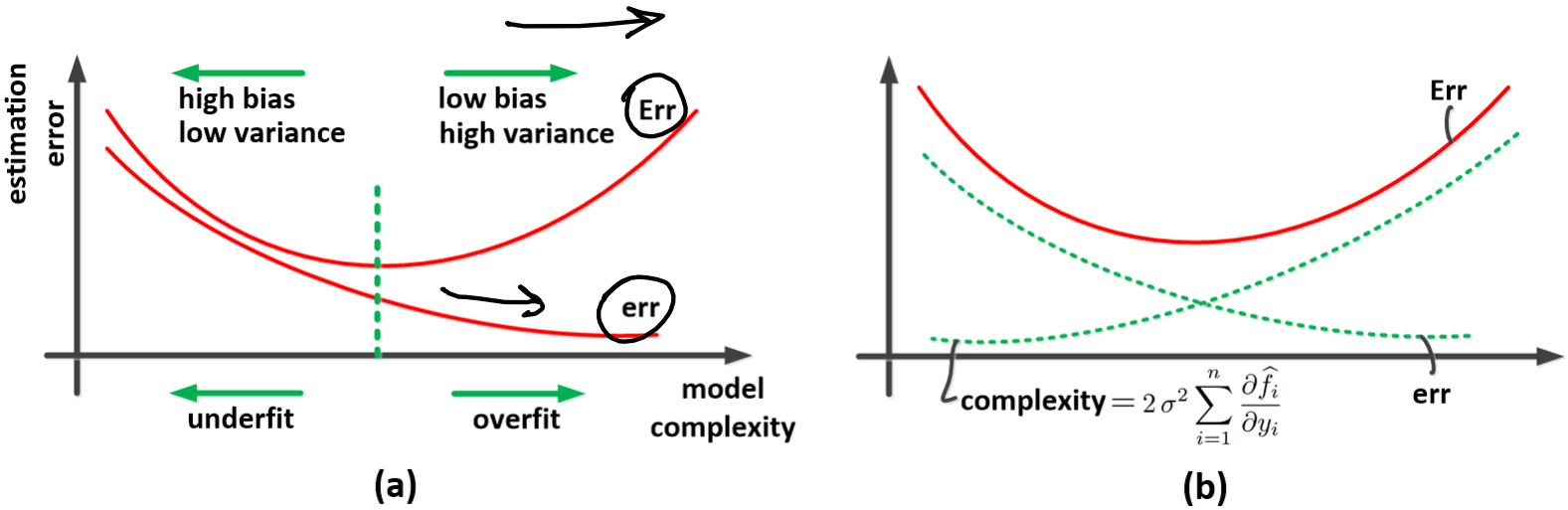
- Recall the Eqs. (10) and (15) where the true error for the test (not in training set) and training instance are related to the training error, respectively:

$$\textbf{Err} = \textbf{err} - m\sigma^2, \quad \text{(instance in training set)}$$

$$\textbf{Err} = \textbf{err} - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{f}_i}{\partial y_i}. \quad \text{(instance not in training set)}$$

- The reason why **Err** increases after a while of training is according to Eq. (15). Dropping the constant $n\sigma^2$ from that expression, we have: $\textbf{Err} = \textbf{err} + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{f}_i}{\partial y_i}$ where the term $2\sigma^2 \sum_{i=1}^{n} \frac{\partial \hat{f}_i}{\partial y_i}$ shows the model complexity. See Fig. (b) where both **err** and the model complexity are illustrated as a function of training stages (iterations). According to Eq. (15), the **Err** is the summation of these two curves which clarifies the reason of its behavior.

Justification of Overfitting



- That is why we should not train a lot on the training set because the model will get too much fitted on the training set and will lose its ability to generalize to new unseen data.
- The Fig. (a) and Eq. (15) show that it is better to have $\mathcal{T} \cap \mathcal{V} = \emptyset$. Otherwise, for example if we have $\mathcal{T} = \mathcal{V}$, the **Err** will be equivalent to **err** and thus it will go down even in overfitting stages. This is harmful to our training because we will not notice overfitting properly.
- The Eq. (10) also explains that the error on validation or test set is a good measure for the true error. That is why we can use test or validation error in order to know until what stage we can train the model without overfitting.

Discussion of Cheating in a Nut Shell

- If we have only training and test sets without validation set:
  - $\mathcal{T} \cap \mathcal{R} \neq \varnothing \implies$ cheating #1
- If we have training, test, and validation sets ~~without validation set~~:
  - $\mathcal{T} \cap \mathcal{R} \neq \varnothing \implies$ cheating #1
  - $\mathcal{V} \cap \mathcal{R} \neq \varnothing \implies$ cheating #2
  - $\mathcal{T} \cap \mathcal{V} \neq \varnothing \implies$ harmful to training (not noticing overfitting properly)
- The first two items are advantageous to the model's performance on test data but that is cheating and also it may be disadvantageous to future new test data.
- The third item is disadvantageous to the model's performance on test data because we may not find out overfitting or we may find it out late and the generalization error will become worse; therefore, it is better not to do it.

Regularization

Regularization: Definition

- We can minimize the true error, **Err**, using optimization. According to Eq. (15), we have:

$$\text{minimize} \quad \mathbf{err} - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i}. \tag{22}$$

- As the term $n\sigma^2$ is a constant, we can drop it.
- Calculation of $\partial \widehat{f}_i / \partial y_i$ is usually very difficult; therefore, we usually use a penalty term in place of it where the penalty increases as the complexity of the model increases in order to imitate the behavior of $\partial \widehat{f}_i / \partial y_i$.
- Therefore, the optimization can be written as a **regularized optimization** problem:

$$\underset{x}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x}; \theta) := J(\boldsymbol{x}; \theta) + \alpha \Omega(\boldsymbol{x}), \tag{23}$$

where θ is the parameter(s) of the cost function, $J(.)$ is the objective **err** to be minimized, $\Omega(.)$ is the penalty function representing the complexity of model, $\alpha > 0$ is the regularization parameter, and $\widetilde{J}(.)$ is the **regularized objective function**.

Regularization: Definition

$$\begin{cases} \text{minimize} & \mathbf{Err} := \mathbf{err} - n\sigma^2 + 2\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i}, \\ \text{minimize}_{\boldsymbol{x}} & \widetilde{J}(\boldsymbol{x};\theta) := J(\boldsymbol{x};\theta) + \alpha\,\Omega(\boldsymbol{x}). \end{cases}$$

- The penalty function can be different things such as ℓ_2 norm [7], ℓ_1 norm [8, 9], $\ell_{2,1}$ norm [10], etc.
- The ℓ_1 and $\ell_{2,1}$ norms are useful for having sparsity [11, 12]. The sparsity is very effective because of the **"bet on sparsity"** principal: "Use a procedure that does well in sparse problems, since no procedure does well in dense problems [7, 13]."
- The effectiveness of the sparsity can also be explained by **Occam's razor** [14] stating that "simpler solutions are more likely to be correct than complex ones" or "simplicity is a goal in itself".
- We are minimizing the **Err** (i.e., $\widetilde{J}(\boldsymbol{x};\theta)$) and not **err** (i.e., $J(\boldsymbol{x};\theta)$). As discussed before, minimizing **err** results in overfitting. Therefore, **regularization helps avoid overfitting**.

ℓ_2 Regularization

Theory for ℓ_2 Norm Regularization

- The ℓ_2 norm regularization [7]:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x};\theta) := J(\boldsymbol{x};\theta) + \frac{\alpha}{2}||\boldsymbol{x}||_2^2. \tag{24}$$

- The ℓ_2 norm regularization is also referred to as **ridge regression** or **Tikhonov regularization** [6].
- Suppose $\boldsymbol{x}^*$ is minimizer of the $J(\boldsymbol{x};\theta)$, i.e.:

$$\nabla J(\boldsymbol{x}^*;\theta) = 0. \tag{25}$$

The Taylor series expansion of $J(\boldsymbol{x};\theta)$ up to the second derivative at $\boldsymbol{x}^*$ gives:

$$\begin{aligned}\widehat{J}(\boldsymbol{x};\theta) &\approx J(\boldsymbol{x}^*;\theta) + \nabla J(\boldsymbol{x}^*;\theta) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^*)^\top \boldsymbol{H}(\boldsymbol{x}-\boldsymbol{x}^*) \\ &= J(\boldsymbol{x}^*;\theta) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^*)^\top \boldsymbol{H}(\boldsymbol{x}-\boldsymbol{x}^*),\end{aligned} \tag{26}$$

where $\boldsymbol{H} \in \mathbb{R}^{d\times d}$ is the Hessian.

Theory for ℓ_2 Norm Regularization

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x};\theta) := J(\boldsymbol{x};\theta) + \frac{\alpha}{2}\,||\boldsymbol{x}||_2^2,$$

$$\widehat{J}(\boldsymbol{x};\theta) \approx J(\boldsymbol{x}^*;\theta) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^*)^\top \boldsymbol{H}(\boldsymbol{x}-\boldsymbol{x}^*).$$

- Using the Taylor approximation in the cost gives us [6]:

$$\widetilde{J}(\boldsymbol{x};\theta) = \widehat{J}(\boldsymbol{x};\theta) + \frac{\alpha}{2}\,||\boldsymbol{x}||_2^2 = J(\boldsymbol{x}^*;\theta) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^*)^\top \boldsymbol{H}(\boldsymbol{x}-\boldsymbol{x}^*) + \frac{\alpha}{2}\,||\boldsymbol{x}||_2^2,$$

$$\frac{\partial \widetilde{J}(\boldsymbol{x};\theta)}{\partial \boldsymbol{x}} = \boldsymbol{0} + \boldsymbol{H}(\boldsymbol{x}^\dagger - \boldsymbol{x}^*) + \alpha\,\boldsymbol{x}^\dagger \overset{\text{set}}{=} \boldsymbol{0} \implies (\boldsymbol{H}+\alpha\boldsymbol{I})\,\boldsymbol{x}^\dagger = \boldsymbol{H}\boldsymbol{x}^*$$

$$\implies \boldsymbol{x}^\dagger = (\boldsymbol{H}+\alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{x}^*, \qquad \boldsymbol{x}^\dagger = \boldsymbol{H}^{-1}\boldsymbol{H}\boldsymbol{x}^* = \boldsymbol{x}^* \tag{27}$$

where $\boldsymbol{x}^\dagger$ is the minimizer of $\widetilde{J}(\boldsymbol{x};\theta)$.

- $\partial J(\boldsymbol{x}^*;\theta)/\partial \boldsymbol{x} = \boldsymbol{0}$ because the $J(\boldsymbol{x}^*;\theta)$ is a constant vector with respect to $\boldsymbol{x}$.
- The Eq. (27) makes sense because if $\alpha = 0$, which means we do not have the regularization term, we will have $\boldsymbol{x}^\dagger = \boldsymbol{x}^*$. This means that the minimizer of $\widetilde{J}(\boldsymbol{x},\theta)$ will be the same as the minimizer of $J(\boldsymbol{x};\theta)$ which is correct according to Eq. (24) where $\alpha = 0$.

Theory for ℓ_2 Norm Regularization

- If we apply eigenvalue decomposition on the Hessian matrix, we will have:

$$\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top. \tag{28}$$

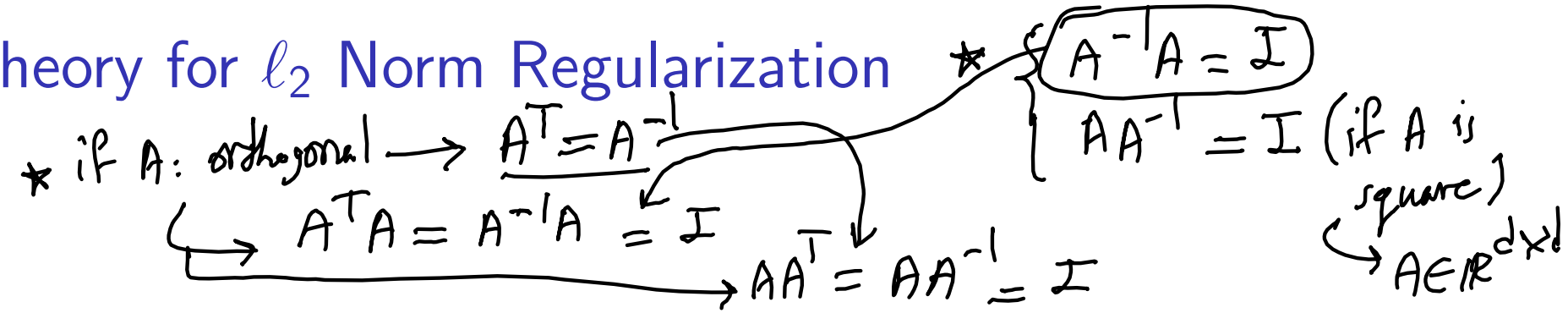
- Using this decomposition in Eq. (27), $\boldsymbol{x}^\dagger = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{x}^*$, gives us:

$$\begin{aligned}
\boldsymbol{x}^\dagger &= (\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top + \alpha \boldsymbol{I})^{-1}\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{x}^* \overset{(a)}{=} (\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top + \boldsymbol{U}\boldsymbol{U}^\top \alpha \boldsymbol{I})^{-1}\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{x}^* \\
&\overset{(b)}{=} (\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top + \boldsymbol{U}\alpha \boldsymbol{I}\boldsymbol{U}^\top)^{-1}\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{x}^* = (\boldsymbol{U}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})\boldsymbol{U}^\top)^{-1}\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{x}^* \\
&\overset{(c)}{=} \boldsymbol{U}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{U}^{-1}\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{x}^* = \boldsymbol{U}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{x}^*,
\end{aligned} \tag{29}$$

where (a) and (c) are because $\boldsymbol{U}$ is an orthogonal matrix so we have $\boldsymbol{U}^{-1} = \boldsymbol{U}^\top$ which yields to $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$ and $\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}$ (because $\boldsymbol{U}$ is not truncated). The (b) is because α is a scalar and can move between the multiplication of matrices.

- The Eq. (29) means that we are rotating $\boldsymbol{x}^*$ by $\boldsymbol{U}^\top \boldsymbol{x}^*$ but before rotating it back with $\boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}^*$, we manipulate it with the term $(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}$.

Theory for ℓ_2 Norm Regularization



Based on Eq. (29), $\boldsymbol{x}^\dagger = \boldsymbol{U}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{U}^\top \boldsymbol{x}^*$, we can have the following interpretations:

- If $\alpha = 0$, we have:

$$\boldsymbol{x}^\dagger = \boldsymbol{U}\underbrace{\boldsymbol{\Lambda}^{-1}\boldsymbol{\Lambda}}_{\boldsymbol{I}}\boldsymbol{U}^\top \boldsymbol{x}^* = \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}^* \overset{(a)}{=} \underbrace{\boldsymbol{U}\boldsymbol{U}^{-1}}_{\boldsymbol{I}}\boldsymbol{x}^* = \boldsymbol{x}^*,$$

where (a) is because $\boldsymbol{U}$ is an orthogonal matrix and (b) is because $\boldsymbol{U}$ is a non-truncated orthogonal matrix. This means that if we do not have the penalty term, the minimizer of $\widetilde{J}(\boldsymbol{x}; \theta)$ is the minimizer of $J(\boldsymbol{x}; \theta)$ as expected. In other words, we are rotating the solution $\boldsymbol{x}^*$ by $\boldsymbol{U}^\top$ and then rotate it back by $\boldsymbol{U}$.

Theory for ℓ_2 Norm Regularization

- If $\alpha \neq 0$, the term $(\mathbf{\Lambda} + \alpha \boldsymbol{I})^{-1}\mathbf{\Lambda}$ is:

$$(\mathbf{\Lambda} + \alpha \boldsymbol{I})^{-1}\mathbf{\Lambda} = \begin{bmatrix} \frac{\lambda_1}{\lambda_1+\alpha} & 0 & \dots & 0 \\ 0 & \frac{\lambda_2}{\lambda_2+\alpha} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\lambda_d}{\lambda_d+\alpha} \end{bmatrix},$$

where $\mathbf{\Lambda} = \mathbf{diag}([\lambda_1, \dots, \lambda_d]^\top)$. Therefore, for the j-th direction of Hessian, we have $\frac{\lambda_j}{\lambda_j+\alpha}$.

  - If $\lambda_j \gg \alpha$, we will have $\frac{\lambda_j}{\lambda_j+\alpha} \approx 1$ so for the j-th direction we have $(\mathbf{\Lambda} + \alpha \boldsymbol{I})^{-1}\mathbf{\Lambda} \approx \boldsymbol{I}$; therefore, $\boldsymbol{x}^\dagger \approx \boldsymbol{x}^*$. This makes sense because $\lambda_j \gg \alpha$ means that the j-th direction of Hessian and thus the j-th direction of $J(\boldsymbol{x}; \theta)$ is large enough to be effective. Therefore, the penalty is roughly ignored with respect to it.
  - If $\lambda_j \ll \alpha$, we will have $\frac{\lambda_j}{\lambda_j+\alpha} \approx 0$ so for the j-th direction we have $(\mathbf{\Lambda} + \alpha \boldsymbol{I})^{-1}\mathbf{\Lambda} \approx \boldsymbol{0}$; therefore, $\boldsymbol{x}^\dagger \approx \boldsymbol{0}$. This makes sense because $\lambda_j \ll \alpha$ means that the j-th direction of Hessian and thus the j-th direction of $J(\boldsymbol{x}; \theta)$ is small and not effective. Therefore, the penalty shrinks that direction to almost zero.

Theory for ℓ_2 Norm Regularization

- Therefore, the ℓ_2 norm regularization keeps the effective directions but shrinks the weak directions to close to zero.
- The following measure is referred to as **effective number of parameters** or **degree of freedom** [7]:

$$\sum_{j=1}^{d} \frac{\lambda_j}{\lambda_j + \alpha}, \tag{30}$$

because it counts the number of effective directions as discussed above. Moreover, the term $\lambda_j/(\lambda_j + \alpha)$ or $(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}$ is called the **shrinkage factor** because it shrinks the weak directions.

ℓ_1 Regularization

Theory for ℓ_1 Norm Regularization

- As explained before, sparsity is very useful and effective. If $\boldsymbol{x} = [x_1, \ldots, x_d]^\top$, for having sparsity, we should use **subset selection** for the regularization:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x}; \theta) := J(\boldsymbol{x}; \theta) + \alpha \, ||\boldsymbol{x}||_0, \tag{31}$$

where:

$$||\boldsymbol{x}||_0 := \sum_{j=1}^{d} \mathbb{I}(x_j \neq 0) = \begin{cases} 0 & \text{if } x_j = 0, \\ 1 & \text{if } x_j \neq 0, \end{cases} \tag{32}$$

is "ℓ_0" norm, which is not a norm (so we used "." for it) because it does not satisfy the norm properties [15]. The "ℓ_0" norm counts the number of non-zero elements so when we penalize it, it means that we want to have sparser solutions with many zero entries.

- According to [16], the convex relaxation of "ℓ_0" norm (subset selection) is ℓ_1 norm. Therefore, we write the regularized optimization as:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x}; \theta) := J(\boldsymbol{x}; \theta) + \alpha \, ||\boldsymbol{x}||_1. \tag{33}$$

- The ℓ_1 regularization is also referred to as **lasso (least absolute shrinkage and selection operator)** regularization [8].

Theory for ℓ_1 Norm Regularization

- Different methods exist for solving optimization having ℓ_1 norm, such as proximal algorithm using **soft thresholding** [17] and **coordinate descent** [18, 19]. Here, we explain solving the optimization using the coordinate descent algorithm.
- The idea of coordinate descent algorithm is similar to the idea of Gibbs sampling [20] where we work on the dimensions of the variable one by one. Similar to what we did for obtaining Eq. (27), we have:

$$\widetilde{J}(\boldsymbol{x};\theta) = \widehat{J}(\boldsymbol{x};\theta) + \alpha\,||\boldsymbol{x}||_1 = J(\boldsymbol{x}^*;\theta) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}^*)^\top \boldsymbol{H}(\boldsymbol{x}-\boldsymbol{x}^*) + \alpha\,||\boldsymbol{x}||_1.$$

- For simplicity in deriving an interpretable expression, we assume that the Hessian matrix is diagonal [6]. For coordinate descent, we look at the j-th coordinate (dimension):

$$\widetilde{J}(x_j;\theta) = \widehat{J}(x_j;\theta) + \alpha\,|x_j| = J(x_j^*;\theta) + \frac{1}{2}(x_j - x_j^*)^2 h_j + \alpha\,|x_j| + c,$$

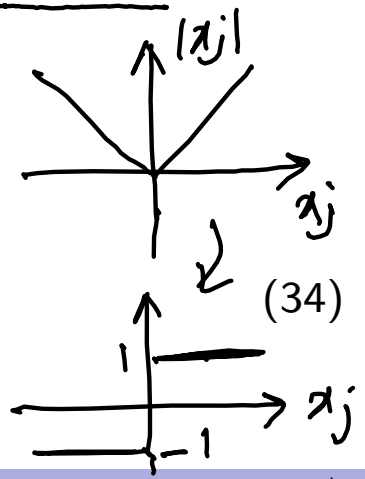
where $\boldsymbol{x} = [x_1, \ldots, x_d]^\top$, $\boldsymbol{x}^* = [x_1^*, \ldots, x_d^*]^\top$, h_j is the (j,j)-th element of the diagonal Hessian matrix, and c is a constant term with respect to x_j (not dependent to x_j).

- Taking derivative with respect to x_j gives us:

$$\frac{\partial \widetilde{J}(x_j;\theta)}{\partial x_j} = 0 + (x_j - x_j^*)\,h_j + \alpha\,\textbf{sign}(x_j) \overset{\text{set}}{=} 0 \implies$$

$$x_j^\dagger = x_j^* - \frac{\alpha}{h_j}\,\textbf{sign}(x_j) = \begin{cases} x_j^* - \frac{\alpha}{h_j} & \text{if } x_j > 0, \\ x_j^* + \frac{\alpha}{h_j} & \text{if } x_j < 0, \end{cases} \quad (34)$$

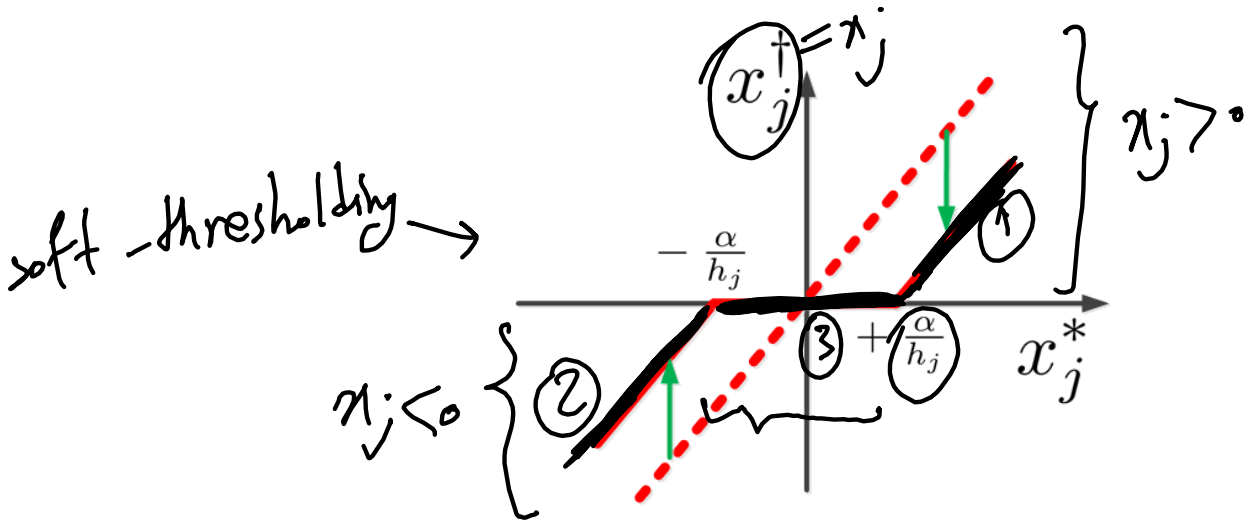
which is a soft thresholding function.



Theory for ℓ_1 Norm Regularization

- Soft-thresholding:

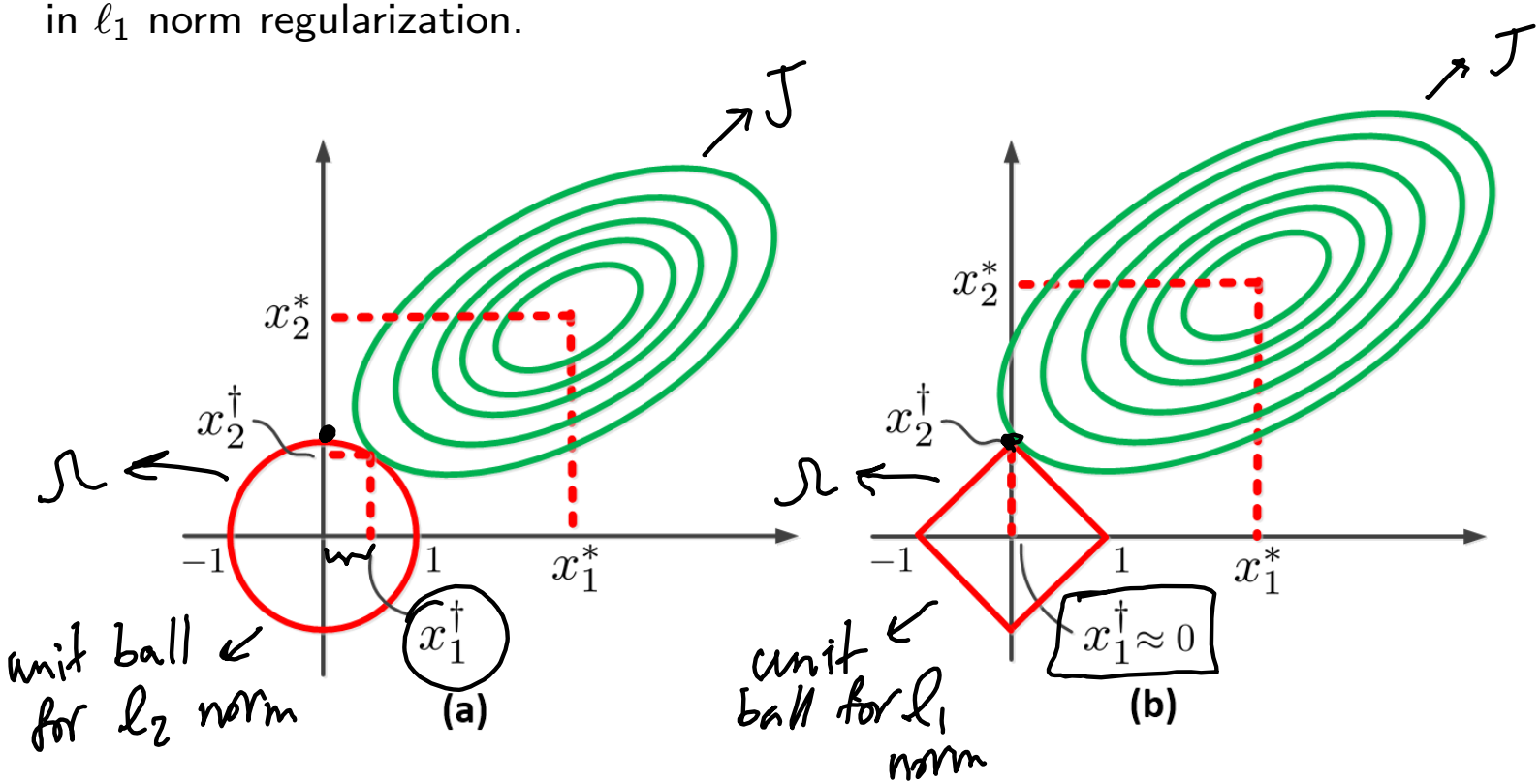
$$x_j^\dagger = x_j^* - \frac{\alpha}{h_j}\,\mathbf{sign}(x_j) = \begin{cases} x_j^* - \frac{\alpha}{h_j} & \text{if } x_j > 0, \\ x_j^* + \frac{\alpha}{h_j} & \text{if } x_j < 0, \end{cases} \qquad x_j = 0$$



- If $|x_j^*| < (\alpha/h_j)$, the solution to the regularized problem, i.e., $\boldsymbol{x}_j^\dagger$, is zero. Recall that in ℓ_2 norm regularization, we shrank the weak solutions close to zero; however, here in ℓ_1 norm regularization, we are setting the weak solutions exactly to zero. That is why the solutions are relatively sparse in ℓ_1 norm regularization.
- In ℓ_1 norm regularization, as shown in this figure, even the strong solutions are a little shrunk (from the $x_j^\dagger = x_j^*$ line), the fact that we also had in ℓ_2 norm regularization.

Comparison of ℓ_2 and ℓ_1 Regularization

- Another intuition for why the ℓ_1 norm regularization is sparse is illustrated below (credit if Tibshirani - 1996 [8]).
- The objective $J(\boldsymbol{x}; \theta)$ has some contour levels like a bowl (if it is convex). The regularization term is also a norm ball, which is a sphere bowl (cone) for ℓ_2 norm and a diamond bowl (cone) for ℓ_1 norm [15].
- For ℓ_2 norm regularization, the objective and the penalty term contact at a point where some of the coordinates might be small; however, for ℓ_1 norm, the contact point can be at some point where some variables are exactly zero. This again shows the reason of sparsity in ℓ_1 norm regularization.



Acknowledgement

- Some slides are inspired by the textbook: Trevor Hastie, Robert Tibshirani, Jerome Friedman, "The elements of statistical learning: Data Mining, Inference, and Prediction", Springer, 2009 [7].
- Another textbook suitable for sparsity in machine learning is: Robert Tibshirani, Martin Wainwright, Trevor Hastie, "Statistical learning with sparsity: the lasso and generalizations", Chapman and Hall/CRC, 2015 [13].
- Some slides are inspired by the lectures of Prof. Ali Ghodsi at the University of Waterloo, Department of Statistics and Actuarial Science, see his YouTube channel: `https://www.youtube.com/@DataScienceCoursesUW`
- Some slides are also inspired by the lectures of Prof. Hoda Mohammadzade at the Sharif University of Technology, Department of Electrical Engineering.
- Some slides are based on our tutorial paper: "The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial" [2]

References

[1] C. M. Stein, "Estimation of the mean of a multivariate normal distribution," *The annals of Statistics*, pp. 1135–1151, 1981.

[2] B. Ghojogh and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial," *arXiv preprint arXiv:1905.12787*, 2019.

[3] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[4] V. Barnett, *Elements of sampling theory*.
English Universities Press, London, 1974.

[5] B. Ghojogh, H. Nekoei, A. Ghojogh, F. Karray, and M. Crowley, "Sampling algorithms, from survey sampling to Monte Carlo methods: Tutorial and literature review," *arXiv preprint arXiv:2011.00901*, 2020.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*.
MIT press, 2016.

[7] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning: Data Mining, Inference, and Prediction*, vol. 2.
Springer series in statistics, New York, NY, USA, 2009.

[8] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

References (cont.)

[9] M. Schmidt, "Least squares optimization with l1-norm regularization," *CS542B Project Report*, vol. 504, pp. 195–221, 2005.

[10] Y. Chang, "$L_{2,1}$ norm and its applications," *Technical Report, University of Central Florida*.

[11] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Convex optimization with sparsity-inducing norms," *Optimization for Machine Learning*, vol. 5, pp. 19–53, 2011.

[12] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski, "Optimization with sparsity-inducing penalties," *Foundations and Trends® in Machine Learning*, vol. 4, no. 1, pp. 1–106, 2012.

[13] R. Tibshirani, M. Wainwright, and T. Hastie, *Statistical learning with sparsity: the lasso and generalizations*.
Chapman and Hall/CRC, 2015.

[14] P. Domingos, "The role of Occam's razor in knowledge discovery," *Data mining and knowledge discovery*, vol. 3, no. 4, pp. 409–425, 1999.

[15] S. Boyd and L. Vandenberghe, *Convex optimization*.
Cambridge university press, 2004.

[16] D. L. Donoho, "For most large underdetermined systems of linear equations the minimal ℓ_1-norm solution is also the sparsest solution," *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, vol. 59, no. 6, pp. 797–829, 2006.

References (cont.)

[17] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[18] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.

[19] T. T. Wu and K. Lange, "Coordinate descent algorithms for lasso penalized regression," *The Annals of Applied Statistics*, vol. 2, no. 1, pp. 224–244, 2008.

[20] G. Casella and E. I. George, "Explaining the Gibbs sampler," *The American Statistician*, vol. 46, no. 3, pp. 167–174, 1992.