# The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial

**Benyamin Ghojogh**                                    BGHOJOGH@UWATERLOO.CA

Department of Electrical and Computer Engineering,
Machine Learning Laboratory, University of Waterloo, Waterloo, ON, Canada

**Mark Crowley**                                         MCROWLEY@UWATERLOO.CA

Department of Electrical and Computer Engineering,
Machine Learning Laboratory, University of Waterloo, Waterloo, ON, Canada

Lecture of Benyamin Ghojogh about this tutorial: https://www.youtube.com/watch?v=wds4KdXQJIA
Lecture of Prof. Ali Ghodsi about this tutorial: https://www.youtube.com/watch?v=21jL0I6wbns

## Abstract

In this tutorial paper, we first define mean squared error, variance, covariance, and bias of both random variables and classification/predictor models. Then, we formulate the true and generalization errors of the model for both training and validation/test instances where we make use of the Stein's Unbiased Risk Estimator (SURE). We define overfitting, underfitting, and generalization using the obtained true and generalization errors. We introduce cross validation and two well-known examples which are $K$-fold and leave-one-out cross validations. We briefly introduce generalized cross validation and then move on to regularization where we use the SURE again. We work on both $\ell_2$ and $\ell_1$ norm regularizations. Then, we show that bootstrap aggregating (bagging) reduces the variance of estimation. Boosting, specifically AdaBoost, is introduced and it is explained as both an additive model and a maximum margin model, i.e., Support Vector Machine (SVM). The upper bound on the generalization error of boosting is also provided to show why boosting prevents from overfitting. As examples of regularization, the theory of ridge and lasso regressions, weight decay, noise injection to input/weights, and early stopping are explained. Random forest, dropout, histogram of oriented gradients, and single shot multi-box detector are explained as examples
of bagging in machine learning and computer vision. Finally, boosting tree and SVM models are mentioned as examples of boosting.

## 1. Introduction

Assume we have a dataset of *instances* $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ with sample size $N$ and dimensionality $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. The $\{\boldsymbol{x}_i\}_{i=1}^N$ are the input data to the model and the $\{y_i\}_{i=1}^N$ are the observations (labels). We denote the dataset by $\mathcal{D}$ so that $N := |\mathcal{D}|$. This dataset is the union of the disjoint subsets, i.e., training set $\mathcal{T}$ and test set $\mathcal{R}$; therefore:

$$\mathcal{D} = \mathcal{T} \cup \mathcal{R}, \tag{1}$$
$$\mathcal{T} \cap \mathcal{R} = \varnothing. \tag{2}$$

For the training set, the observations (labels), $y_i$'s, are available. Although for the test set, we might also have $y_i$'s, but we do not use them for training the model. The observations are continuous or come from a finite discrete set of values in classification and prediction (regression) tasks, respectively. Assume the sample size of training and test sets are $n := |\mathcal{T}|$ and $m := N - n$, respectively; therefore, we have $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ as the training set. In some cases where we want to have validation set $\mathcal{V}$ as well, the datasets includes three disjoint subsets:

$$\mathcal{D} = \mathcal{T} \cup \mathcal{R} \cup \mathcal{V}, \tag{3}$$
$$\mathcal{T} \cap \mathcal{R} = \varnothing, \mathcal{T} \cap \mathcal{V} = \varnothing, \mathcal{V} \cap \mathcal{R} = \varnothing. \tag{4}$$

We will define the intuitions of training, test, and validation sets later in this paper.

In this paper, we introduce overfitting, cross validation, generalized cross validation, regularization, bagging, and
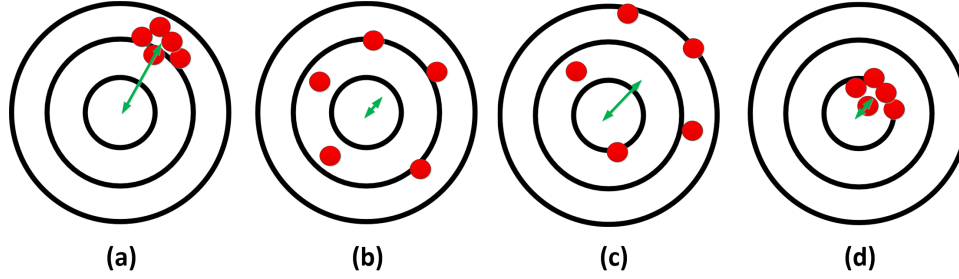
*Figure 1.* The dart example for (a) high bias and low variance, (b) low bias and high variance, (c) high bias and high variance, and (d) low bias and low variance. The worst and best cases are (c) and (d), respectively. The center of the circles is the true value of the variable.

boosting and explain why they work theoretically. We also provide some examples of these methods in machine learning and computer vision.

It is noteworthy that Sections 3, 4, 5.2, and some parts of Section 7 (i.e., analysis of overfitting and regularization using SURE), and Section 8.2 (analysis of bagging), were primarily proposed by Prof. Ali Ghodsi verbally in his lectures, at University of Waterloo, available on YouTube[1]. The credit of those sections is his. Moreover some parts of Section 7 have been discussed in the books (Friedman et al., 2009) and (Goodfellow et al., 2016).

## 2. Mean Squared Error, Variance, and Bias

### 2.1. Measures for a Random Variable

Assume we have variable $X$ and we estimate it. Let the random variable $\widehat{X}$ denote the estimate of $X$. The *variance* of estimating this random variable is defined as:

$$\mathbb{Var}(\widehat{X}) := \mathbb{E}\big((\widehat{X} - \mathbb{E}(\widehat{X}))^2\big), \tag{5}$$

which means average deviation of $\widehat{X}$ from the mean of our estimate, $\mathbb{E}(\widehat{X})$, where the deviation is squared for symmetry of difference. This variance can be restated as:

$$\mathbb{Var}(\widehat{X}) = \mathbb{E}\big(\widehat{X}^2 + (\mathbb{E}(\widehat{X}))^2 - 2\widehat{X}\mathbb{E}(\widehat{X})\big)$$
$$\overset{(a)}{=} \mathbb{E}(\widehat{X}^2) + (\mathbb{E}(\widehat{X}))^2 - 2\mathbb{E}(\widehat{X})\mathbb{E}(\widehat{X})$$
$$= \mathbb{E}(\widehat{X}^2) - (\mathbb{E}(\widehat{X}))^2, \tag{6}$$

where $(a)$ is because expectation is a linear operator and $\mathbb{E}(\widehat{X})$ is not a random variable.

Our estimation can have a bias. The *bias* of our estimate is defined as:

$$\mathbb{Bias}(\widehat{X}) := \mathbb{E}(\widehat{X}) - X, \tag{7}$$

which means how much the mean of our estimate deviates from the original $X$.

The *Mean Squared Error (MSE)* of our estimate, $\widehat{X}$, is defined as:

$$\mathrm{MSE}(\widehat{X}) := \mathbb{E}\big((\widehat{X} - X)^2\big), \tag{8}$$

which means how much our estimate deviates from the original $X$.

The intuition of bias, variance, and MSE is illustrated in Fig. 1 where the estimations are like a dart game. We have four cases with low/high values of bias and variance which are depicted in this figure.

The relation of MSE, variance, and bias is as follows:

$$\mathrm{MSE}(\widehat{X}) = \mathbb{E}\big((\widehat{X} - X)^2\big)$$
$$= \mathbb{E}\big((\widehat{X} - \mathbb{E}(\widehat{X}) + \mathbb{E}(\widehat{X}) - X)^2\big)$$
$$= \mathbb{E}\big((\widehat{X} - \mathbb{E}(\widehat{X}))^2 + (\mathbb{E}(\widehat{X}) - X)^2$$
$$\quad + 2(\widehat{X} - \mathbb{E}(\widehat{X}))(\mathbb{E}(\widehat{X}) - X)\big)$$
$$\overset{(a)}{=} \mathbb{E}\big((\widehat{X} - \mathbb{E}(\widehat{X}))^2\big) + (\mathbb{E}(\widehat{X}) - X)^2$$
$$\quad + 2\underbrace{(\mathbb{E}(\widehat{X}) - \mathbb{E}(\widehat{X}))}_{0}(\mathbb{E}(\widehat{X}) - X)$$
$$\overset{(b)}{=} \mathbb{Var}(\widehat{X}) + (\mathbb{Bias}(\widehat{X}))^2, \tag{9}$$

where $(a)$ is because expectation is a linear operator and $X$ and $\mathbb{E}(\widehat{X})$ are not random, and $(b)$ is because of Eqs. (5) and (7).

If we have two random variables $\widehat{X}$ and $\widehat{Y}$, we can say:

$$\mathbb{Var}(a\widehat{X} + b\widehat{Y}) \overset{(6)}{=} \mathbb{E}\big((a\widehat{X} + b\widehat{Y})^2\big) - \big(\mathbb{E}(a\widehat{X} + b\widehat{Y})\big)^2$$
$$\overset{(a)}{=} a^2\,\mathbb{E}(\widehat{X}^2) + b^2\,\mathbb{E}(\widehat{Y}^2) + 2ab\,\mathbb{E}(\widehat{X}\widehat{Y})$$
$$\quad - a^2\,(\mathbb{E}(\widehat{X}))^2 - b^2\,(\mathbb{E}(\widehat{Y}))^2 - 2ab\,\mathbb{E}(\widehat{Y})\mathbb{E}(\widehat{Y})$$
$$\overset{(6)}{=} a^2\,\mathbb{Var}(\widehat{X}) + b^2\,\mathbb{Var}(\widehat{X}) + 2ab\,\mathbb{Cov}(\widehat{X}, \widehat{Y}), \tag{10}$$

where $(a)$ is because of linearity of expectation and the $\mathbb{Cov}(\widehat{X}, \widehat{Y})$ is *covariance* defined as:

$$\mathbb{Cov}(\widehat{X}, \widehat{Y}) := \mathbb{E}(\widehat{X}\widehat{Y}) - \mathbb{E}(\widehat{X})\,\mathbb{E}(\widehat{Y}). \tag{11}$$

If the two random variables are independent, i.e., $X \perp\!\!\!\perp Y$, we have:

$$\mathbb{E}(\widehat{X}\widehat{Y}) \overset{(a)}{=} \iint \widehat{x}\widehat{y}f(\widehat{x},\widehat{y})d\widehat{x}d\widehat{y} \overset{\perp\!\!\!\perp}{=} \iint \widehat{x}\widehat{y}f(\widehat{x})f(\widehat{y})d\widehat{x}d\widehat{y}$$

$$= \int \widehat{y}f(\widehat{y})\underbrace{\int \widehat{x}f(\widehat{x})d\widehat{x}}_{\mathbb{E}(\widehat{X})} d\widehat{y} = \mathbb{E}(\widehat{X})\underbrace{\int \widehat{y}f(\widehat{y})d\widehat{y}}_{\mathbb{E}(\widehat{Y})}$$

$$= \mathbb{E}(\widehat{X})\,\mathbb{E}(\widehat{Y}) \implies \mathbb{C}\text{ov}(\widehat{X},\widehat{Y}) = 0, \qquad (12)$$

where $(a)$ is according to definition of expectation. Note that Eq. (12) is not true for the reverse implication (we can prove by counterexample).

We can extend Eqs. (10) and (11) to multiple random variables:

$$\mathbb{V}\text{ar}\Big(\sum_{i=1}^{k} a_i X_i\Big)$$

$$= \sum_{i=1}^{k} a_i^2\,\mathbb{V}\text{ar}(X_i) + \sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k} a_i a_j \mathbb{C}\text{ov}(X_i, X_j),$$

$$(13)$$

$$\mathbb{C}\text{ov}\Big(\sum_{i=1}^{k_1} a_i X_i, \sum_{j=1}^{k_2} b_j Y_j\Big) = \sum_{i=1}^{k_1}\sum_{j=1}^{k_2} a_i\,b_j\,\mathbb{C}\text{ov}(X_i, Y_j),$$

$$(14)$$

where $a_i$'s and $b_j$'s are not random.

## 2.2. Measures for a Model

Assume we have a *function* $f$ which gets the $i$-th input $\boldsymbol{x}_i$ and outputs $f_i = f(\boldsymbol{x}_i)$. Figure 2 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 $\varepsilon_i$:

$$y_i = f_i + \varepsilon_i, \qquad (15)$$

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

$$\mathbb{E}(\varepsilon_i) = 0, \quad \mathbb{E}(\varepsilon_i^2) \overset{(6)}{=} \mathbb{V}\text{ar}(\varepsilon_i) + (\mathbb{E}(\varepsilon_i))^2 = \sigma^2, \quad (16)$$

The true observation $f_i$ is not random, thus:

$$\mathbb{E}(f_i) = f_i. \qquad (17)$$

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 $\{y_i\}_{i=1}^n$
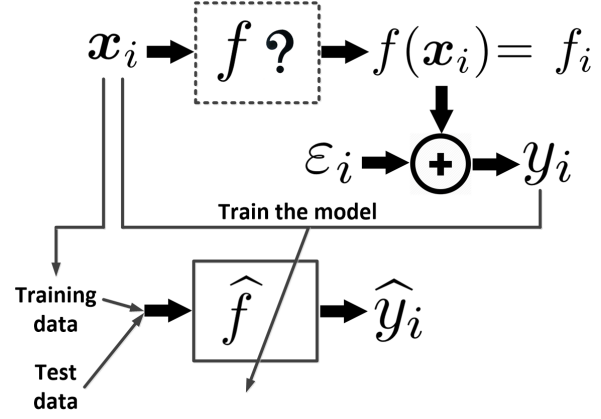


*Figure 2.* The true model and the estimated model which is trained using the input training data and their observations. The observations are obtained from the outputs of the true model fed with the training input but corrupted by the noise. After the model is trained, it can be used to estimate the observation for either training or test input data.
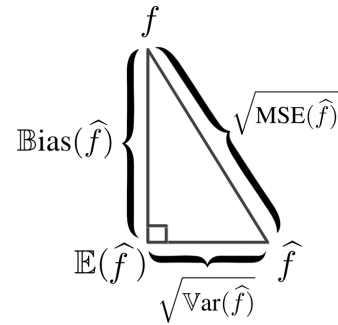


*Figure 3.* The triangle of variance, bias, and MSE. The $f$ and $\widehat{f}$ are the true and estimated models, respectively.

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. The explained details are illustrated in Fig. 2.

In this work, 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, i.e., Eqs. (5), (7), and (8), can also be used for the estimation $\widehat{f}_i$ of the true model $f_i$. The Eq. (9) can be illustrated for the model $f$ as in Fig. 3 which holds because of Pythagorean theorem.

## 2.3. Measures for Ensemble of Models

If we have an ensemble of models (Polikar, 2012), we can have some similar definitions of bias and variance (e.g., see the Appendix C in (Schapire et al., 1998)). Here, we assume the models are classifiers.

If $\mathbb{P}(.)$ denotes the probability, the *expected error* or *prediction error* (PE) of the model $f_i$ is defined as:

$$\text{PE}(f) := \mathbb{P}(\widehat{f}_i \neq y_i), \tag{18}$$

where $\widehat{f}_i$ is the estimation of trained model for the observation $y_i$ (input $\boldsymbol{x}_i$).

In the parentheses, it is required to mention that Bayesian classifier is the optimal classifier because it can be seen as an ensemble of hypotheses (models) in the hypothesis (model) space and no other ensemble of hypotheses can outperform it (see Chapter 6, Page 175 in (Mitchell, 1997)). In the literature, it is referred to as *Bayes optimal classifier*. However, implementing Bayesian classifier is difficult so they approximate it by *naive Bayes* (Zhang, 2004).

Back to our main discussion, let $\widehat{f}^*$ denote the Bayes optimal prediction. Also, let the estimate of each of the trained models by an ensemble learning method (such as bagging or boosting which will be introduced later) be denoted by $\widehat{f}$ which is trained using $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Finally, let $\widehat{f}^m$ denote the classification using majority voting between the models. The bias and variance of the model can be defined as (Kong & Dietterich, 1995):

$$\mathbb{B}\text{ias}(\widehat{f}) := \text{PE}(\widehat{f}^m) - \text{PE}(\widehat{f}^*), \tag{19}$$

$$\mathbb{V}\text{ar}(\widehat{f}) := \mathbb{E}(\text{PE}(\widehat{f})) - \text{PE}(\widehat{f}^m). \tag{20}$$

There also exist another definition in the literature. Suppose the sample space of data is the union of two disjoint subsets $\mathcal{U}$ and $\mathcal{B}$ which are the unbiased and biased sets with $\widehat{f}_i^m = \widehat{f}_i^*$ and $\widehat{f}_i^m \neq \widehat{f}_i^*$, respectively. We can define (Breiman, 1998):

$$\mathbb{B}\text{ias}(\widehat{f}_i)$$
$$:= \mathbb{P}(\widehat{f}_i^* = y_i, \boldsymbol{x}_i \in \mathcal{B}) - \mathbb{E}(\mathbb{P}(\widehat{f}_i = y_i, \boldsymbol{x}_i \in \mathcal{B})), \tag{21}$$
$$\mathbb{V}\text{ar}(\widehat{f}_i)$$
$$:= \mathbb{P}(\widehat{f}_i^* = y_i, \boldsymbol{x}_i \in \mathcal{U}) - \mathbb{E}(\mathbb{P}(\widehat{f}_i = y_i, \boldsymbol{x}_i \in \mathcal{U})). \tag{22}$$

## 3. 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. (15), the observation $y_0$ is:

$$y_0 = f_0 + \varepsilon_0. \tag{23}$$

Assume the model's estimation of $y_0$ is $\widehat{f}_0$. According to Eq. (8), the MSE of the estimation is:

$$\mathbb{E}((\widehat{f}_0 - y_0)^2) \overset{(23)}{=} \mathbb{E}((\widehat{f}_0 - f_0 - \varepsilon_0)^2)$$
$$= \mathbb{E}((\widehat{f}_0 - f_0)^2 + \varepsilon_0^2 - 2\varepsilon_0(\widehat{f}_0 - f_0))$$
$$= \mathbb{E}((\widehat{f}_0 - f_0)^2) + \mathbb{E}(\varepsilon_0^2) - 2\mathbb{E}(\varepsilon_0(\widehat{f}_0 - f_0))$$
$$\overset{(16)}{=} \mathbb{E}((\widehat{f}_0 - f_0)^2) + \sigma^2 - 2\mathbb{E}(\varepsilon_0(\widehat{f}_0 - f_0)). \tag{24}$$

The last term is:

$$\mathbb{E}(\varepsilon_0(\widehat{f}_0 - f_0)) \overset{(23)}{=} \mathbb{E}((y_0 - f_0)(\widehat{f}_0 - f_0)). \tag{25}$$

For calculation of this last term, we consider 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.

### 3.1. 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:

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

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

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

where $(c)$ is because of Eq. (17) and:

$$\mathbb{E}(y_0) \overset{(23)}{=} \mathbb{E}(f_0) + \mathbb{E}(\varepsilon_0) = f_0 + 0 = f_0.$$

Therefore, in this case, the last term in Eq. (24) is zero. Thus:

$$\mathbb{E}((\widehat{f}_0 - y_0)^2) = \mathbb{E}((\widehat{f}_0 - f_0)^2) + \sigma^2 \tag{26}$$

Suppose the number of instances which are not in the training set is $m$. By Monte Carlo approximation of the expectation terms (Ghojogh et al., 2020), we have:

$$\frac{1}{m}\sum_{i=1}^m (\widehat{f}_i - y_i)^2 = \frac{1}{m}\sum_{i=1}^m (\widehat{f}_i - f_i)^2 + \sigma^2 \implies$$
$$\sum_{i=1}^m (\widehat{f}_i - y_i)^2 = \sum_{i=1}^m (\widehat{f}_i - f_i)^2 + m\sigma^2. \tag{27}$$

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:

$$\mathbf{err} = \mathbf{Err} + m\,\sigma^2 \implies \mathbf{Err} = \mathbf{err} - m\,\sigma^2. \quad (28)$$

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.

### 3.2. Case II: Instance in the Training Set

In case II, 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:

$$\mathbb{E}\big((\boldsymbol{z} - \boldsymbol{\mu})^\top \boldsymbol{g}(\boldsymbol{z})\big) = \sigma^2 \sum_{i=1}^{d} \mathbb{E}\big(\frac{\partial g_i}{\partial z_i}\big), \quad (29)$$

which is used in *Stein's Unbiased Risk Estimate (SURE)* (Stein, 1981). See Appendix A in this tutorial paper for the proof of Eq. (29).

If the random variable is a univariate variable, the Stein's lemma becomes:

$$\mathbb{E}\big((z - \mu)\, g(z)\big) = \sigma^2 \mathbb{E}\big(\frac{\partial g(z)}{\partial z}\big). \quad (30)$$

Suppose we take $\varepsilon_0$, $0$, and $\widehat{f}_0 - f_0$ as the $z$, $\mu$, and $g(z)$, respectively, in the Stein's lemma for univariate variable. Using Eq. (30), the last term in Eq. (24) is:

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

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

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

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:

$$y_0 \overset{(23)}{=} f_0 + \varepsilon_0 \implies \frac{\partial y_0}{\partial\varepsilon_0} = 1.$$

Therefore, in this case, the Eq. (24) is:

$$\mathbb{E}\big((\widehat{f}_0 - y_0)^2\big) = \mathbb{E}\big((\widehat{f}_0 - f_0)^2\big) + \sigma^2 - 2\sigma^2 \mathbb{E}\big(\frac{\partial\widehat{f}_0}{\partial y_0}\big). \quad (31)$$

Suppose the number of training instances is $n$. By Monte Carlo approximation of the expectation terms (Ghojogh et al., 2020), we have:

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

$$\implies$$

$$\sum_{i=1}^{n}(\widehat{f}_i - y_i)^2 = \sum_{i=1}^{n}(\widehat{f}_i - f_i)^2 + n\sigma^2 - 2\sigma^2\sum_{i=1}^{n}\frac{\partial\widehat{f}_i}{\partial y_i}. \quad (32)$$

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:

$$\mathbf{Err} = \mathbf{err} - n\,\sigma^2 + 2\,\sigma^2 \sum_{i=1}^{n}\frac{\partial\widehat{f}_i}{\partial y_i}. \quad (33)$$

The last term in Eq. (33) 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). See Fig. 4 illustrating the explained examples.

According to Eq. (33), 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.

### 3.3. Estimation of $\sigma$ in MSE of the Model

The MSE of the model in the both mentioned cases include $\sigma$ which is the standard deviation of the noise. An unbiased estimation of the variance of the noise is:

$$\sigma^2 \approx \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \widehat{f}_i), \quad (34)$$
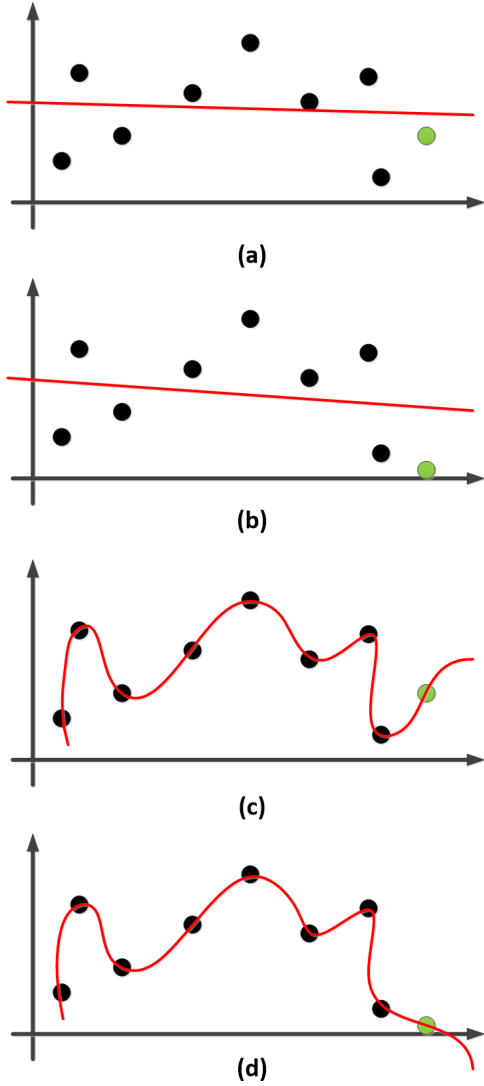
*Figure 4.* An example for a simple and a complex model: (a) a simple model, (b) the modified simple model after moving a training instance (shown in green), (c) a complex model, and (d) the modified simple model after moving a training instance (shown in green). The complex model is impacted significantly by moving the instance while the simple model is not that much affected.

which uses the training observations and their estimation by the model. However, the model's estimations of the training observations are themselves dependent on the complexity of the model.

For the explained problem, in practice, we use an estimator with high bias and low variance in order to estimate the $\sigma$ in order not to have an estimation dependent on the complexity fo the model. For example, we use a line fitted to the training data using least squares problem (linear regression) in order to have estimations $\widehat{f}_i$'s of the $y_i$'s and then we use Eq. (34). Thereafter, for the sake of simplicity, we

do not change the $\sigma$ (assume it is fixed) when we change the complexity of the model.

## 4. 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.

An example for underfitting, good fit, and overfitting is illustrated in Fig. 5. 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!

## 5. Cross Validation

### 5.1. Definition

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* (Arlot & Celisse, 2010). 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:

$$\mathcal{T} \cup \mathcal{R} = \mathcal{D}, \tag{35}$$
$$\mathcal{T} \cap \mathcal{R} = \varnothing. \tag{36}$$

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).

In $K$-fold cross validation, we randomly split the dataset $\mathcal{D}$
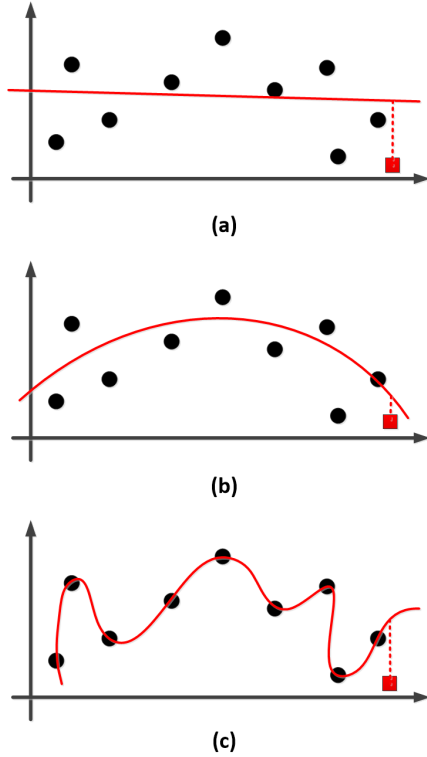
**(a)**

**(b)**

**(c)**

*Figure 5.* An example for (a) underfitting, (b) good fit, and (c) overfitting. The black circles and red square are training and test instances, respectively. The red curve is the fitted curve.

into $K$ partitions $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ where:

$$|\mathcal{D}_1| \approx |\mathcal{D}_2| \approx \cdots \approx |\mathcal{D}_K|, \tag{37}$$

$$\bigcup_{i=1}^{K} \mathcal{D}_i = \mathcal{D}, \tag{38}$$

$$\mathcal{D}_i \cap \mathcal{D}_j = \varnothing, \quad \forall i, j \in \{1, \ldots, K\}, \; i \neq j, \tag{39}$$

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 (Barnett, 1974; Ghojogh et al., 2020) 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. Note that we usually have $K = 2, 5, 10$ in the literature but $K = 10$ is the most common. The algorithm of $K$-fold cross validation is shown in Algorithm 1.

In 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. The algorithm of LOOCV is shown in Algorithm 2. Usually,

---

**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

---

**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

---

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.

If we want to train the model and then test it, the cross validation should be done using training and test sets as explained. Note that 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).

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.

In cross validation with validation set, we have:

$$\mathcal{T} \cap \mathcal{R} = \varnothing, \ \mathcal{T} \cap \mathcal{V} = \varnothing, \ \mathcal{V} \cap \mathcal{R} = \varnothing. \qquad (40)$$

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). Note that 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.

In order 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 $|\mathcal{V}| = 1$.

### 5.2. Theory

Recall the Eqs. (28) and (33) where the true error for the test (not in the training set) and training instance are related to the training error, respectively. When the instance is in the training set, the true error, **Err**, and the test error, **err**, behave differently as shown in Fig. 6-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 (Goodfellow et al., 2016) where **Err** is measured after every epoch for example. For neural networks, we usually save a history of **Err** for several last epochs and if the overall pattern of **Err** is increasing, we stop and take the last best trained model with least **Err**. We do this because in complex models such as neural networks, the curve of **Err** usually has some small fluctuations which we do not want to misjudge the stopping criterion based on those. This procedure is named *early stopping* in neural networks (Prechelt, 1998) which will be explained later in Section 7.4.4.

The reason why **Err** increases after a while of training is according to Eq. (33). Dropping the constant $n\sigma^2$ from that

expression, we have: $\mathbf{Err} = \mathbf{err} + 2\,\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i}$ where the term $2\,\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i}$ shows the model complexity. See Fig. 6-b where both **err** and the model complexity are illustrated as a function of training stages (iterations). According to Eq. (33), the **Err** is the summation of these two curves which clarifies the reason of its behavior. 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. 6-a and Eq. (33) show that it is better to have $\mathcal{T} \cap \mathcal{V} = 0$. 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. (28) 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.

Finally, it is noteworthy to discuss the intersections of training, test, and validation sets according to above explanations and the previous sub-section. 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:

- $\mathcal{T} \cap \mathcal{R} \neq \varnothing \implies$ 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.

## 6. Generalized Cross Validation

In this section, we consider the model which estimates the observations $\{y_i\}_{i=1}^{N}$ as:

$$\widehat{\boldsymbol{y}} = \boldsymbol{\Gamma}\,\boldsymbol{y}, \qquad (41)$$

where $\widehat{\boldsymbol{y}} = [\widehat{y}_1, \ldots, \widehat{y}_N]^\top$ and $\boldsymbol{y} = [y_1, \ldots, y_N]^\top$ assuming that the observations for the whole dataset are available. The $\boldsymbol{\Gamma} \in \mathbb{R}^{N \times N}$ is called the hat matrix because it puts a hat on $\boldsymbol{y}$. An example of $\boldsymbol{\Gamma}$ is $\boldsymbol{\Gamma} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top$ which is used in linear regression (Friedman et al., 2009). If $\gamma_{ij}$ denotes the $(i, j)$-th element of $\boldsymbol{\Gamma}$, the $i$-th element of $\widehat{\boldsymbol{y}}$ can
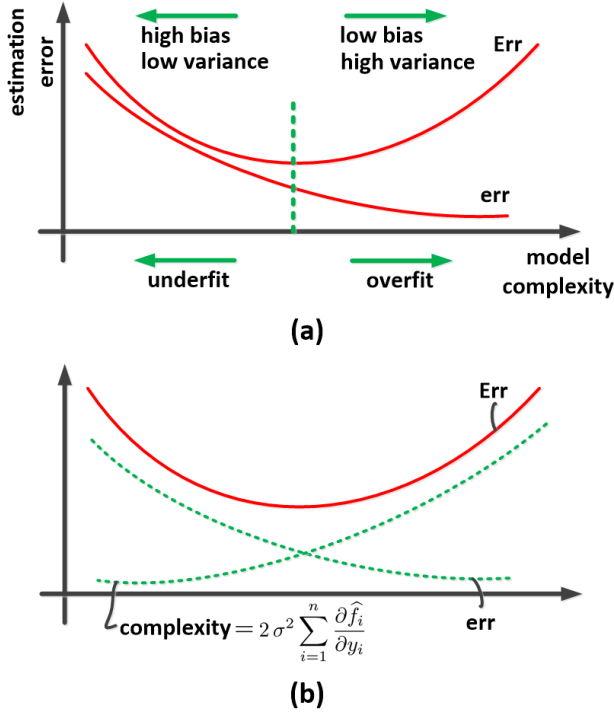
Figure 6. The overfitting of model: (a) training error and true error, (b) depiction of Eq. (33).

be stated as:

$$\widehat{y}_i = \sum_{j=1}^{N} \gamma_{ij}\, y_j. \tag{42}$$

Now assume that we remove the $i$-th instance for the sake of having LOOCV. Assume that $\widehat{y}_i^{(-i)}$ denotes the model's estimate of $y_i$ where the model is trained using $\mathcal{D} \setminus \{x_i\}$ (using the entire data except the $i$-th instance). We can say:

$$\widehat{y}_i^{(-i)} = \Big( \sum_{j=1}^{N} \gamma_{ij}\, y_j \Big) - \gamma_{ii}\, y_i + \gamma_{ij}\, \widehat{y}_i^{(-i)}, \tag{43}$$

which means that we remove the estimation of $y_i$ using the model trained by the whole $\mathcal{D}$ and instead we put the estimation of the model trained by $\mathcal{D} \setminus \{x_i\}$. Adding $y_i$ to the left- and right-hand sides of Eq. (43) gives:

$$y_i - \widehat{y}_i^{(-i)} = \frac{y_i - \widehat{y}_i}{1 - \gamma_{ii}}. \tag{44}$$

The Eq. (44) means that we can do LOOCV for the model of Eq. (41) without the need of iteration over the instances. We can train the model once using the whole $\mathcal{D}$ and then use Eq. (44) to find the error of every iteration of LOOCV. The overall scaled mean squared error of LOOCV, then, is:

$$\sum_{i=1}^{N} \big(y_i - \widehat{y}_i^{(-i)}\big)^2 = \sum_{i=1}^{N} \Big( \frac{y_i - \widehat{y}_i}{1 - \gamma_{ii}} \Big)^2. \tag{45}$$

Suppose that we replace the $\gamma_{ii}$ by its average:

$$\frac{1}{N} \sum_{i=1}^{N} \gamma_{ii} \overset{(a)}{=} \frac{1}{N}\, \mathbf{tr}(\mathbf{\Gamma}) \overset{(b)}{=} \frac{p}{N}, \tag{46}$$

where $\mathbf{tr}(.)$ is the trace of matrix, $(a)$ is because trace is equivalent to summation of diagonal, and $(b)$ assumes that the trace of the hat matrix is $p$. The $p$ can be considered as a the dimensionality of the subspace if the Eq. (41) is considered as a projection into a subspace.

Using Eq. (46) in Eq. (45) gives:

$$\sum_{i=1}^{N} \big(y_i - \widehat{y}_i^{(-i)}\big)^2 = \sum_{i=1}^{N} \Big( \frac{y_i - \widehat{y}_i}{1 - p/N} \Big)^2. \tag{47}$$

The Eq. (47) is referred to as *generalized cross validation* (Carven & Wahba, 1979; Golub et al., 1979). It is noteworthy that the generalized cross validation can also be related to SURE (Stein, 1981) which was introduced before (see (Li, 1985)).

## 7. Regularization

### 7.1. Definition

We can minimize the true error, **Err**, using optimization. According to Eq. (33), we have:

$$\text{minimize} \quad \mathbf{err} - n\,\sigma^2 + 2\,\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i}. \tag{48}$$

As the term $n\,\sigma^2$ is a constant, we can drop it. Moreover, 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:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x}; \theta) := J(\boldsymbol{x}; \theta) + \alpha\,\Omega(\boldsymbol{x}), \tag{49}$$

where $\theta$ 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*.

The penalty function can be different things such as $\ell_2$ norm (Friedman et al., 2009), $\ell_1$ norm (Tibshirani, 1996; Schmidt, 2005), $\ell_{2,1}$ norm (Chang), etc. The $\ell_1$ and $\ell_{2,1}$ norms are useful for having sparsity (Bach et al., 2011; 2012). 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 (Friedman et al., 2009; Tibshirani et al., 2015)." The effectiveness of the sparsity can also be explained by

Occam's razor (Domingos, 1999) stating that "simpler solutions are more likely to be correct than complex ones" or "simplicity is a goal in itself".

Note that in Eqs. (48) and (49), we are minimizing the **Err** (i.e., $\widetilde{J}(x; \theta)$) and not **err** (i.e., $J(x; \theta)$). As discussed in Sections 4 and 5, minimizing **err** results in overfitting. Therefore, regularization helps avoid overfitting.

### 7.2. Theory for $\ell_2$ Norm Regularization

In this section, we briefly explain the theory behind the $\ell_2$ norm regularization (Friedman et al., 2009), which is:

$$\underset{x}{\text{minimize}} \quad \widetilde{J}(x; \theta) := J(x; \theta) + \frac{\alpha}{2} \|x\|_2^2. \qquad (50)$$

The $\ell_2$ norm regularization is also referred to as *ridge regression* or *Tikhonov regularization* (Goodfellow et al., 2016).

Suppose $x^*$ is minimizer of the $J(x; \theta)$, i.e.:

$$\nabla J(x^*; \theta) = 0. \qquad (51)$$

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

$$\begin{aligned} \widehat{J}(x; \theta) &\approx J(x^*; \theta) + \nabla J(x^*; \theta) \\ &\quad + \frac{1}{2}(x - x^*)^\top H(x - x^*) \\ &= J(x^*; \theta) + \frac{1}{2}(x - x^*)^\top H(x - x^*), \end{aligned} \qquad (52)$$

where $H \in \mathbb{R}^{d \times d}$ is the Hessian. Using the Taylor approximation in the cost gives us (Goodfellow et al., 2016):

$$\begin{aligned} \widetilde{J}(x; \theta) &= \widehat{J}(x; \theta) + \frac{\alpha}{2} \|x\|_2^2 \\ &= J(x^*; \theta) + \frac{1}{2}(x - x^*)^\top H(x - x^*) + \frac{\alpha}{2} \|x\|_2^2, \\ \frac{\partial \widetilde{J}(x; \theta)}{\partial x} &= 0 + H(x^\dagger - x^*) + \alpha x^\dagger \overset{\text{set}}{=} 0, \\ \implies & (H + \alpha I) x^\dagger = H x^* \\ \implies & x^\dagger = (H + \alpha I)^{-1} H x^*, \end{aligned} \qquad (53)$$

where $x^\dagger$ is the minimizer of $\widetilde{J}(x; \theta)$. Note that in calculations we take $\partial J(x^*; \theta)/\partial x = 0$ because the $J(x^*; \theta)$ is a constant vector with respect to $x$. The Eq. (53) makes sense because if $\alpha = 0$, which means we do not have the regularization term, we will have $x^\dagger = x^*$. This means that the minimizer of $\widetilde{J}(x, \theta)$ will be the same as the minimizer of $J(x; \theta)$ which is correct according to Eq. (50) where $\alpha = 0$.

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

$$H = U \Lambda U^\top, \qquad (54)$$

where $U$ and $\Lambda$ contain the eigenvectors and eigenvalues, respectively. Using this decomposition in Eq. (53) gives us:

$$\begin{aligned} x^\dagger &= (U \Lambda U^\top + \alpha I)^{-1} U \Lambda U^\top x^* \\ &\overset{(a)}{=} (U \Lambda U^\top + U U^\top \alpha I)^{-1} U \Lambda U^\top x^* \\ &\overset{(b)}{=} (U \Lambda U^\top + U \alpha I U^\top)^{-1} U \Lambda U^\top x^* \\ &= \big(U(\Lambda + \alpha I) U^\top\big)^{-1} U \Lambda U^\top x^* \\ &\overset{(c)}{=} U(\Lambda + \alpha I)^{-1} \underbrace{U^{-1} U}_{I} \Lambda U^\top x^* \\ &= U(\Lambda + \alpha I)^{-1} \Lambda U^\top x^*, \end{aligned} \qquad (55)$$

where $(a)$ and $(c)$ are because $U$ is an orthogonal matrix so we have $U^{-1} = U^\top$ which yields to $U^\top U = I$ and $U U^\top = I$ (because $U$ is not truncated). The $(b)$ is because $\alpha$ is a scalar and can move between the multiplication of matrices. The Eq. (55) means that we are rotating $x^*$ by $U^\top x^*$ but before rotating it back with $U U^\top x^*$, we manipulate it with the term $(\Lambda + \alpha I)^{-1} \Lambda$.

Based on Eq. (55), we can have the following interpretations:

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

$$x^\dagger = U \underbrace{\Lambda^{-1} \Lambda}_{I} U^\top x^* = U U^\top x^*$$

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

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

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

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

  where $\Lambda = \mathbf{diag}([\lambda_1, \ldots, \lambda_d]^\top)$. 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 $(\Lambda + \alpha I)^{-1} \Lambda \approx I$; therefore, $x^\dagger \approx x^*$. This makes sense because $\lambda_j \gg \alpha$ means that the $j$-th direction of Hessian and thus the $j$-th direction of $J(x; \theta)$ 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 $(\boldsymbol{\Lambda}+\alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda} \approx \boldsymbol{0}$; 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.

Therefore, the $\ell_2$ norm regularization keeps the effective directions but shrinks the weak directions to close to zero.

Note that the following measure is referred to as *effective number of parameters* or *degree of freedom* (Friedman et al., 2009):

$$\sum_{j=1}^{d} \frac{\lambda_j}{\lambda_j + \alpha}, \qquad (56)$$

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.

### 7.3. Theory for $\ell_1$ 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:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x};\theta) := J(\boldsymbol{x};\theta) + \alpha \, ||\boldsymbol{x}||_0, \qquad (57)$$

where:

$$||\boldsymbol{x}||_0 := \sum_{j=1}^{d} \mathbb{I}(x_j \neq 0) = \begin{cases} 0 & \text{if } x_j = 0, \\ 1 & \text{if } x_j \neq 0, \end{cases} \qquad (58)$$

is "$\ell_0$" norm, which is not a norm (so we used "." for it) because it does not satisfy the norm properties (Boyd & Vandenberghe, 2004). The "$\ell_0$" 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 (Donoho, 2006), the convex relaxation of "$\ell_0$" norm (subset selection) is $\ell_1$ norm. Therefore, we write the regularized optimization as:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{x};\theta) := J(\boldsymbol{x};\theta) + \alpha \, ||\boldsymbol{x}||_1. \qquad (59)$$

Note that the $\ell_1$ regularization is also referred to as *lasso* (least absolute shrinkage and selection operator) regularization (Tibshirani, 1996). Different methods exist for solving optimization having $\ell_1$ norm, such as proximal algorithm using soft thresholding (Parikh & Boyd, 2014) and coordinate descent (Wright, 2015; Wu & Lange, 2008). Here, we explain solving the optimization using the coordinate descent algorithm.
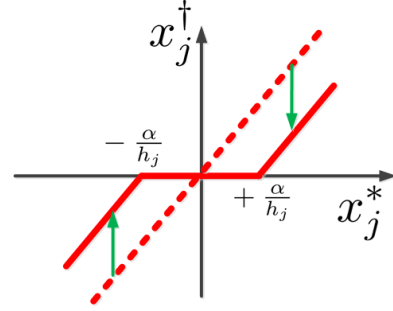


*Figure 7.* The soft thresholding function.

The idea of coordinate descent algorithm is similar to the idea of Gibbs sampling (Casella & George, 1992) where we work on the dimensions of the variable one by one. Similar to what we did for obtaining Eq. (53), we have:

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

For simplicity in deriving an interpretable expression, we assume that the Hessian matrix is diagonal (Goodfellow et al., 2016). For coordinate descent, we look at the $j$-th coordinate (dimension):

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

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:

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

$$x_j^\dagger = x_j^* - \frac{\alpha}{h_j}\,\mathbf{sign}(x_j^\dagger) = \begin{cases} x_j^* - \frac{\alpha}{h_j} & \text{if } x_j^\dagger > 0, \\ x_j^* + \frac{\alpha}{h_j} & \text{if } x_j^\dagger < 0, \end{cases} \qquad (60)$$

which is a soft thresholding function. This function is depicted in Fig. 7. As can be seen in this figure, if $|x_j^*| < (\alpha/h_j)$, the solution to the regularized problem, i.e., $x_j^\dagger$, is zero. Recall that in $\ell_2$ norm regularization, we shrank the weak solutions close to zero; however, here in $\ell_1$ norm regularization, we are setting the weak solutions exactly to zero. That is why the solutions are relatively sparse in $\ell_1$ norm regularization. Notice that in $\ell_1$ norm regularization, as shown in Fig. 7, even the strong solutions are a little shrunk (from the $x_j^\dagger = x_j^*$ line), the fact that we also had in $\ell_2$ norm regularization.

**(a)**



**(b)**
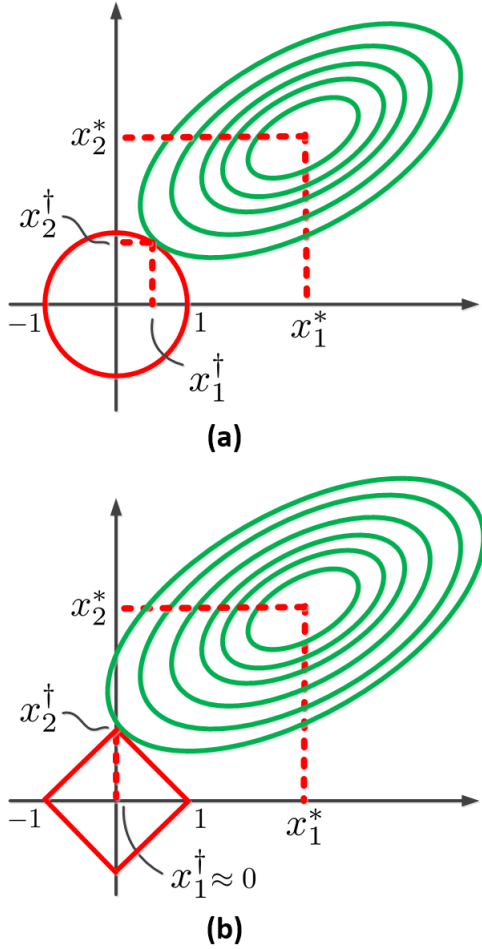
Figure 8. The unit balls for $\ell_1$ and $\ell_2$ norm regularizations: (a) $\ell_2$ norm regularization and (b) $\ell_1$ norm regularization. The green curves are the contour levels of the non-regularized objective function. The red balls show the unit balls for the norm penalties. The data are assumed to be two dimensional. A third dimension can be imagined for the value of cost function. This illustration was first proposed in (Tibshirani, 1996).

Another intuition for why the $\ell_1$ norm regularization is sparse is illustrated in Fig. 8 (Tibshirani, 1996). As this figure shows, 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 $\ell_2$ norm and a diamond bowl (cone) for $\ell_1$ norm (Boyd & Vandenberghe, 2004). As Fig. 8 shows, for $\ell_2$ norm regularization, the objective and the penalty term contact at a point where some of the coordinates might be small; however, for $\ell_1$ norm, the contact point can be at some point where some variables are exactly zero. This again shows the reason of sparsity in $\ell_1$ norm regularization.

## 7.4. Examples in Machine Learning: Regression, Weight Decay, Noise Injection, and Early Stopping

### 7.4.1. LINEAR, RIDGE, AND LASSO REGRESSION

Let $\boldsymbol{X} = [\boldsymbol{1}, [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top] \in \mathbb{R}^{n \times (d+1)}$ and $\boldsymbol{\beta} \in \mathbb{R}^{d+1}$. In *linear regression*, the optimization is (Friedman et al., 2009):

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2. \tag{61}$$

The result of this optimization is:

$$\boldsymbol{\beta} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{62}$$

We can penalize the regression coefficients using $\ell_2$ norm regularization. This is referred to as *ridge regression* whose optimization is (Friedman et al., 2009):

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad ||\boldsymbol{y} - \boldsymbol{\beta}\boldsymbol{X}||_2^2 + \frac{\alpha}{2}||\boldsymbol{\beta}||_2^2. \tag{63}$$

The result of this optimization is:

$$\boldsymbol{\beta} = (\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{64}$$

Note that one intuition of ridge regression is that adding $\alpha \boldsymbol{I}$ strengthens the main diagonal of $\boldsymbol{X}^\top \boldsymbol{X}$ in order to make it full-rank and non-singular for inversion.

We can also have $\ell_1$ norm regularization, named *lasso regression* (Tibshirani, 1996), which makes the coefficients sparse. The optimization of lasso regression is:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad ||\boldsymbol{y} - \boldsymbol{\beta}\boldsymbol{X}||_2^2 + \alpha ||\boldsymbol{\beta}||_1, \tag{65}$$

which dose not have a closed form solution but an iterative solution as explained before.

### 7.4.2. WEIGHT DECAY

Recall Eq. (50). If we replace the objective variable $\boldsymbol{x}$ with the vector of neural network weights $\boldsymbol{w}$, we will have:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{w}; \theta) := J(\boldsymbol{w}; \theta) + \frac{\alpha}{2}||\boldsymbol{w}||_2^2, \tag{66}$$

which can be the loss function optimized in a neural network (Goodfellow et al., 2016). Penalizing the weights with regularization is referred to as *weight decay* (Krogh & Hertz, 1992; Chiu et al., 1994). This penalty prevents neural network from becoming too non-linear (complex) and thus overfitted. the reason is that according to non-linear activation functions such as hyperbolic tangent, very large weights (very positive or very negative) are in the very non-linear parts of the activation functions. although neural network should not be completely linear in order to be able to learn non-linear patterns, it should not be very non-linear as well not to be overfitted to the training data. Penalizing the weights makes the weights relatively small (where the

activation functions are almost linear) in to have a balance in linearity and non-linearity.

According to Eq. (55), the result of Eq. (66) is:

$$\boldsymbol{w}^\dagger = \boldsymbol{U}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \boldsymbol{U}^\top \boldsymbol{w}^*, \tag{67}$$

which has the similar interpretations as we discussed before.

### 7.4.3. NOISE INJECTION TO INPUT IN NEURAL NETWORKS

In training neural networks, it is beneficial to add noise to the input (Matsuoka, 1992). One perspective to why adding noise to input helps better training of network is *data augmentation* (Van Dyk & Meng, 2001; DeVries & Taylor, 2017). Data augmentation is useful for training deep networks because they have a huge number of weights (parameters) and if we do not introduce enough training data to them, they will overfit to the training data.

Another interpretation of noise injection to input is regularization (Grandvalet et al., 1997; Goodfellow et al., 2016). Assume that the optimization of neural network is:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad J := \mathbb{E}((\widehat{y}(\boldsymbol{x}) - y)^2), \tag{68}$$

where $\boldsymbol{x}$, $\widehat{y}(\boldsymbol{x})$, and $y$ are the input, the estimation (output) of network, and the training label, respectively. We add noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ to the input, so the objective function changes to:

$$\begin{aligned} \widetilde{J} &:= \mathbb{E}((\widehat{y}(\boldsymbol{x} + \boldsymbol{\varepsilon}) - y)^2) \\ &= \mathbb{E}(\widehat{y}^2(\boldsymbol{x} + \boldsymbol{\varepsilon}) - 2y\widehat{y}(\boldsymbol{x} + \boldsymbol{\varepsilon}) + y^2) \\ &= \mathbb{E}(\widehat{y}^2(\boldsymbol{x} + \boldsymbol{\varepsilon})) - 2\mathbb{E}(y\widehat{y}(\boldsymbol{x} + \boldsymbol{\varepsilon})) + \mathbb{E}(y^2). \end{aligned}$$

Assuming that the variance of noise is small, the Taylor series expansion of $\widehat{y}(\boldsymbol{x} + \boldsymbol{\varepsilon})$ is:

$$\begin{aligned} \widehat{y}(\boldsymbol{x} + \boldsymbol{\varepsilon}) = \widehat{y}(\boldsymbol{x}) &+ \boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}) \\ &+ \frac{1}{2} \boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x}) \, \boldsymbol{\varepsilon} + o(\boldsymbol{\varepsilon}^3). \end{aligned}$$

Therefore:

$$\begin{aligned} \widetilde{J} &\approx \mathbb{E}\Big(\big(\widehat{y}(\boldsymbol{x}) + \boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}) + \frac{1}{2} \boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x}) \, \boldsymbol{\varepsilon}\big)^2\Big) \\ &- 2\mathbb{E}\Big(y\widehat{y}(\boldsymbol{x}) + y\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}) + \frac{1}{2} y\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x}) \, \boldsymbol{\varepsilon}\Big) \\ &+ \mathbb{E}(y^2) \\ &= \mathbb{E}\Big(\widehat{y}(\boldsymbol{x})^2 + y^2 - 2y\widehat{y}(\boldsymbol{x})\Big) - 2\mathbb{E}\Big(\frac{1}{2} y\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x}) \, \boldsymbol{\varepsilon}\Big) \\ &+ \mathbb{E}\Big(\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon} + (\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}))^2 + o(\boldsymbol{\varepsilon}^3)\Big). \end{aligned}$$

The first term, $\mathbb{E}(\widehat{y}(\boldsymbol{x})^2 + y^2 - 2y\widehat{y}(\boldsymbol{x})) = \mathbb{E}((\widehat{y}(\boldsymbol{x}) - y)^2)$, is the loss function before adding the noise to the input,

according to Eq. (68). Also, because of $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$, we have $\mathbb{E}(\boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}) = \sigma^2$. As the noise and the input are independent, the following term is simplified as:

$$\begin{aligned} \mathbb{E}\Big((\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}))^2\Big) &\overset{\perp\!\!\!\perp}{=} \mathbb{E}(\boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}) \, \mathbb{E}(\|\nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\|_2^2) \\ &= \sigma^2 \, \mathbb{E}(\|\nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\|_2^2), \end{aligned}$$

and the rest of expression is simplified as:

$$\begin{aligned} &\mathbb{E}\Big(\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}\Big) - 2\mathbb{E}\Big(\frac{1}{2} y\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x}) \, \boldsymbol{\varepsilon}\Big) \\ &= \mathbb{E}\Big(\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}\Big) - \mathbb{E}\Big(y\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x}) \, \boldsymbol{\varepsilon}\Big) \\ &\overset{\perp\!\!\!\perp}{=} \mathbb{E}(\boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}) \, \mathbb{E}\big(\widehat{y}(\boldsymbol{x})\nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x})\big) - \mathbb{E}(\boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}) \, \mathbb{E}\big(y\nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x})\big) \\ &= \sigma^2 \, \mathbb{E}\Big(\big(\widehat{y}(\boldsymbol{x}) - y\big)\nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x})\Big). \end{aligned}$$

Hence, the overall loss function after noise injection to the input is simplified to:

$$\begin{aligned} \widetilde{J} &\approx J + \sigma^2 \, \mathbb{E}\Big(\big(\widehat{y}(\boldsymbol{x}) - y\big)\nabla_{\boldsymbol{x}}^2 \widehat{y}(\boldsymbol{x})\Big) \\ &+ \sigma^2 \, \mathbb{E}(\|\nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\|_2^2), \end{aligned} \tag{69}$$

which is a regularized optimization problem with $\ell_2$ norm penalty (see Eq. (50)). The penalty is on the second derivatives of outputs of neural network. This means that we do not want to have significant changes in the output of neural network. This penalization prevents from overfitting.

Note that the technique of adding noise to the input is also used in denoising autoencoders (Vincent et al., 2008). Moreover, an overcomplete autoencoder with one hidden layer (Goodfellow et al., 2016) (where the number of hidden neurons is greater than the dimension of data) needs a noisy input; otherwise, the mapping in the autoencoder will be just coping the input to output without learning a latent space.

It is also noteworthy that injecting noise to the weights of neural network (Goodfellow et al., 2016; Ho et al., 2008) can be interpreted similar to injecting noise to the input. Therefore, noise injection to the weights can also be interpreted as regularization where the regularization penalty term is $\sigma^2 \, \mathbb{E}(\|\nabla_{\boldsymbol{w}} \widehat{y}(\boldsymbol{x})\|_2^2)$ where $\boldsymbol{w}$ is the vector of weights (Goodfellow et al., 2016).

### 7.4.4. EARLY STOPPING IN NEURAL NETWORKS

As we mentioned in the explanations of Fig. 6-a, we train neural network up to a point where the overfitting is starting. This is referred to as *early stopping* (Prechelt, 1998; Yao et al., 2007) which helps avoid overfitting (Caruana et al., 2001).

According to Eq. (52), we have:

$$\nabla_{\boldsymbol{w}} \widehat{J}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*) + \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \overset{(51)}{=} \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*).$$

The gradient descent (with $\eta$ as the learning rate) used in back-propagation of neural network is (Boyd & Vandenberghe, 2004):

$$\boldsymbol{w}^{(t)} := \boldsymbol{w}^{(t-1)} - \eta \nabla_{\boldsymbol{w}} \widehat{J}(\boldsymbol{w}^{(t)})$$
$$= \boldsymbol{w}^{(t-1)} - \eta \boldsymbol{H}(\boldsymbol{w}^{(t-1)} - \boldsymbol{w}^*)$$
$$\implies \boldsymbol{w}^{(t)} - \boldsymbol{w}^* = (\boldsymbol{I} - \eta \boldsymbol{H})(\boldsymbol{w}^{(t-1)} - \boldsymbol{w}^*),$$

where $t$ is the index of iteration. According to Eq. (54), we have:

$$\boldsymbol{w}^{(t)} - \boldsymbol{w}^* = (\boldsymbol{I} - \eta \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top)(\boldsymbol{w}^{(t-1)} - \boldsymbol{w}^*).$$

Assuming the initial weights are $\boldsymbol{w}^{(0)} = 0$, we have:

$$\boldsymbol{w}^{(1)} - \boldsymbol{w}^* = -(\boldsymbol{I} - \eta \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top) \boldsymbol{w}^*$$
$$\implies \boldsymbol{w}^{(1)} = \big( \boldsymbol{I} - (\boldsymbol{I} - \eta \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top) \big) \boldsymbol{w}^*$$
$$\overset{(a)}{\implies} \boldsymbol{w}^{(1)} = \big( \boldsymbol{U} \boldsymbol{U}^\top - (\boldsymbol{U} \boldsymbol{U}^\top - \eta \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^\top) \big) \boldsymbol{w}^*$$
$$\implies \boldsymbol{w}^{(1)} = \boldsymbol{U} \big( \boldsymbol{I} - (\boldsymbol{I} - \eta \boldsymbol{\Lambda}) \big) \boldsymbol{U}^\top \boldsymbol{w}^*,$$

where $(a)$ is because $\boldsymbol{U}$ is a non-truncated orthogonal matrix so $\boldsymbol{U} \boldsymbol{U}^\top = \boldsymbol{I}$. By induction, we have:

$$\boldsymbol{w}^{(t)} = \boldsymbol{U} \big( \boldsymbol{I} - (\boldsymbol{I} - \eta \boldsymbol{\Lambda})^t \big) \boldsymbol{U}^\top \boldsymbol{w}^*,$$
$$\implies \boldsymbol{U}^\top \boldsymbol{w}^{(t)} = \boldsymbol{U}^\top \boldsymbol{U} \big( \boldsymbol{I} - (\boldsymbol{I} - \eta \boldsymbol{\Lambda})^t \big) \boldsymbol{U}^\top \boldsymbol{w}^*,$$
$$\overset{(a)}{\implies} \boldsymbol{U}^\top \boldsymbol{w}^{(t)} = \big( \boldsymbol{I} - (\boldsymbol{I} - \eta \boldsymbol{\Lambda})^t \big) \boldsymbol{U}^\top \boldsymbol{w}^*, \quad (70)$$

where $(a)$ is because $\boldsymbol{U}$ is an orthogonal matrix so $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$.

On the other hand, recall Eq. (67):

$$\boldsymbol{w}^\dagger = \boldsymbol{U}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \boldsymbol{U}^\top \boldsymbol{w}^*,$$
$$\implies \boldsymbol{U}^\top \boldsymbol{w}^\dagger = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \boldsymbol{U}^\top \boldsymbol{w}^*,$$
$$\overset{(a)}{\implies} \boldsymbol{U}^\top \boldsymbol{w}^\dagger = \big( \boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \alpha \big) \boldsymbol{U}^\top \boldsymbol{w}^*, \quad (71)$$

where $(a)$ is because of an expression rearrangement asserted in (Goodfellow et al., 2016). Comparing Eqs. (70) and (71) shows that early stopping can be seen as a $\ell_2$ norm regularization or weight decay (Goodfellow et al., 2016). Actually, the Eqs. (70) and (71) are equivalent if:

$$(\boldsymbol{I} - \eta \boldsymbol{\Lambda})^t = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \alpha, \quad (72)$$

for some $\eta$, $t$, and $\alpha$. If we take the logarithm from these expressions and use Taylor series expansion for $\log(1 + x)$, we have:

$$\log(\boldsymbol{I} - \eta \boldsymbol{\Lambda})^t = t \log(\boldsymbol{I} - \eta \boldsymbol{\Lambda})$$
$$\approx -t \, (\eta \boldsymbol{\Lambda} + \frac{1}{2} \eta^2 \boldsymbol{\Lambda}^2 + \frac{1}{3} \eta^3 \boldsymbol{\Lambda}^3 + \cdots), \quad (73)$$

$$\log(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \alpha = -\log(\boldsymbol{\Lambda} + \alpha \boldsymbol{I}) + \log \alpha$$
$$= -\log(\alpha(\boldsymbol{I} + \frac{1}{\alpha} \boldsymbol{\Lambda})) + \log \alpha$$
$$= -\log \alpha - \log(\boldsymbol{I} + \frac{1}{\alpha} \boldsymbol{\Lambda}) + \log \alpha$$
$$\approx \frac{-1}{\alpha} \boldsymbol{\Lambda} + \frac{1}{2\alpha^2} \boldsymbol{\Lambda}^2 - \frac{1}{3\alpha^3} \boldsymbol{\Lambda}^3 + \cdots. \quad (74)$$

Equating Eqs. (73) and (74) because of Eq. (72) gives us:

$$\alpha \approx \frac{1}{t \, \eta}, \quad t \approx \frac{1}{\alpha \, \eta}, \quad (75)$$

which shows that the inverse of number of iterations is proportional to the weight decay ($\ell_2$ norm) regularization parameter. In other words, the more training iterations we have, the less we are penalizing the weights and the more the network might get overfitted.

Moreover, some empirical studies (Zur et al., 2009) show that noise injection and weight decay have more effectiveness than early stopping for avoiding overfitting, although early stopping has its own merits.

## 8. Bagging

### 8.1. Definition

Bagging is short for Bootstrap AGGregatING, first proposed by (Breiman, 1996). It is a meta algorithm which can be used with any model (classifier, regression, etc).

The definition of *bootstrapping* is as follows. Suppose we have a sample $\{\boldsymbol{x}_i\}_{i=1}^n$ with size $n$ where $f(\boldsymbol{x})$ is the unknown distribution of the sample, i.e., $\boldsymbol{x}_i \overset{iid}{\sim} f(\boldsymbol{x})$. We would like to sample from this distribution but we do not know the $f(\boldsymbol{x})$. Approximating the sampling from the distribution by randomly sampling from the available sample is named bootstrapping. In bootstrapping, we use simple random sampling with replacement. The drawn sample is named *bootstrap sample*.

In bagging, we draw $k$ *bootstrap* samples each with some sample size. Then, we train the model $h_j$ using the $j$-th bootstrap sample, $\forall j \in \{1, \ldots, k\}$. Hence, we have $k$ trained models rather than one model. Finally, we *aggregate* the results of estimations of the $k$ models for an instance $\boldsymbol{x}$:

$$\widehat{f}(\boldsymbol{x}) = \frac{1}{k} \sum_{j=1}^{k} h_j(\boldsymbol{x}). \quad (76)$$

If the model is classifier, we should probably use sign function:

$$\widehat{f}(\boldsymbol{x}) = \text{sign}\big( \frac{1}{k} \sum_{j=1}^{k} h_j(\boldsymbol{x}) \big). \quad (77)$$

## 8.2. Theory

Let $e_j$ denote the error of the $j$-th model in estimation of the observation of an instance. Suppose this error is a random variable with normal distribution having mean zero, i.e., $e_j \overset{iid}{\sim} \mathcal{N}(0, s)$ where $s := \sigma^2$. We denote the covariance of estimations of two trained models using two different bootstrap samples by $c$. Therefore, we have:

$$\mathbb{E}(e_j^2) = s$$
$$\implies \mathbb{V}\mathrm{ar}(e_j) = \mathbb{E}(e_j^2) - (\mathbb{E}(e_j))^2 = s - 0 = s$$
$$\implies \mathbb{V}\mathrm{ar}(h_j(\boldsymbol{x})) = s, \tag{78}$$
$$\mathbb{E}(e_j e_\ell) = c$$
$$\implies \mathbb{C}\mathrm{ov}(e_j, e_\ell) = \mathbb{E}(e_j e_\ell) - \mathbb{E}(e_j)\mathbb{E}(e_\ell)$$
$$= c - (0 \times 0) = c \implies \mathbb{C}\mathrm{ov}(h_j(\boldsymbol{x}), h_\ell(\boldsymbol{x})) = c, \tag{79}$$

for all $j, \ell \in \{1, \dots, k\}, j \neq \ell$.

According to Eqs. (76), (78), and (79), we have:

$$\mathbb{V}\mathrm{ar}(\widehat{f}(\boldsymbol{x})) = \frac{1}{k^2} \mathbb{V}\mathrm{ar}\Big(\sum_{j=1}^{k} h_j(\boldsymbol{x})\Big)$$

$$\overset{(13)}{=} \frac{1}{k^2} \sum_{j=1}^{k} \mathbb{V}\mathrm{ar}(h_j(\boldsymbol{x}))$$

$$+ \frac{1}{k^2} \sum_{j=1}^{k} \sum_{\ell=1, \ell \neq j}^{k} \mathbb{C}\mathrm{ov}(h_j(\boldsymbol{x}), h_\ell(\boldsymbol{x}))$$

$$= \frac{1}{k^2} k s + \frac{1}{k^2} k(k-1) c = \frac{1}{k} s + \frac{k-1}{k} c. \tag{80}$$

The obtained expression has an interesting interpretation: If two trained models with two different bootstrap samples are very correlated, we will have $c \approx s$, thus:

$$\lim_{c \to s} \mathbb{V}\mathrm{ar}(\widehat{f}(\boldsymbol{x})) = \frac{1}{k} s + \frac{k-1}{k} s = s, \tag{81}$$

and if the two trained models are very different (uncorrelated), we will have $c \approx 0$, hence:

$$\lim_{c \to 0} \mathbb{V}\mathrm{ar}(\widehat{f}(\boldsymbol{x})) = \frac{1}{k} s + \frac{k-1}{k} 0 = \frac{1}{k} s. \tag{82}$$

This means that if the trained models are very correlated in bagging, there is not any difference from using only one model; however, if we have different trained models, the variance of estimation improves significantly by the factor of $k$. This also implies that bagging never is destructive; it either is not effective or improves the estimation in terms of variance (Bühlmann & Yu, 2000; Breiman, 1996).

Figure 5 shows that the more complex model usually has more variance and less bias. This trade-off is shown in Fig. 6. Therefore, the more variance corresponds to over-fitting. As bagging helps decrease the variance of estimation, it helps prevent overfitting. Therefore, bagging is a

meta algorithm useful to have less variance and not to get overfitted (Breiman, 1998). Moreover, as also will be mentioned in Section 9.3.1, bagging can be seen as an *ensemble learning* method (Polikar, 2012) which is useful because of *model averaging* (Hoeting et al., 1999; Claeskens & Hjort, 2008).

## 8.3. Examples in Machine Learning: Random Forest and Dropout

### 8.3.1. RANDOM FOREST

One of the examples of using bagging in machine learning is *random forest* (Liaw & Wiener, 2002). In random forest, we train different models (trees) using different bootstrap samples (subsets of the training set). However, as the trees work similarly, they will be very correlated. Foe the already explained reason, this will not have a significant improvement from using one tree. Random forest addresses this issue by also sampling from the features (dimensions) of the bootstrap sample. This makes the trained trees very different and thus results in a noticeable improvement.

### 8.3.2. DROPOUT

Another example of bagging is *dropout* in neural networks (Srivastava et al., 2014). According to dropout, in every iteration of training phase, the neurons are randomly removed with probability $p = 0.5$, i.e., we sample from a Bernoulli distribution. This makes the training phase as training different neural networks as we have different models in bagging. In the test time, all the neurons are used but their output is multiplied by the $p$. This imitates the model averaging of bagging in Eq. (76). That is why dropout prevents neural network from overfitting. Another intuition of why dropout works is making the neural network sparse which is very effective because of principal of sparsity (Friedman et al., 2009; Tibshirani et al., 2015) or Occam's razor (Domingos, 1999) introduced before.

## 8.4. Examples in Computer Vision: HOG and SSD

### 8.4.1. HISTOGRAM OF ORIENTED GRADIENTS

An example of bagging is Histogram of Oriented Gradients (HOG) (Dalal & Triggs, 2005) used in computer vision, especially for human detection in images. In HOG, different cells or blocks are used each of which includes a histogram of gradients of a sub-region of image. Finally, using bagging, the histograms are combined into one histogram. The effectiveness of HOG is because of effectiveness of bagging.

### 8.4.2. SINGLE SHOT MULTI-BOX DETECTOR

Single Shot multi-box Detector (SSD) (Liu et al., 2016) is another usage of bagging in computer vision and object detection using deep neural networks (LeCun et al., 2015; Goodfellow et al., 2016). In SSD, a set of bounding boxes (i.e., the models in bagging) with different sizes are used

which are processed and learned using convolution layers in neural network. Some of the boxes are matched and their weighted summation is used as the loss function of the neural network to optimize.

## 9. Boosting

### 9.1. Definition

Boosting is a meta algorithm which can be used with any model (classifier, regression, etc). For binary classification, for example, if we use boosting with a classifier even slightly better than flipping a coin, we will have a strong classifier (we will explain the reason later in Section 9.3.1). Thus, we can say boosting makes the estimation or classification very strong. In other words, boosting addresses the question whether a strong classifier can be obtained from a set of weak classifiers (Kearns, 1988; Kearns & Valiant, 1994).

The idea of boosting is to learn $k$ models in a hierarchy where every model gives more attention (larger weight) to the instances misclassified (or estimated very badly) by the previous model. Figure 9 shows this hierarchy. Finally, the overall estimation or classification is a weighted summation (average) of the $k$ estimations. For an instance $\boldsymbol{x}$, we have:

$$\widehat{f}(\boldsymbol{x}) = \sum_{j=1}^{k} \alpha_j \, h_j(\boldsymbol{x}). \tag{83}$$

If the model is classifier, we should probably use sign function:

$$\widehat{f}(\boldsymbol{x}) = \text{sign}\Big(\sum_{j=1}^{k} \alpha_j h_j(\boldsymbol{x})\Big), \tag{84}$$

which is equivalent to *majority voting* among the trained classifiers.

Different methods have been proposed for boosting, one of the most well-known ones is AdaBoost (Adaptive Boosting) (Freund & Schapire, 1996). The algorithm of AdaBoost for binary classification is shown in Algorithm 3. In this algorithm, $L_j$ is the cost function minimized in the $j$-th model $h_j$, the $\mathbb{I}(.)$ is the indicator function which is one and zero if its condition is and is not satisfied, respectively, and $w_i$ is the weight associated to the $i$-th instance for weighting it as the input to the next layer of boosting. Here, we can have several cases which help us understand the interpretation of the AdaBoost algorithm:

- if an instance is correctly classified, the $\mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))$ is zero and thus the $w_i$ will be still $w_i$ without any change. This makes sense because the correctly classified instance should not gain a significant weight in the next layer of boosting.



training instances
$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$

*Figure 9.* The training phase in boosting $k$ models.

- if an instance is misclassified, the $\mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))$ is one. In this case, we can have two sub-cases:

  - If the classifier which classified that instance was a bad classifier, its cost would be like flipping a coin, i.e., $L_j = 0.5$. Therefore, we will have $\alpha_j = \log(1) = 0$ and again the $w_i$ will still be $w_i$ without any change. This makes sense because we cannot trust the bad classifier whether the instance is correctly or incorrectly classified and thus we should not make any decision based on that.

  - If the classifier which classified that instance was a good classifier, then we have $L_j = 0.5$ and as we also have $\mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i)) = 1$, the weight will change as $w_i := w_i \exp(\alpha_j)$. This also is intuitive because the previous model in the boosting was a good classifier and we can trust it and that good classifier could not classify the instance correctly. therefore, we should notice that instance more in the next model in the boosting hierarchy.

Note that the cost in AdaBoost is:

$$L_j = \frac{\sum_{i=1}^{n} w_i \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i}, \tag{85}$$

which makes sense because it gets larger if the observations of more instances are estimated incorrectly.

### 9.2. Theory Based on Additive Models

Additive models (Hastie & Tibshirani, 1986) can be used to explain why boosting works (Friedman et al., 2000; Rojas, 2009). In additive model, we map the data as $\boldsymbol{x} \mapsto$

---

> **1 Initialize** $w_i = 1/n, \forall i \in \{1, \dots, n\}$
> **2 for** $j$ *from* $1$ *to* $k$ **do**
> **3**   $\quad h_j(\boldsymbol{x}) = \arg\min L_j$
> **4**   $\quad \alpha_j = \log(\frac{1-L_j}{L_j})$
> **5**   $\quad w_i = w_i \exp\left(\alpha_j \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))\right)$

**Algorithm 3:** The AdaBoost Algorithm

$\phi_j(\boldsymbol{x}), \forall j \in \{1, \dots, k\}$ and then add them using some weights $\beta_j$'s:

$$\phi(\boldsymbol{x}) = \sum_{j=1}^{k} \beta_j \, \phi_j(\boldsymbol{x}). \tag{86}$$

A well-known example of the additive model is Radial Basis Function (RBF) neural network (here with $k$ hidden nodes) which uses Gaussian mappings (Broomhead & Lowe, 1988; Schwenker et al., 2001).

Now, consider a cost function for an instance as:

$$L(y, h(\boldsymbol{x})) := \exp(-y \, h(\boldsymbol{x})), \tag{87}$$

where $y$ is the observation or label for $\boldsymbol{x}$ and $h(\boldsymbol{x})$ is the model's estimation of $y$. This cost is intuitive because when the instance is misclassified, the signs of $y$ and $h(\boldsymbol{x})$ will be different and the cost will be large, while in case of correct classification, the signs are similar and the cost is small. If we add up the cost over the $n$ training instances, we have:

$$L_t(y, h(\boldsymbol{x})) := \sum_{i=1}^{n} \exp(-y_i \, h(\boldsymbol{x}_i)), \tag{88}$$

where $L_t$ denotes the total cost.

In Eq. (86), if we rename the mapping to $h(\boldsymbol{x})$, which is the model used in boosting, we will have:

$$h(\boldsymbol{x}) = \sum_{j=1}^{k} \beta_j \, h_j(\boldsymbol{x}). \tag{89}$$

We can write this expression as a *forward stage-wise additive model* (Friedman et al., 2000; Rojas, 2009) in which we work with the models one by one where we add up the previously worked models:

$$f_{q-1}(\boldsymbol{x}) = \sum_{j=1}^{q-1} \beta_j \, h_j(\boldsymbol{x}), \tag{90}$$

$$f_q(\boldsymbol{x}) = f_{q-1}(\boldsymbol{x}) + \beta_q \, h_q(\boldsymbol{x}), \quad q \leq k, \tag{91}$$

where $h(\boldsymbol{x}) = f_k(\boldsymbol{x})$. Therefore, minimizing the cost, i.e.,

Eq. (88), for the $j$-th model in the additive manner is:

$$\min_{\beta_j, h_j} \sum_{i=1}^{n} \exp\left(-y_i \left[f_{j-1}(\boldsymbol{x}_i) + \beta_j \, h_j(\boldsymbol{x}_i)\right]\right)$$
$$= \min_{\beta_j, h_j} \sum_{i=1}^{n} \exp(-y_i \, f_{j-1}(\boldsymbol{x}_i)) \exp(-y_i \, \beta_j \, h_j(\boldsymbol{x}_i)).$$

The first term is a constant with respect to $\beta_j$ and $h_j$ so we name it by $w_i$:

$$w_i := \exp(-y_i \, f_{j-1}(\boldsymbol{x}_i)). \tag{92}$$

Thus:

$$\min_{\beta_j, h_j} \sum_{i=1}^{n} w_i \exp(-y_i \, \beta_j \, h_j(\boldsymbol{x}_i)).$$

As in binary AdaBoost, we have $\pm 1$ for $y_i$ and $h_j$, we can say:

$$\min_{\beta_j, h_j} \exp(-\beta_j) \sum_{i=1}^{n} w_i \, \mathbb{I}(y_i = h_j(\boldsymbol{x}_i))$$
$$+ \exp(\beta_j) \sum_{i=1}^{n} w_i \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))$$
$$\overset{(a)}{=} \min_{\beta_j, h_j} \exp(-\beta_j) \sum_{i=1}^{n} w_i$$
$$- \exp(-\beta_j) \sum_{i=1}^{n} w_i \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))$$
$$+ \exp(\beta_j) \sum_{i=1}^{n} w_i \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i)),$$

where $(a)$ is because:

$$\sum_{i=1}^{n} w_i \, \mathbb{I}(y_i = h_j(\boldsymbol{x}_i)) = \sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i)).$$

For the sake of minimization, we take the derivative:

$$\frac{\partial L_t}{\partial \beta_j} = -\exp(-\beta_j) \sum_{i=1}^{n} w_i$$
$$+ \exp(-\beta_j) \sum_{i=1}^{n} w_i \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))$$
$$+ \exp(\beta_j) \sum_{i=1}^{n} w_i \, \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i)) \overset{\text{set}}{=} 0,$$

which gives:

$$\implies (\exp(-\beta_j) + \exp(\beta_j)) \times$$
$$\frac{\mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))}{\sum_{i=1}^n w_i} = \exp(-\beta_j)$$
$$\overset{(85)}{\implies} (\exp(-\beta_j) + \exp(\beta_j)) L_j = \exp(-\beta_j)$$
$$\implies L_j = \frac{\exp(-\beta_j)}{\exp(-\beta_j) + \exp(\beta_j)}$$
$$\implies \exp(2\beta_j) = \frac{1 - L_j}{L_j} \implies 2\beta = \log(\frac{1 - L_j}{L_j})$$
$$\overset{(a)}{\implies} \alpha_j = 2\beta_j, \tag{93}$$

where $(a)$ is because of the line 4 in Algorithm 3. According to Eqs. (90), (91), and (92), we have:

$$w_i := w_i \exp(-y_i \beta_j h_j(\boldsymbol{x}_i)). \tag{94}$$

As we have $y_i h_j(\boldsymbol{x}_i) = \pm 1$, we can say:

$$-y_i h_j(\boldsymbol{x}_i) = 2 \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i)) - 1. \tag{95}$$

According Eqs. (93), (94), and (95), we have:

$$w_i := w_i \exp(\alpha_j \mathbb{I}(y_i \neq h(\boldsymbol{x}_i))) \exp(-\beta_j), \tag{96}$$

which is equivalent to the line 5 in Algorithm 3 with a factor of $\exp(-\beta_j)$. This factor does not have impact on whether the instance is correctly classified or not.

## 9.3. Theory Based on Maximum Margin

### 9.3.1. UPPER BOUND ON THE GENERALIZATION ERROR OF BOOSTING

There is an upper bound on the generalization error of boosting (Schapire et al., 1998). In binary boosting, we have $\pm 1$ for $y_i$ and also the sign of $\widehat{f}(\boldsymbol{x}_i)$ is important; therefore, $y_i \widehat{f}(\boldsymbol{x}_i) < 0$ means that we have error for estimating the $i$-th instance. Thus, for an error, we have:

$$y_i \widehat{f}(\boldsymbol{x}_i) \leq \theta, \tag{97}$$

for a $\theta > 0$. Recall the Eq. (83). We can normalize this equation because the sign of it is important:

$$\widehat{f}(\boldsymbol{x}_i) = \frac{\sum_{j=1}^k \alpha_j h_j(\boldsymbol{x}_i)}{\sum_{j=1}^k \alpha_j}. \tag{98}$$

According to Eqs. (97) and (98), we have:

$$y_i \widehat{f}(\boldsymbol{x}_i) \leq \theta \iff y_i \sum_{j=1}^k \alpha_j h_j(\boldsymbol{x}_i) \leq \theta \sum_{j=1}^k \alpha_j$$
$$\iff \exp\left(-y_i \sum_{j=1}^k \alpha_j h_j(\boldsymbol{x}_i) + \theta \sum_{j=1}^k \alpha_j\right) \geq 1.$$

Therefore, in terms of probability, we have:

$$\mathbb{P}(y_i \widehat{f}(\boldsymbol{x}_i) \leq \theta)$$
$$= \mathbb{P}\left(\exp\left(-y_i \sum_{j=1}^k \alpha_j h_j(\boldsymbol{x}_i) + \theta \sum_{j=1}^k \alpha_j\right) \geq 1\right). \tag{99}$$

According to the Markov's inequality which is (for $a > 0$ and a random variable $X$):

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}(X)}{a}, \tag{100}$$

and Eq. (99), we have (take $a = 1$ and the exponential term as $X$ in Markov's inequality):

$$\mathbb{P}(y_i \widehat{f}(\boldsymbol{x}_i) \leq \theta)$$
$$\leq \mathbb{E}\left(\exp\left(-y_i \sum_{j=1}^k \alpha_j h_j(\boldsymbol{x}_i) + \theta \sum_{j=1}^k \alpha_j\right)\right)$$
$$\overset{(a)}{=} \exp\left(\theta \sum_{j=1}^k \alpha_j\right) \mathbb{E}\left(\exp\left(-y_i \sum_{j=1}^k \alpha_j h_j(\boldsymbol{x}_i)\right)\right)$$
$$\overset{(b)}{=} \frac{1}{n} \exp\left(\theta \sum_{j=1}^k \alpha_j\right) \sum_{i=1}^n \exp\left(-y_i \sum_{j=1}^k \alpha_j h_j(\boldsymbol{x}_i)\right), \tag{101}$$

where $(a)$ is because the expectation is with respect to the data, i.e., $\boldsymbol{x}_i$ and $y_i$ and $(b)$ is according to definition of expectation.

Recall the line 5 in Algorithm 3:

$$w_i^{(j+1)} = w_i^{(j)} \exp(\alpha_j \mathbb{I}(y_i \neq h_j(\boldsymbol{x}_i))),$$

which can be restated as:

$$w_i^{(j+1)} = w_i^{(j)} \exp(-y_i \alpha_j h_j(\boldsymbol{x}_i)),$$

because $y_i = \pm 1$ and $h_j(\boldsymbol{x}_i) = \pm 1$. It is not harmful to AdaBoost if we use the normalized weights:

$$w_i^{(j+1)} = \frac{w_i^{(j)} \exp(-y_i \alpha_j h_j(\boldsymbol{x}_i))}{z_j}, \tag{102}$$

where:

$$z_j := \sum_{i=1}^n w_i^{(j)} \exp(-y_i \alpha_j h_j(\boldsymbol{x}_i)). \tag{103}$$

Considering that $w_i^{(1)} = 1/n$, we can have recursive ex-

pression for the weights:

$$w_i^{(k+1)} = \frac{w_i^{(k)} \exp\left(-y_i\,\alpha_k\,h_k(\boldsymbol{x}_i)\right)}{z_k}$$

$$= w_i^{(1)} \times \frac{1}{z_k \times \cdots \times z_1} \times$$
$$\exp\left(-y_i\,\alpha_k\,h_k(\boldsymbol{x}_i)\right) \times \cdots \times \exp\left(-y_i\,\alpha_1\,h_1(\boldsymbol{x}_i)\right)$$

$$= \frac{1}{n} \times \frac{1}{\prod_{j=1}^k z_j} \times \prod_{j=1}^k \exp\left(-y_i\,\alpha_j\,h_j(\boldsymbol{x}_i)\right)$$

$$= \frac{1}{n} \times \frac{1}{\prod_{j=1}^k z_j} \times \exp\left(-y_i\sum_{j=1}^k \alpha_j\,h_j(\boldsymbol{x}_i)\right). \quad (104)$$

We continue the Eq. (101):

$$\mathbb{P}(y_i\,\widehat{f}(\boldsymbol{x}_i) \le \theta)$$

$$\le \frac{1}{n} \exp\left(\theta\sum_{j=1}^k \alpha_j\right) \sum_{i=1}^n \exp\left(-y_i\sum_{j=1}^k \alpha_j\,h_j(\boldsymbol{x}_i)\right)$$

$$\overset{(104)}{=} \exp\left(\theta\sum_{j=1}^k \alpha_j\right) \left(\prod_{j=1}^k z_j\right) \sum_{i=1}^n w_i^{(k+1)}.$$

According to Eqs. (102) and (103), we have:

$$\sum_{i=1}^n w_i^{(j+1)} = \frac{\sum_{i=1}^n w_i^{(j)} \exp\left(-y_i\,\alpha_j\,h_j(\boldsymbol{x}_i)\right)}{\sum_{i=1}^n w_i^{(j)} \exp\left(-y_i\,\alpha_j\,h_j(\boldsymbol{x}_i)\right)} = 1.$$

Therefore:

$$\therefore \quad \mathbb{P}(y_i\,\widehat{f}(\boldsymbol{x}_i) \le \theta) \le \exp\left(\theta\sum_{j=1}^k \alpha_j\right) \left(\prod_{j=1}^k z_j\right). \quad (105)$$

On the other hand, according to Eq. (103), we have:

$$z_j = \sum_{i=1}^n w_i^{(j)} \exp\left(-y_i\,\alpha_j\,h_j(\boldsymbol{x}_i)\right)$$

$$= \sum_{i=1}^n w_i^{(j)} \exp(-\alpha_j)\,\mathbb{I}(y_i = h_j(\boldsymbol{x}_i))$$

$$+ \sum_{i=1}^n w_i^{(j)} \exp(\alpha_j)\,\mathbb{I}(y_i \ne h_j(\boldsymbol{x}_i))$$

$$= \exp(-\alpha_j) \sum_{i=1}^n w_i^{(j)}\mathbb{I}(y_i = h_j(\boldsymbol{x}_i))$$

$$+ \exp(\alpha_j) \sum_{i=1}^n w_i^{(j)}\mathbb{I}(y_i \ne h_j(\boldsymbol{x}_i)). \quad (106)$$

Recall Eq. (102) for $w_i^{j+1}$. This is in the range $[0,1]$ and its summation over error cases can be considered as the probability of error:

$$\sum_{i=1}^n w_i^{(j)}\mathbb{I}(y_i \ne h_j(\boldsymbol{x}_i)) = \mathbb{P}(y_i \ne h_j(\boldsymbol{x}_i)) \overset{(a)}{=} L_j, \quad (107)$$

where $(a)$ is because the Eq. (85) is the cost which is the probability of error. Therefore, the Eq. (106) becomes:

$$z_j = \exp(-\alpha_j)\,(1 - L_j) + \exp(\alpha_j)\,L_j.$$

Recall the $\alpha_j$ in line 4 in Algorithm 3. Scaling it is not harmful to AdaBoost:

$$\alpha_j = \frac{1}{2}\log\left(\frac{1 - L_j}{L_j}\right). \quad (108)$$

Therefore, we can have:

$$z_j = 2\sqrt{L_j(1 - L_j)}. \quad (109)$$

Plugging Eqs. (108) and (109) in Eq. (105) gives:

$$\mathbb{P}(y_i\,\widehat{f}(\boldsymbol{x}_i) \le \theta)$$

$$\le \exp\left(\frac{1}{2}\theta\sum_{j=1}^k \log\left(\frac{1 - L_j}{L_j}\right)\right) \left(2^k \prod_{j=1}^k \sqrt{L_j(1 - L_j)}\right)$$

$$= 2^k \exp\left(\sum_{j=1}^k \log\left(\left(\frac{1 - L_j}{L_j}\right)^{\theta/2}\right)\right) \prod_{j=1}^k \sqrt{L_j(1 - L_j)}$$

$$= 2^k \prod_{j=1}^k \exp\left(\log\left(\left(\frac{1 - L_j}{L_j}\right)^{\theta/2}\right)\right) \prod_{j=1}^k \sqrt{L_j(1 - L_j)},$$

which simplifies to the upper bound on the generalization error of AdaBoost (Schapire et al., 1998):

$$\mathbb{P}(y_i\,\widehat{f}(\boldsymbol{x}_i) \le \theta) \le 2^k \prod_{j=1}^k \sqrt{L_j^{1-\theta}(1 - L_j)^{1+\theta}}, \quad (110)$$

where $\mathbb{P}(y_i\,\widehat{f}(\boldsymbol{x}_i) \le \theta)$ is the probability that the generalization (true) error for the $i$-th instance is less than $\theta > 0$.

According to Eq. (85), we have $L_j \in [0,1]$. If we have $L_j \le 0.5 - \xi$, where $\xi \in (0, 0.5)$, the Eq. (110) becomes:

$$\mathbb{P}(y_i\,\widehat{f}(\boldsymbol{x}_i) \le \theta) \le \left(\sqrt{(1 - 2\xi)^{1-\theta}(1 + 2\xi)^{1+\theta}}\right)^k, \quad (111)$$

which is a very good upper bound because if $\theta < \xi$, we have $\sqrt{(1 - 2\xi)^{1-\theta}(1 + 2\xi)^{1+\theta}} < 1$; thus, the probability of error, $\mathbb{P}(y_i\,\widehat{f}(\boldsymbol{x}_i) \le \theta)$, decreases *exponentially* with $k$ which is the number of models used in boosting. This shows that boosting helps us reduce the generalization error and thus helps us avoid overfitting. In other words, because of the bound on generalization error, boosting overfits very hardly.

If $\xi$ is a very small positive number, the $L_j \le 0.5 - \xi$ is a little smaller than $0.5$, i.e., $L_j \lessapprox 0.5$. As we are discussing binary classification in boosting, $L_j = 0.5$ means random classification by flipping a coin. Therefore, for having the

great bound of Eq. (111), having weak base models (a little better than random decision) suffices. This shows the effectiveness of boosting. Note that a very small $\xi$ means a very small $\theta$ because of $\theta < \xi$; therefore, it means a very small probability of error because of $\mathbb{P}\big(y_i\,\widehat{f}(\boldsymbol{x}_i) \leq \theta\big)$.

It is noteworthy that both boosting and bagging can be seen as *ensemble learning* (Polikar, 2012) (or majority voting) methods which use *model averaging* (Hoeting et al., 1999; Claeskens & Hjort, 2008) and are very effective in learning theory. Moreover, both boosting and bagging reduce the variance of estimation (Breiman, 1998; Schapire et al., 1998), especially for the models with high variance of estimation such as trees (Quinlan, 1996).

In the above, we analyzed boosting for *binary* classification. A similar discussion can be done for *multi-class* classification in boosting and find an upper bound on the generalization error (see the appendix in (Schapire et al., 1998) for more details).

### 9.3.2. BOOSTING AS MAXIMUM MARGIN CLASSIFIER

In another perspective, the found upper bound for boosting shows that boosting can be seen as a method to increase (maximize) the margins of training error which results in a good generalization error (Boser et al., 1992). This phenomenon is the base for the theory of Support Vector Machines (SVM) (Cortes & Vapnik, 1995; Burges, 1998). In the following, we analyze the analogy between maximum margin classifier (i.e., SVM) and boosting (Schapire et al., 1998). In addition to (Schapire et al., 1998), some more discussion exist for upper bound and margin of boosting (Wang et al., 2008; Gao & Zhou, 2013) to which we refer the interested readers.

Assume we have training instances $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $y_i \in \{-1, +1\}$ for binary classification. The two classes may not be linearly separable. In order to handle this case, we map the data to higher dimensional feature space using kernels (Scholkopf & Smola, 2001; Hofmann et al., 2008), hoping that they become linearly separable in the feature space. Assume $\boldsymbol{h}(\boldsymbol{x})$ is a vector which non-linearly maps data to the feature space. Considering $\boldsymbol{\alpha}$ as the vector of dual variables, the dual optimization problem (Boyd & Vandenberghe, 2004) in SVM is (Burges, 1998; Schapire et al., 1998):

$$\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad \underset{\{(\boldsymbol{x}_i,y_i)\}_{i=1}^n}{\text{minimize}} \quad \frac{y_i\,(\boldsymbol{\alpha}^\top \boldsymbol{h}(\boldsymbol{x}_i))}{||\boldsymbol{\alpha}||_2}. \tag{112}$$

Note that $y_i = \pm 1$ and $\boldsymbol{\alpha}^\top \boldsymbol{h}(\boldsymbol{x}_i) \gtrless 0$; therefore, the sign of $y_i\,(\boldsymbol{\alpha}^\top \boldsymbol{h}(\boldsymbol{x}_i))$ determines the class of the $i$-th instance. On the other hand, the Eq. (98) can be written in a vector form:

$$\widehat{f}(\boldsymbol{x}_i) = \frac{\sum_{j=1}^k \alpha_j\, h_j(\boldsymbol{x}_i)}{\sum_{j=1}^k \alpha_j} = \frac{\boldsymbol{\alpha}^\top \boldsymbol{h}(\boldsymbol{x}_i)}{||\boldsymbol{\alpha}||_1}, \tag{113}$$

where $\boldsymbol{h}(\boldsymbol{x}_i) = [h_1(\boldsymbol{x}_i), \ldots, h_k(\boldsymbol{x}_i)]^\top$ and $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_k]^\top$. Note that here, $h(\boldsymbol{x}_i) = \pm 1$ and $\alpha_j$ is obtained from Eq. (108) or the line 4 in Algorithm 3.

The similarity between the Eq. (113) and the cost function in Eq. (112) shows that boosting can be seen as maximizing the margin of classification resulting in a good generalization error (Schapire et al., 1998). In other words, finding a linear combination in the high dimensional feature space having a large margin between the training instances of the classes is performed in the two methods. Note that a slight difference is the type of norm which is interpretable because the mapping to feature space in boosting is only to $h(\boldsymbol{x}_i) = \pm 1$ while in SVM, it can be any number where the sign is important. Therefore, $\ell_1$ and $\ell_2$ norms are suitable in boosting and SVM, respectively (Schapire et al., 1998).

Another connection between SVM (maximum margin classifier) and boosting is that some of the training instances are found to be most important instances, called *support vectors* (Burges, 1998). In boosting, also, weighting the training instances can be seen as selecting some *informative* models (Freund, 1995) which can be analogous to support vectors.

### 9.4. Examples in Machine Learning: Boosting Trees and SVMs

Both bagging and boosting are used a lot with trees (Quinlan, 1996; Friedman et al., 2009). The reason why boosting is very effective with trees is that trees have a large variance of estimation where the boosting and bagging can significantly reduce the variance of estimation as discussed before. Note that boosting is also used with SVM for imbalanced data (Wang & Japkowicz, 2010).

## 10. Conclusion

This paper was a tutorial paper introducing overfitting, cross validation, generalized cross validation, regularization, bagging, and boosting. The theory behind these methods were explained and some examples of them in machine learning and computer vision were provided.

## Acknowledgment

---

[2]See `https : / / www . youtube . com / watch ? v =`

his. Moreover some parts of Section 7 have been discussed in the books (Friedman et al., 2009) and (Goodfellow et al., 2016).

## A. Proof of Stein's Lemma

As the components of $\boldsymbol{z} = [z_1, \dots, z_d]^\top \in \mathbb{R}^d$ are independent random variables with normal distribution, i.e., $z_i \sim \mathcal{N}(\mu_i, \sigma)$, we have:

$$
\begin{aligned}
f(\boldsymbol{z}) = f(z_1, \dots, z_d) &\overset{(a)}{=} f(z_1) \times \cdots \times f(z_d) \\
&= \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(z_i - \mu_i)^2}{2\sigma^2}) \\
&= \frac{1}{\sqrt{(2\pi\sigma^2)^d}} \exp(-\frac{\sum_{i=1}^{d}(z_i - \mu_i)^2}{2\sigma^2}) \\
&= \frac{1}{\sqrt{(2\pi\sigma^2)^d}} \exp(-\frac{||\boldsymbol{z} - \boldsymbol{\mu}||_2^2}{2\sigma^2}),
\end{aligned}
$$

where $(a)$ is because $z_1 \perp\!\!\!\perp \dots \perp\!\!\!\perp z_d$.
We also have:

$$
(\boldsymbol{z} - \boldsymbol{\mu})^\top \boldsymbol{g}(\boldsymbol{z}) = \sum_{i=1}^{d}(z_i - \mu_i) g_i.
$$

According to the definition of expectation, we have:

$$
\begin{aligned}
\mathbb{E}\big((\boldsymbol{z} - \boldsymbol{\mu})^\top \boldsymbol{g}(\boldsymbol{z})\big) &= \int_{\mathbb{R}^d} f(\boldsymbol{z})(\boldsymbol{z} - \boldsymbol{\mu})^\top \boldsymbol{g}(\boldsymbol{z}) \, d\boldsymbol{z} \\
&= \int_{\mathbb{R}^d} \frac{1}{\sqrt{(2\pi\sigma^2)^d}} \exp(-\frac{||\boldsymbol{z} - \boldsymbol{\mu}||_2^2}{2\sigma^2}) \sum_{i=1}^{d}(z_i - \mu_i) g_i \\
&\qquad\qquad\qquad\qquad\qquad\qquad dz_1 \dots dz_d
\end{aligned}
$$

$$
\overset{(a)}{=} \sigma^2 \sum_{i=1}^{d} \int_{\mathbb{R}^d} \frac{1}{\sqrt{(2\pi\sigma^2)^d}} \exp(-\frac{||\boldsymbol{z} - \boldsymbol{\mu}||_2^2}{2\sigma^2}) \frac{\partial g_i}{\partial z_i} \\
\qquad\qquad\qquad\qquad\qquad\qquad dz_1 \dots dz_d
$$

$$
\overset{(b)}{=} \sigma^2 \sum_{i=1}^{d} \mathbb{E}(\frac{\partial g_i}{\partial z_i}),
$$

where $(a)$ uses integration by parts and $(b)$ is according to the definition of expectation. Q.E.D.

## References

Arlot, Sylvain and Celisse, Alain. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

Bach, Francis, Jenatton, Rodolphe, Mairal, Julien, and Obozinski, Guillaume. Convex optimization with sparsity-inducing norms. *Optimization for Machine Learning*, 5:19–53, 2011.

Bach, Francis, Jenatton, Rodolphe, Mairal, Julien, and Obozinski, Guillaume. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.

Barnett, Vick. *Elements of sampling theory*. English Universities Press, London, 1974.

Boser, Bernhard E, Guyon, Isabelle M, and Vapnik, Vladimir N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152. ACM, 1992.

Boyd, Stephen and Vandenberghe, Lieven. *Convex optimization*. Cambridge university press, 2004.

Breiman, Leo. Bagging predictors. *Machine learning*, 24 (2):123–140, 1996.

Breiman, Leo. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3): 801–849, 1998.

Broomhead, D. S. and Lowe, David. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

Bühlmann, Peter Lukas and Yu, Bin. Explaining bagging. In *Research report/Seminar für Statistik, Eidgenössische Technische Hochschule Zürich*, volume 92. Seminar für Statistik, Eidgenössische Technische Hochschule (ETH), 2000.

Burges, Christopher JC. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

Caruana, Rich, Lawrence, Steve, and Giles, C Lee. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pp. 402–408, 2001.

Carven, P and Wahba, G. Smoothing noisy data with spline functions: estimationg the correct degree of smoothing by the method of generalized cross-validation. *Numer. Math*, 31:377–403, 1979.

Casella, George and George, Edward I. Explaining the Gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

Chang, Yale. $L_{2,1}$ norm and its applications. *Technical Report, University of Central Florida*.

Chiu, Ching-Tai, Mehrotra, Kishan, Mohan, Chilukuri K, and Ranka, Sanjay. Modifying training algorithms for improved fault tolerance. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pp. 333–338. IEEE, 1994.

Claeskens, Gerda and Hjort, Nils Lid. *Model selection and model averaging*. Cambridge Books, Cambridge University Press, 2008.

Cortes, Corinna and Vapnik, Vladimir. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Dalal, Navneet and Triggs, Bill. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pp. 886–893. IEEE Computer Society, 2005.

DeVries, Terrance and Taylor, Graham W. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

Domingos, Pedro. The role of Occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4): 409–425, 1999.

Donoho, David L. For most large underdetermined systems of linear equations the minimal $\ell_1$-norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(6):797–829, 2006.

Freund, Yoav. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

Freund, Yoav and Schapire, Robert E. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148—-156. Morgan Kaufman, San Francisco, 1996.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert. Additive logistic regression: a statistical view of boosting. *The annals of statistics*, 28(2):337–407, 2000.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert. *The elements of statistical learning: Data Mining, Inference, and Prediction*, volume 2. Springer series in statistics, New York, NY, USA, 2009.

Gao, Wei and Zhou, Zhi-Hua. On the doubt about margin explanation of boosting. *Artificial Intelligence*, 203:1–18, 2013.

Ghodsi, Ali. Classification course, department of statistics and actuarial science, university of Waterloo. Online Youtube Videos, 2015a. Accessed: January 2019.

Ghodsi, Ali. Deep learning course, department of statistics and actuarial science, university of Waterloo. Online Youtube Videos, 2015b. Accessed: January 2019.

Ghojogh, Benyamin, Nekoei, Hadi, Ghojogh, Aydin, Karray, Fakhri, and Crowley, Mark. Sampling algorithms, from survey sampling to monte carlo methods: Tutorial and literature review. *arXiv preprint arXiv:2011.00901*, 2020.

Golub, Gene H, Heath, Michael, and Wahba, Grace. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep learning*. MIT press, 2016.

Grandvalet, Yves, Canu, Stéphane, and Boucheron, Stéphane. Noise injection: Theoretical prospects. *Neural Computation*, 9(5):1093–1108, 1997.

Hastie, Trevor J and Tibshirani, Robert. Generalized additive models. *Statistical Science*, 1(3):297–318, 1986.

Ho, Kevin, Leung, Chi-sing, and Sum, John. On weight-noise-injection training. In *International Conference on Neural Information Processing*, pp. 919–926. Springer, 2008.

Hoeting, Jennifer A, Madigan, David, Raftery, Adrian E, and Volinsky, Chris T. Bayesian model averaging: a tutorial. *Statistical science*, pp. 382–401, 1999.

Hofmann, Thomas, Schölkopf, Bernhard, and Smola, Alexander J. Kernel methods in machine learning. *The annals of statistics*, pp. 1171–1220, 2008.

Kearns, Michael. Thoughts on hypothesis boosting. *Technical Report, Machine Learning class project*, pp. 1–9, 1988.

Kearns, Michael and Valiant, Leslie. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

Kong, Eun Bae and Dietterich, Thomas G. Error-correcting output coding corrects bias and variance. In *Machine Learning Proceedings 1995*, pp. 313–321. Elsevier, 1995.

Krogh, Anders and Hertz, John A. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pp. 950–957, 1992.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *nature*, 521(7553):436, 2015.

Li, Ker-Chau. From Stein's unbiased risk estimates to the method of generalized cross validation. *The Annals of Statistics*, 13(4):1352–1377, 1985.

Liaw, Andy and Wiener, Matthew. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Liu, Wei, Anguelov, Dragomir, Erhan, Dumitru, Szegedy, Christian, Reed, Scott, Fu, Cheng-Yang, and Berg, Alexander C. SSD: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Matsuoka, Kiyotoshi. Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440, 1992.

Mitchell, Thomas. *Machine learning*. McGraw Hill Higher Education, 1997.

Parikh, Neal and Boyd, Stephen. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

Polikar, Robi. Ensemble learning. In *Ensemble machine learning*, pp. 1–34. Springer, 2012.

Prechelt, Lutz. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pp. 55–69. Springer, 1998.

Quinlan, J Ross. Bagging, boosting, and c4.5. In *AAAI/IAAI Conference*, volume 1, pp. 725–730, 1996.

Rojas, Raúl. Adaboost and the super bowl of classifiers: a tutorial introduction to adaptive boosting. *Freie University, Berlin, Technical Report*, 2009.

Schapire, Robert E, Freund, Yoav, Bartlett, Peter, and Lee, Wee Sun. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.

Schmidt, Mark. Least squares optimization with l1-norm regularization. *CS542B Project Report*, 504:195–221, 2005.

Scholkopf, Bernhard and Smola, Alexander J. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

Schwenker, Friedhelm, Kestler, Hans A, and Palm, Günther. Three learning phases for radial-basis-function networks. *Neural networks*, 14(4-5):439–458, 2001.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Stein, Charles M. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pp. 1135–1151, 1981.

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

Tibshirani, Robert, Wainwright, Martin, and Hastie, Trevor. *Statistical learning with sparsity: the lasso and generalizations*. Chapman and Hall/CRC, 2015.

Van Dyk, David A and Meng, Xiao-Li. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Wang, Benjamin X and Japkowicz, Nathalie. Boosting support vector machines for imbalanced data sets. *Knowledge and information systems*, 25(1):1–20, 2010.

Wang, Liwei, Sugiyama, Masashi, Yang, Cheng, Zhou, Zhi-Hua, and Feng, Jufu. On the margin explanation of boosting algorithms. In *COLT*, pp. 479–490. Citeseer, 2008.

Wright, Stephen J. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

Wu, Tong Tong and Lange, Kenneth. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.

Yao, Yuan, Rosasco, Lorenzo, and Caponnetto, Andrea. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

Zhang, Harry. The optimality of naive Bayes. In *American Association for Artificial Intelligence (AAAI)*, 2004.

Zur, Richard M, Jiang, Yulei, Pesce, Lorenzo L, and Drukker, Karen. Noise injection for training artificial neural networks: A comparison with weight decay and early stopping. *Medical physics*, 36(10):4810–4818, 2009.