Machine Learning Tutorial

Wei-Lun Chao, weilunchao760414@gmail.com
First edited in December, 2011
DISP Lab, Graduate Institute of Communication Engineering, National Taiwan University

Contents

Notation

General notation:

a : scalar
$\boldsymbol{a}$: vector
A: matrix
a_i : the ith entry of $\boldsymbol{a}$
a_{ij} : the entry (i, j) of A
$\boldsymbol{a}^{(n)}$: the nth vector $\boldsymbol{a}$ in a dataset
$A^{(n)}$: the nth matrix A in a dataset
$\boldsymbol{b}_k$: the vector corresponding to the kth class in a dataset (or kth component in a model)
B_k : the matrix corresponding to the kth class in a dataset (or kth component in a model)
$\boldsymbol{b}_k^{(i)}$: the ith vector of the kth class in a dataset
$|A|$, $|A^{(n)}|$, $|B_k|$: the number of column vectors in A, $A^{(n)}$, and B_k

Special notation:

***** In some conditions, some special notations will be used and desribed at those places.
Ex: $\boldsymbol{b}_k$ denotes a k-dimensional vector, and $B_{k\times j}$ denotes a $k \times j$ matrix

1. Introduction

In the topics of face recognition, face detection, and facial age estimation, machine learning plays an important role and is served as the fundamental technique in many existing literatures.

For example, in face recognition, many researchers focus on using dimensionality reduction techniques for extracting personal features. The most well-known ones are (1) eigenfaces [1], which is based on principal component analysis (PCA,) and (2) fisherfaces [2], which is based on linear discriminant analysis (LDA).

In face detection, the popular and efficient technique based on Adaboost cascade structure [3][4], which drastically reduces the detection time while maintains comparable accuracy, has made itself available in practical usage. Based on our knowledge, this technique is the basis of automatic face focusing in digital cameras. Machine learning techniques are also widely used in facial age estimation to extract the hardly found features and to build the mapping from the facial features to the predicted age.

Although machine learning is not the only method in pattern recognition (for example, there are still many researches aiming to extract useful features through image and video analysis), it could provide some theoretical analysis and practical guidelines to refine and improve the recognition performance. In addition, with the fast development of technology and the burst usage of Internet, now people can easily take, make, and access lots of digital photos and videos either by their own digital cameras or from popular on-line photo and video collections such as Flicker [5], Facebook [6], and Youtube [7]. Based on the large amount of available data and the intrinsic ability to learn knowledge from data, we believe that the machine learning techniques will attract much more attention in pattern recognition, data mining, and information retrieval.

In this tutorial, a brief but broad overview of machine learning is given, both in theoretical and practical aspects. In Section 2, we describe what machine learning is and its availability. In Section 3, the basic concepts of machine learning are presented, including categorization and learning criteria. The principles and effects about the learning performance are discussed in Section 4, and several supervised and unsupervised learning algorithms are introduced in Sections 5 and 6. In Section 7, a general framework of pattern recognition based on machine learning technique is provided. Finally, in Section 8, we give a conclusion.

2. What is Machine Learning?

"Optimizing a performance criterion using example data and past experience", said by E. Alpaydin [8], gives an easy but faithful description about machine learning. In machine learning, data plays an indispensable role, and the learning algorithm is used to discover and learn knowledge or properties from the data. The quality or quantity of the dataset will affect the learning and prediction performance. The textbook (have not been published yet) written by Professor Hsuan-Tien Lin, the machine learning course instructor in National Taiwan University (NTU), is also titled as "Learning from Data", which emphasizes the importance of data in machine learning. Fig. 1 shows an example of two-class dataset.

2.1 Notation of Dataset

Before going deeply into machine learning, we first describe the notation of dataset, which will be used through the whole section as well as the tutorial. There are two general dataset types. One is labeled and the other one is unlabeled:

- **Labeled dataset** $\mathbb{D}$**:** $X = \{\boldsymbol{x}^{(n)} \in R^d\}_{n=1}^N$, $Y = \{y^{(n)} \in R\}_{n=1}^N$
- **Unlabeled dataset** $\mathbb{D}$**:** $X = \{\boldsymbol{x}^{(n)} \in R^d\}_{n=1}^N$

, where X denotes the **feature set** containing N samples. Each sample is a d-dimensional vector $\boldsymbol{x}^{(n)} = [x_1^{(n)}, x_2^{(n)}, ..., x_d^{(n)}]^T$ and called a feature vector or feature sample, while each dimension of a vector is called an attribute, feature, variable, or element. Y stands for the **label set**, recording what label a feature vector corresponds to (the color assigned on each point in Fig. 1). In some applications, the label set is unobserved or ignored. Another form of labeled dataset is described as $\{\boldsymbol{x}^{(n)} \in R^d, y^{(n)} \in R\}_{n=1}^N$, where each $\{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}$ is called a data pair.

2.2 Training Set and Test Set

In machine learning, an unknown **universal dataset** is assumed to exist, which contains all the possible data pairs as well as their probability distribution of

appearance in the real world. While in real applications, what we observed is only a subset of the universal dataset due to the lack of memory or some other unavoidable reasons. This acquired dataset is called the **training set** (**training data**) and used to learn the properties and knowledge of the universal dataset. In general, **vectors in the training set are assumed independently and identically sampled (i.i.d) from the universal dataset**.

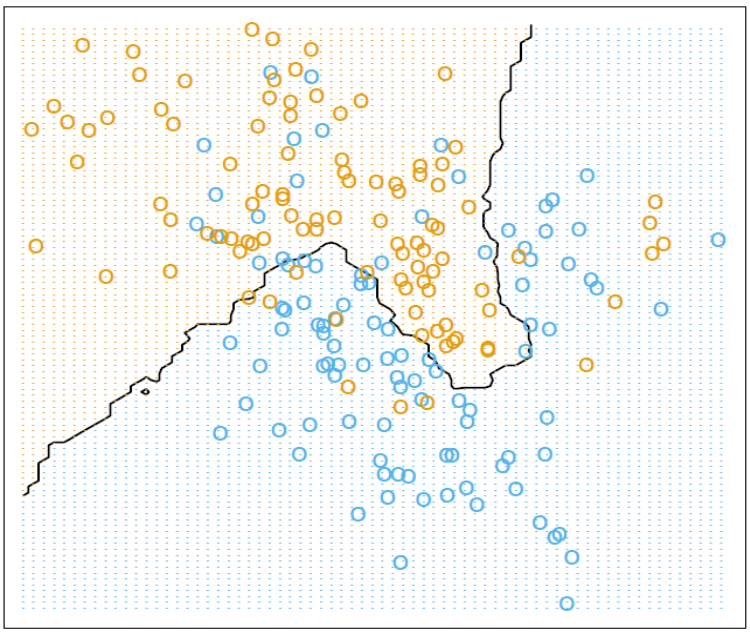


Fig. 1 An example of two-class dataset is showed, where two measurements of each sample are extracted. In this case, each sample is a 2-D vector [9].

In machine learning, what we desire is that these learned properties can not only explain the training set, but also be used to predict unseen samples or future events. In order to examine the performance of learning, another dataset may be reserved for testing, called the **test set** or **test data**. For example, before final exams, the teacher may give students several questions for practice (training set), and the way he judges the performances of students is to examine them with another problem set (test set). In order to distinguish the training set and the test set when they appear together, we use $\mathbb{D}$ and $\mathbb{D}$ to denote them, respectively.

We have not clearly discussed what kinds of properties can be learned from the dataset and how to estimate the learning performance, while the readers can just leave it as a black box and go forth. In Fig. 2, an explanation of the three datasets above is presented, and the first property a machine can learn in a labeled data set is shown, the separating boundary.

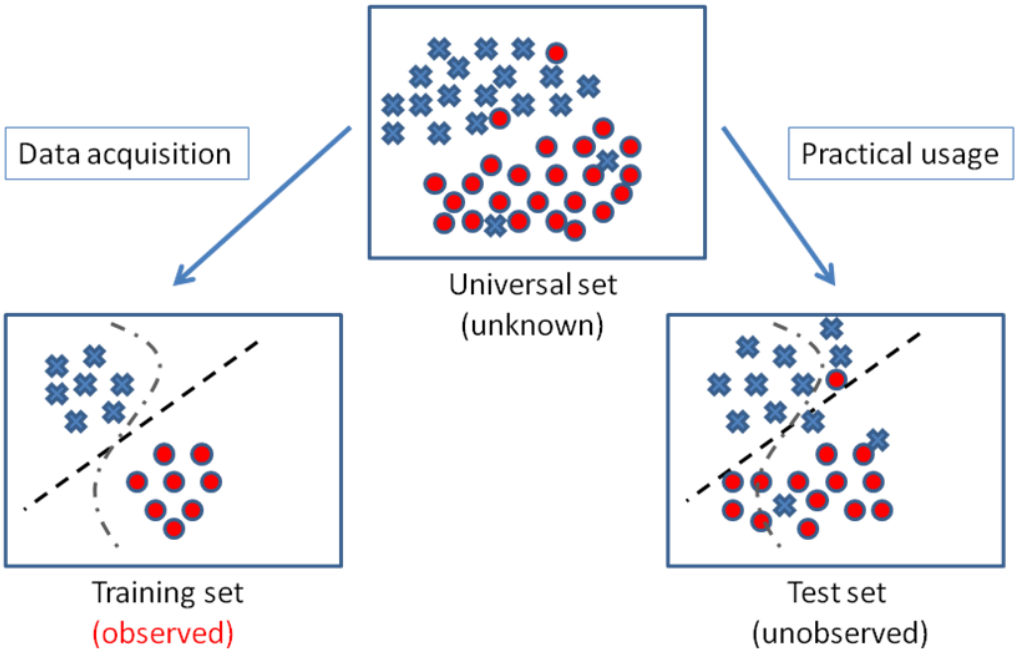


Fig. 2 An explanation of three labeled datasets. The universal set is assumed to exist but unknown, and through the data acquisition process, only a subset of universal set is observed and used for training (training set). Two learned separating lines (the first example of properties a machine can learn in this tutorial) are shown in both the training set and test set. As you can see, these two lines definitely give 100% accuracy on the training set, while they may perform differently in the test set (the curved line shows higher error rate).

2.3 No Free Lunch Rule

If the learned properties (which will be discussed later) can only explain the training set but not the test or universal set, then machine learning is infeasible. Fortunately, thanks to the Hoeffding inequality [10] presented below, the connection between the learned knowledge from the training set and its availability in the test set is described in a probabilistic way:

$$P[|\nu - \mu| > \varepsilon] \leq 2e^{-2\varepsilon^2 N}. \tag{1}$$

In this inequality, N denotes the size of training set, and ν and μ describe how the leaned properties perform in the training set and the test set. For example, if the learned property is a separating boundary, these two quantities usually correspond to the classification errors. Finally, ε is the tolerance gap between ν and μ. Details of the Hoeffding inequality are beyond the scope of this tutorial, and later an extended version of the inequality will be discussed.

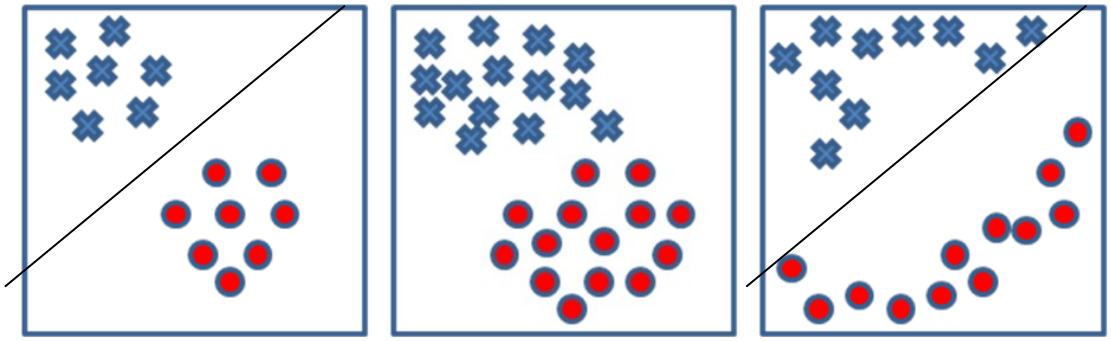


Fig. 3 The no free lunch rule for dataset: (a) is the training set we have, and (b), (c) are two test sets. As you can see, (c) has different sample distributions from (a) and (b), so we cannot expect that the properties learned from (a) to be useful in (c).

While (1) gives us the confidence on applying machine learning, there are some necessary rules to ensure its availability. These rules are called the "no free lunch rules" and are defined on both the dataset and the properties to learn. On the dataset, the no free lunch rules require the training set and the test set to come from the same distribution (same universal set). And on the properties, the no free lunch rules ask the users to make assumptions on what property to learn and how to model the property. For example, if the separating boundary in a labeled dataset is desired, we also need to define the type of the boundary (ex. a straight line or a curve). On the other hand, if we want to estimate the probability distribution of an unlabeled dataset, the distribution type should also be defined (ex. Gaussian distribution). Fig. 3 illustrates the no free lunch rules for dataset.

2.4 Relationships with Other Disciplines

Machine learning involves the techniques and basis from both statistics and computer science:

- **Statistics:** Learning and inference the statistical properties from given data
- **Computer science:** Efficient algorithms for optimization, model representation, and performance evaluation.

In addition to the importance of data set, machine learning is generally composed of the two critical factors, **modeling** and **optimization**. Modeling means how to model the separating boundary or probability distribution of the given training set, and then the optimization techniques are used to seek the best parameters of the chosen model.

Machine learning is also related to other disciplines such as artificial neural networks, pattern recognition, information retrieval, artificial intelligence, data mining, and function approximation, etc. Compared to those areas, machine learning focus more on why machine can learn and how to model, optimize, and regularize in order to make the best use of the accessible training data.

3. Basic Concepts and Ideals of Machine Learning

In this section, more details of machine learning will be presented, including the categorization of machine learning and what we can learn, the goals we are seeking, the structure of learning process, and the optimization criterion, etc. To begin with, a small warming up is given for readers to get clearer why we need to learn.

3.1 Designing versus Learning

In daily life, people are easily facing some decisions to make. For example, if the sky is cloudy, we may decide to bring an umbrella or not. For a machine to make these kinds of choices, the intuitive way is to model the problem into a mathematical expression. The mathematical expression could directly be designed from the problem background. For instance, the vending machine could use the standards and security decorations of currency to detect false money. While in some other problems that we can only acquire several measurements and the corresponding labels, but do not know the specific relationship among them, learning will be a better way to find the underlying connection.

Another great illustration to distinguish designing from learning is the image compression technique. JPEG, the most widely used image compression standard, exploits the block-based DCT to extract the spatial frequencies and then unequally quantizes each frequency component to achieve data compression. The success of using DCT comes from not only the image properties, but also the human visual perception. While without counting the side information, the KL transform (Karhunen-Loeve transform), which learns the best projection basis for a given image, has been proved to best reduce the redundancy [11]. In many literatures, the knowledge acquired from human understandings or the intrinsic factors of problems are called the **domain knowledge**. And the knowledge learned from a given training set is called the **data-driven knowledge**.

3.2 The Categorization of Machine Learning

There are generally three types of machine learning based on the ongoing problem and the given data set, (1) **supervised learning**, (2) **unsupervised learning**, and (3) **reinforcement learning**:

- **Supervised learning:** The training set given for supervised learning is the labeled dataset defined in Section 2.1. Supervised learning tries to **find the relationships between the feature set and the label set**, which is the knowledge and properties we can learn from labeled dataset. If each feature vector $\boldsymbol{x}$ is corresponding to a label $y \in L, L = \{l_1, l_2, ..., l_c\}$ (c is usually ranged from 2 to a hundred), the learning problem is denoted as **classification**. On the other hand, if each feature vector $\boldsymbol{x}$ is corresponding to a real value $y \in R$, the learning problem is defined as **regression** problem. The knowledge extracted from supervised learning is often utilized for prediction and recognition.

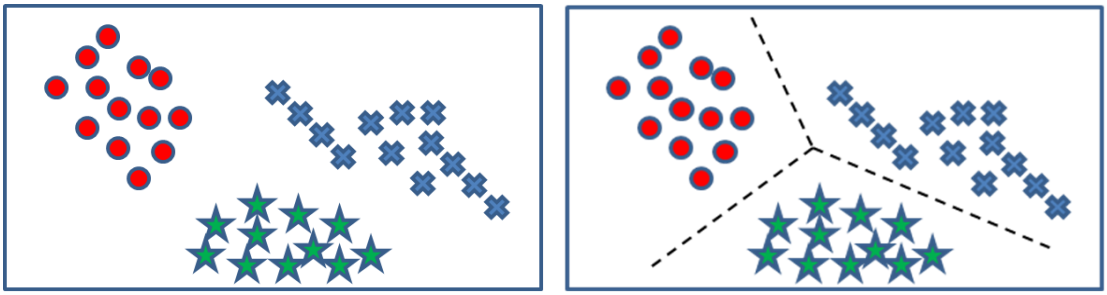
- **Unsupervised learning:** The training set given for unsupervised leaning is the unlabeled dataset also defined in Section 2.1. Unsupervised learning aims at **clustering** [12], **probability density estimation**, **finding association among features**, and **dimensionality reduction** [13]. In general, an unsupervised algorithm may simultaneously learn more than one properties listed above, and the results from unsupervised learning could be further used for supervised learning.

- **Reinforcement learning:** Reinforcement learning is used to solve problems of decision making (usually a sequence of decisions), such as robot perception and movement, automatic chess player, and automatic vehicle driving. This learning category won't be discussed further in this thesis, and readers could refer to [14] for more understanding.

In addition to these three types, a forth type of machine learning category, **semi-supervised learning**, has attracted increasing attention recently. It is defined between supervised and unsupervised learning, contains both labeled and unlabeled data, and jointly learns knowledge from them. Figs. 4-6 provide clear comparisons among these three types of learning based on nearly the same training set, and the dotted lines show the learned knowledge.

3.3 The Structure of Learning

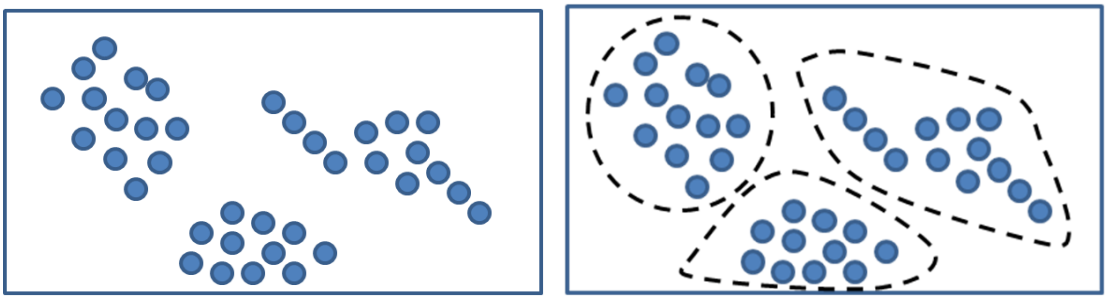
In this subsection, the structure of machine learning is presented. In order to avoid confusion about the variety of unsupervised learning structures, only the supervised learning structure is shown. While in later sections, several unsupervised learning techniques will still be mentioned and introduced, and important references for further reading are listed. An overall illustration of the supervised learning structure is given in Fig. 7. Above the horizontal dotted line, an unknown target function f (or target distribution) that maps each feature sample in the universal dataset to its corresponding label is assumed to exist. And below the dotted line, a training set coming from the unknown target function is used to learn or approximate the target function. Because there is no idea about the target function or distribution f (looks like a linear boundary or a circular boundary?), a **hypothesis set** $\boldsymbol{H}$ is necessary to be defined, which contains several **hypotheses** $\boldsymbol{h}$ (a mapping function or distribution).



(a) (b)

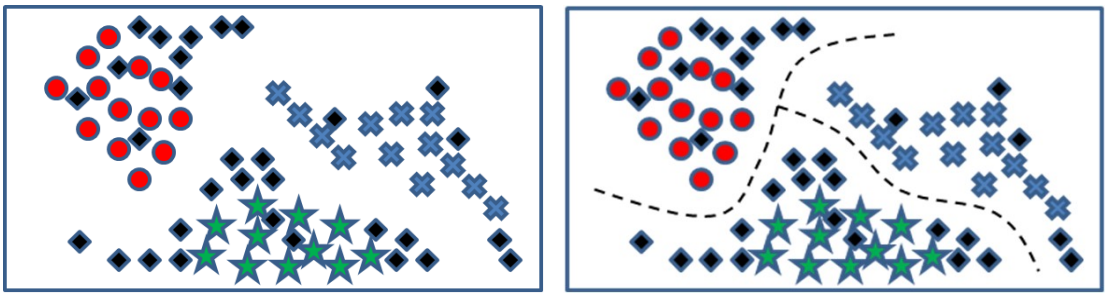
Fig. 4 Supervised learning: (a) presents a three-class labeled dataset, where the color shows the corresponding label of each sample. After supervised learning, the class-separating boundary could be found as the dotted lines in (b).

Insides the hypothesis set H, the goal of supervised learning is to find the best h, called the **final hypothesis** $\boldsymbol{g}$, in some sense approximating the target function f. In order to do so, we need further define the **learning algorithm** $\boldsymbol{A}$, which includes the **objective function** (the function to be optimized for searching g) and the **optimization methods**. The hypothesis set and the objective function jointly model the property to learn of the no free lunch rules, as mentioned in Section 2.3. **Finally, the final hypothesis $\boldsymbol{g}$ is expected to approximate $\boldsymbol{f}$ in some way and used for future prediction.** Fig. 8 provides an explanation on how hypothesis set works with the learning algorithm.



(a) (b)

Fig. 5 Unsupervise learning (clustering): (a) shows the same feature set as above while missing the label set. After performing the clustering lagorithm, three underlined groups are discovered from the data in (b). Also, users can perform other konds of unsupervides learning algorithm to learn different kinds of knowledge (ex. Probability distributuion) from the unlabeled dataset.



(a) (b)

Fig. 6 Semi-supervised learning. (a) presents a labeled dataset (with red, green, and blue) together with a unlabeled dataset (marked with black). The distribution of the unlabeled dataset could guide the position of separating boundary. After learning, a different boundary is depicted against the one in Fig. 4.

There are three general requirements for the learning algorithm. First, the algorithm should find a stable final hypothesis g for the specific d and N of the training set (ex. convergence). Second, it has to search out the correct and optimal g defined through the objective function. The last but not the least, the algorithm is expected to be efficient.

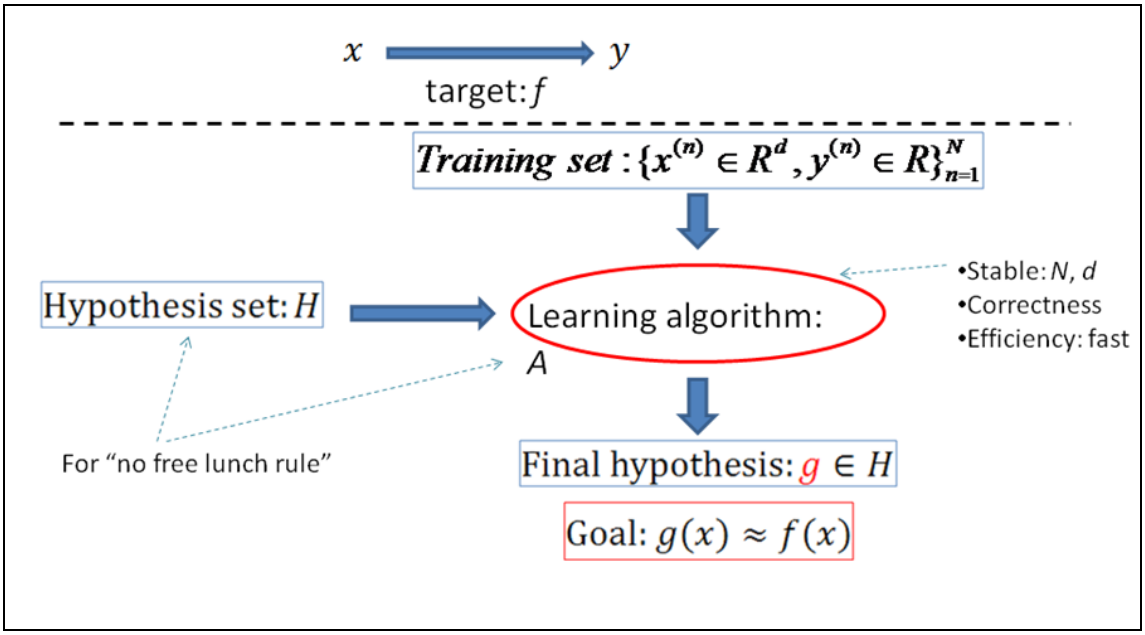


Fig. 7 The overall illustration of supervised learning structure: The part above the dotted line is assumed but inaccessible, and the part below the line is trying to approximate the unknown target function (f is the true target function and g is the learned function).

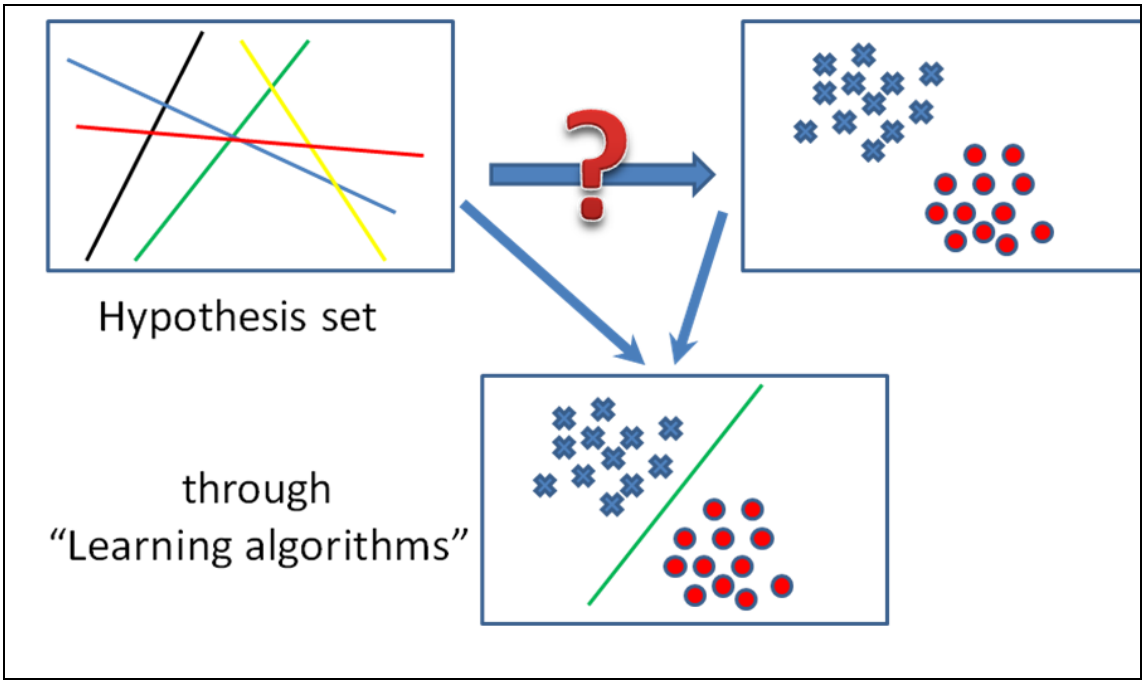


Fig. 8 An illustration on hypothesis set and learning algorithm. Take linear classifiers as an example, there are five hypothesis classifiers shown in the up-left rectangle, and in the up-right one, a two-class training set in shown. Through the learning algorithm, the green line is chosen as the most suitable classifier.

3.4 What are We Seeking?

From the previous subsection, under a fixed hypothesis set H and a learning algorithm A, learning from labeled dataset is trying to find g that best approximates the unknown f "in some sense". And in this subsection, the phrase in the quotes is explained for both supervised and unsupervised learning:

- **Supervised learning:** In supervised learning, especially for the classification case, the desired goal (also used as the performance evaluation) of learning is to find g that has the lowest error rate for classifying data generated from f. The definition of classification error rate measured on a hypothesis h is shown as below:

$$E(h) = \frac{1}{N}\sum_{n=1}^{N} \quad \boldsymbol{x} \quad \quad (2)$$

, where stands for the indicator function. When the error rate (2) is defined on the training set, it is named the "**in-sample error** $E_{in}(h)$", while the error rate calculated on **the universal set** or more practically the (unknown or reserved) **test set** is named the "**out-of-sample error** $E_{out}(h)$". Based on these definitions, the desired final hypothesis g is the one that achieves the lowest out-of-sample error over the whole hypothesis set:

$$g = \arg\min_{h} E_{out}(h). \quad (3)$$

While in the learning phase, we can only observe the training set, measure $E_{in}(h)$, and search g based on the objective function. From the contradiction above, a question the readers may ask, "What is the connection among the objective function, $E_{in}(g)$, and $E_{out}(g)$, and what should we optimize in the learning phase?"

As mentioned in (1), the connection between the learned knowledge from the training set and its availability on the test set can be formulated as a probability equation. That equation is indeed available when the hypothesis set contains only one hypothesis. For more practical hypothesis sets which may contain infinite many hypotheses, an extended version of (1) is introduced as:

$$E_{out}(g) \le E_{in}(g) + O(\sqrt{\frac{d_{VC}}{N}\log N}), \text{ with probability} \ge 1-\delta. \quad (4)$$

This inequality is called the **VC bound** (Vapnik–Chervonenkis bound), where d_{VC} is the VC dimension used as a measure of **model (hypothesis set and objective function)** complexity, and N is the training set size. The VC bound listed here is a simplified version, but provides a valuable relationship between $E_{out}(g)$ and $E_{in}(g)$: A hypothesis g that can minimize $E_{in}(h)$ may induce a low $E_{out}(g)$. The complete definition of the VC dimension is beyond the scope of this tutorial.

Based on the VC bound, a supervised learning strategy called empirical risk minimization (ERM) is proposed to achieve low $E_{out}(g)$ by minimizing $E_{in}(g)$:

$$g = \arg\min_{h} E_{in}(h). \tag{5}$$

ERM is probably the most widely-used strategy is supervised learning, and $E_{in}(h)$ is the first objective function presented in this tutorial. In fact, an objective function can be separated into a loss function and a penalty function. The loss function measures the classification error defined on the training set, while the penalty function gives each hypothesis a priority. Before Section 4.4, the penalty term is set as a constant and can be ignored. There are other kinds of supervised learning strategies seeking the minimum $E_{out}(g)$ based on theorems apart from the VC bound and will be mentioned in Section 3.6.

For regression problem, the widely-used strategy is to minimize the root mean square (RMS) between the predicted label and the ground truth label (the label provided by the dataset):

$$E(h) = \frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)} - h(\boldsymbol{x}^{(n)})\right|^2. \tag{6}$$

- **Unsupervised learning:** Apart from supervised learning, the strategies of unsupervised learning are very diverse. Some unsupervised learning algorithms exploit probabilistic distribution model and find the best distribution parameters through maximum likelihood estimation (MLE), maximum a posterior (MAP), or a more complicated Bayes methods. On the other hand, algorithms without probability models may learn knowledge based on statistical measurement, quantization error, variance preserving, or entropy gap, etc.

3.5 The Optimization Criterion for Supervised Learning

As mentioned in Section 2.4, **modeling** and **optimization** are the two main factors of machine learning. The modeling contains the choice of hypothesis set H and

the objective function, and optimization is performed to find the final hypothesis *g* in *H*, which reaches the minimum or maximum of the objective function (if necessary, within the user-defined number of iteration). Given a training set, there are indeed many kinds of models to choose. In order to avoid confusion, in this section we assume the hypothesis set is fixed, and what we want to do is searching *g* based on the selected objective function. In Section 4, we will introduce the methods to choose a model for the training set and problem at hand, and in Section 5, the types of hypothesis sets and their corresponding objective functions are discussed in more detailed.

We first focus on the hypothesis set of **linear classifiers** and show how the optimization methods interact with the choices of objective functions. The general form of the linear classifier for two-class classification problems ($y^{(n)} \in [-1,1]$) is formulated as below:

$$h(\boldsymbol{x}^{(n)}) = sign(\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)} + c), \tag{7}$$

where *c* is some constant, *w* is a (*d*+1)-dimensional vector $\boldsymbol{w} = [w_0, w_1, ..., w_d]^T$, and $\underline{\boldsymbol{x}} = [1, x_1, x_2, ..., x_d]^T$ is the extended version of $\boldsymbol{x}$ also with (*d*+1) dimensions. The vector $\boldsymbol{w}$ stands for the classifier parameters, and the additional 1 in $\underline{\boldsymbol{x}}$ is used to compute the offset of the classification line. Based on the goal of **ERM** introduced in Section 3.4, the objective function is the in-sample error term (or say the loss term, calculating how many training samples are wrongly predicted) and the optimization method is used to find a linear classifier to minimize the objective function. Fig. 9 shows a **linearly-separable** training set as well as the corresponding final hypothesis *g*. As you can see, there are many hypotheses that could achieve zero error.

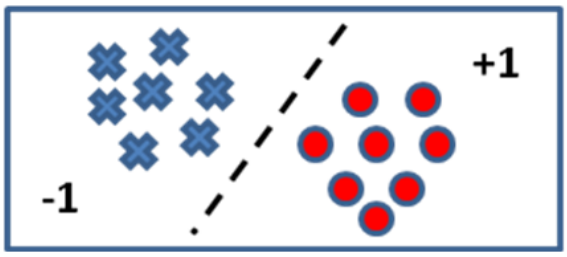


Fig. 9 An example of two-class classification problem using linear classfiers, where the training set is linearly-separable.

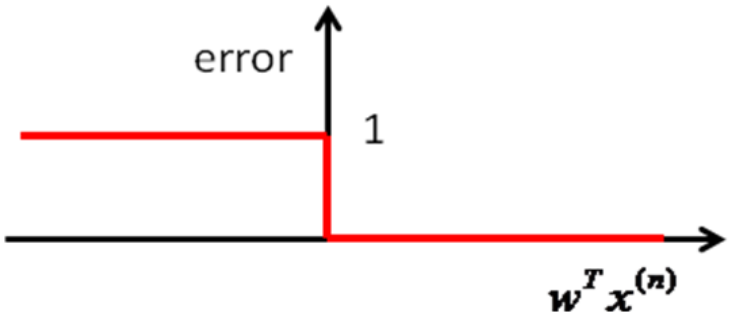


Fig. 10 The objective (in-sample error rate) considering only a sample with $y^{(n)} = 1$ based on linear classfiers. The *x*-axis denotes the inner product of the extended feature vector and the parameter vector of the linear classifier. As shown, the objective function is non-continuous around $\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)} = 0$.

If we simplify (2) and just look at one sample without normalization, the error term will become:

$$loss^{(n)}(g) = \| \qquad \boldsymbol{x} \qquad \| \tag{8}$$

Fig. 10 shows this one-sample objective function for a sample with $y^{(n)} = 1$. As can be seen, the fuction is non-continuous around $w^T \underline{\boldsymbol{x}}^{(n)} = 0$ and flat in the other ranges, so the use of differentiation-based (iterative) optimization methods is nearly out of work. For example, if the attained $\boldsymbol{w}$ brings $\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)} < 0$ for $y^{(n)} = 1$ at the current iteration, with zero gradients according to $\boldsymbol{w}$, the optimization algorithm has no idea to adjust the current $\boldsymbol{w}$ towards lower error rate. Fig. 11 illustrates this problem for a linear-separable training set.

Differentiation-based optimization methods are probably the most widely-used optimization techniques in machine learning, especially for objective functions that can be directly written as a function form of the traning samples and the classifier or regressor parameters $\boldsymbol{w}$ (not always in the vector form). The popular gradient descent, stochastic gradeint descent, Newton's method, coordinate descent, and convex optimization are of this optimization category. The differentiation-based methods are usually performed in the iterative manner, which may suffer from the local optimal problem. Besides, some of them cannot even reach the exact local optimal due to convergence concern, where slow updating and small vibration usually occur around the exact optimal parameters. Despite these drawbacks, the optimization category is

popular because of its intuitive geometrical meaning and usually easy to start with by simple caluculus such as the Taylor's expansion.

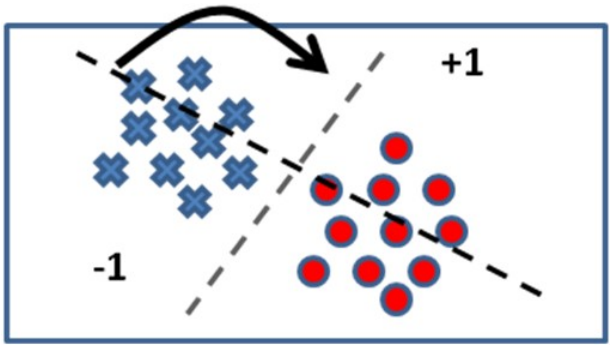


Fig. 11 Assume at the current iteration, the attained *w* and one of the desired *w* are shown as the black and the gray dotted lines, **the optimization algorithm may have no idea on how to adjust *w* towards its desired quantities**.

The basic concerns to exploit this optimization category are the "differentiability" of the objective function and the "continuity" of the parameter space. The objective function may have some non-continuous or undifferentiable points, while it should at least be in a piecewise differentiable form. In addition to differentiability, we also expect that the fuction has non-zero gradients along the path of optimization, and the zero gradients only happen at the desired optimal position. As shown in **錯誤! 找不到參照來源。**, the in-sample error rate of linear classifiers is neither continuous nor with non-zero gradients along the optimization path. This non-continuous objective function may still be solved by some other optimization techniques such as the perceptron learning algorithm, the neural evolution, and the genetic algorithm, etc., while they are either much more complicated, require more computational time, or are only available in certain convergence-guarantee conditions. The objective function should not be confused with the classifier or regressor functions. The second term is the function for predicting the label of the feature vector, and the first term is the function used to find the optimal parameters of classifiers or regressors.

To make differentiation-based optimization methods available for ERM in the linear classifier case, we need to modify the in-sample error term into some other approximation functions that is (piecewise) differentiable and continuous. There are many choices of approximation functions (denoted as $E_{app}(h)$), and the only constraint on them is made as below:

$$E_{app}(h) \geq E_{in}(h) \tag{9}$$

, which means the in-sample error is always upper-bounded by the approximation function. Based on this modification, the learning goal and procedure for ERM is reformulated. The original learning goal aims at finding g^* which approaches the target function f through minimizing $E_{out}(h)$. Because of the inaccessibility to the test and universal set, the learning phase turns to optimize $E_{in}(h)$ with g by ERM, and expects g to approach g^* ($E_{out}(g) \approx E_{out}(g^*)$) through the VC bound introduced in (4). Furthermore, due to the difficulty of optimizing $E_{in}(h)$, an approximation function $E_{app}(h)$ is defined to take place of $E_{in}(h)$. Trough searching g which optimizes $E_{app}(h)$ with the constraints defined in (9), we expect the final hypothesis g could achieve low $E_{in}(g)$ as well as low $E_{out}(g)$ ($E_{in}(g) \approx E_{in}(g)$ and $E_{out}(g) \approx E_{out}(g^*)$). Table 1 summarizes this important concept and relationship, and 錯誤! 找不到參照來源。 shows several approximation functions as well as the algorithm names for linear classifiers against the in-sample error term (0/1 loss). Similar to 錯誤! 找不到參照來源。 and 錯誤! 找不到參照來源。, this figure denotes the objective function for a single sample with $y^{(n)} = 1$, and for sample with $y^{(n)} = -1$, the mirror operation is taken for these fuctions around the origin. The objective function for the whole training set ($E_{app}(h)$) is just the normalized summation of these one-sample functions.

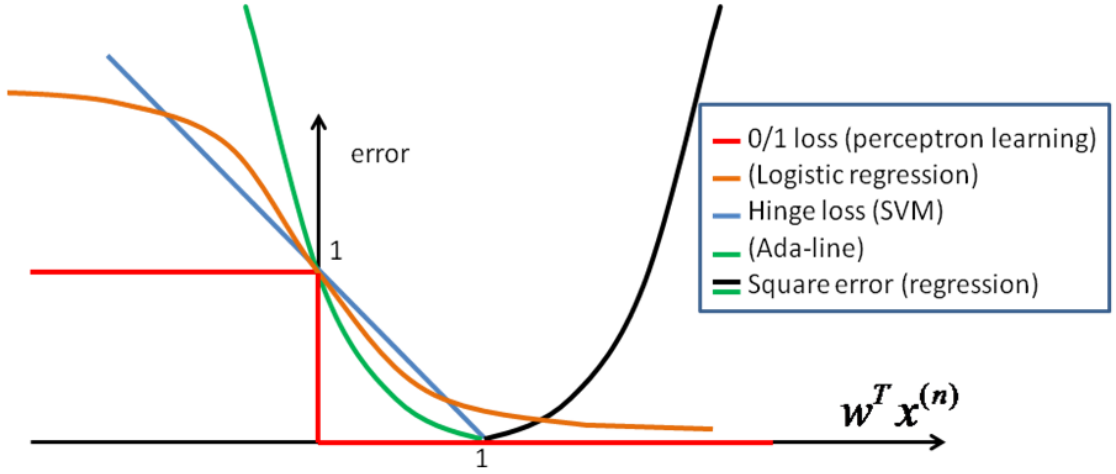


Fig. 12 Different objective functions for linear classifiers defined on a sample with $y^{(n)} = 1$. The terms in parentheses are the corresponding algorithm names.

Table 1 The supervised learning concept from ERM to the objective functions.

Original goal:	Find a final hypothesis g^*, which approaches the target function f through achieving the minimum $E_{out}(h)$.
Given:	A labeled training set $\mathbb{D}$.
Learning structure:	A hypothesis set H, an objective function $E_{app}(h)$, and the corresponding optimization method
Term definition:	$g^* = \arg\min_h E_{out}(h)$ $g = \arg\min_h E_{in}(h)$ $g = \arg\min_h E_{app}(h)$
Modified goal:	Find g which optimizes $E_{app}(h)$, and use it for further application.
Relationship:	**(a) Small $E_{app}(g)$ may brings small $E_{in}(g)$, through 錯誤! 找不到參照來源。.** **(b) Small $E_{in}(g)$ has probability relationship to achieve small $E_{out}(g)$, through the VC bound introduced in (4).** **(c) We expect $g \approx g^* \approx f$ through $E_{out}(g) \approx E_{out}(g^*)$ based on the above two relationships.**

Table 2 Several hypothesis set types as well as the corresponding objective functions and optimization techniques.

Hypothesis type	Objective function	Optimization technique
Linear classifier	0/1 loss hinge loss one- and two-side square error	perceptron learning (PLA) convex optimization (stochastic) gradient descent
Decision tree	Gini index Entropy	Divide and conquer Brute-force search
Generative classifier	Maximum a posterior (MAP) Maximum likelihood (MLE)	(stochastic) gradient descent Expectation maximization (EM)
Linear regressor	Square error	Closed form (pseudo inverse)

	Square error with regularization	(stochastic) gradient descent

In addition to the linear classifiers, there are still many kinds of hypothesis sets as well as different objective functions and optimization techniques (as listed in

Table 2) for supervised learning on a given training set. To be noticed, both the hypothesis set types and the corresponding objective functions affect the VC dimension introduced in (4) for model complexity measurement. And even based on the same hypothesis set, different objective functions may result in different final hypothesis g.

3.6 The Strategies of Supervised Learning

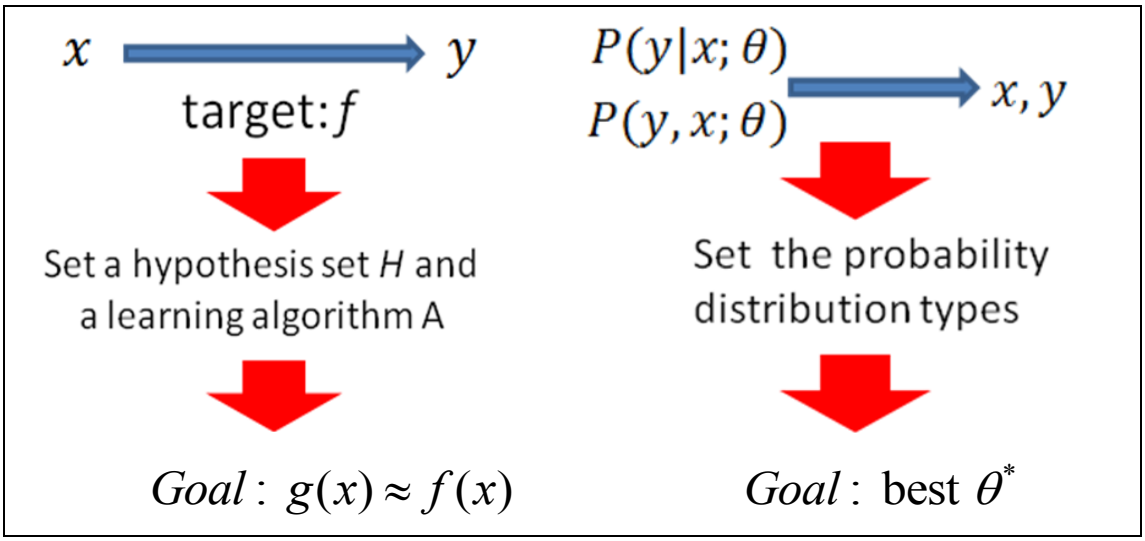
In this subsection, we discuss the other supervised learning strategies, their learning structures and assumptions, and the comparison and relationship with the ERM strategy. Only the classification problem is discussed, while these strategies can be extended into the regression problems by considering continuous labels.

There are generally two strategies of classifiers, the **one-shot (discriminant)**, and the **two-stage (probabilistic)** strategies. The one-shot (discriminant) strategy aims at finding a function that directly maps the feature vector to the label, which is usually optimized through the idea of ERM and its approximated versions. On the other hand, the two-stage strategy exploits probabilistic methods and can be further divided into two groups, the **discriminative** and **generative** models. The discriminative model tries to model the classifier as a conditional probability distribution (CPD) given the feature vector, while the generative model utilizes an extended version, modeling the classifier as several CPDs given each label as well as a prior probability distribution of labels.

The basic idea behind the two-stage strategy is the assumption that the training set is coming from a probability distribution. There are many kinds of parametric probability models as well as semi- or non-parametric probability models, and in practice, the users are asked to select a model based on their knowledge and the trade-off between complexity and learning efficiency. During the learning phase, the two-stage strategy **searches the parameter set θ of the selected probability distribution model which best describes the observed training set based on MAP**, MLE, or even the Bayes methods, while the one-shot strategy tries to find a final

hypothesis g from the hypothesis set H. Table 3 lists both the classifier modeling and the optimization criteria, and Fig. 13 illustrates the different learning structures of these two strategies.

Compared to the one-shot strategy which only outputs the predicted label, the two-stage strategy comes up with a soft decision, the probability of each label given a feature vector. The generative model further discovers the joint distribution between feature vectors and labels, and provides a unified framework for supervised, semi-supervised, and unsupervised learning. Although the two-stage strategy seems to extract more information from the training set, the strong assumption, the training samples come from a user-defined probability distribution model, may misleads the learning process if the assumption is wrong, and results in a poor model. Besides, the optimization of a flexible probability distribution model is usually highly complicated and requires much more computational time and resources. In Table 4, a general comparison and illustration of the one-shot and two-stage strategies is presented. In this tutorial, we focus more on the one-shot strategy, and readers who are interested in the two-stage strategy can referred to several excellent published books [9][15]. To be noticed, although the two strategies are different through what they are seeking during the learning phase, in the testing phase, both of them are measured by the classification error for performance evaluation.



(a) (b)

Fig. 13 The learning structures of the one-shot and two-stage strategies: (a) The one-shot strategy, and (b) the two-shot strategy, where θ is the parameter set of the selected probability distribution model.

Table 3 The classifier modeling and optimization criteria for these two strategies.

Classifier type	Classifier modeling	Optimization criterion
One-shot (discriminant)	$y^* = f(\boldsymbol{x})$	$g = \arg\min_h \frac{1}{N}\sum_{n=1}^{N}$ $\boldsymbol{x}$
Two-stage (discriminative)	$y^* = \arg\max_y P(y \mid \boldsymbol{x})$	$\theta^* = \arg\max_\theta P(Y \mid X;\theta)$
Two-stage (generative)	$y^* = \arg\max_y P(y \mid \boldsymbol{x})$ $= \arg\max_y \frac{P(\boldsymbol{x} \mid y)P(y)}{P(\boldsymbol{x})}$	$\theta^* = \arg\max_\theta P(X, Y;\theta)$

Table 4 Comparisons of the one-shot and two-stage strategies from several aspects.

Category	One-shot	Two-stage
Model	Discriminant	Discriminative, Generative
Advantage	• Fewer assumptions • Model: direct towards the classification goal • Optimization: direct towards low error rate	• More flexible • More discovery power • Provide uncertainty • Domain knowledge is easily included
Disadvantage	No probabilistic information	• More assumption • Computational complexity
Usage	Usually supervised learning	Supervised and unsupervised
Symbolic classifiers	Adaboost support vector machines (SVM), multilayer perceptrons (MLP)	Gaussian discriminant analysis, Hidden Markov model (HMM), Naïve Bayes

4. Principles and Effects of Machine Learning

In previous two sections, the definition of machine learning as well as the optimization and categorization has been mentioned, and in this section, more

practical issues will be introduced and discussed, **especially for classification problems**. At the beginning, the VC bound is revisited and explained in more detail, and three effects based on it are introduced. Then, How to select and modify a model (hypothesis set + objective function) for the training set and problem at hand is discussed. Three principles which we should keep in mind when considering a machine learning problem are coming later, and finally we take a fist glance on some practical issues.

4.1 The VC Bound and Generalization Error

The basic while probably the most important theory of ERM learning strategy is the VC (Vapnik–Chervonenkis) bound, where it builds a bridge between what we learn in the training set and how it performs in the test set. The VC bound could also be extended into other supervised strategies as well as unsupervised learning tasks by changing the two "error rates" of the training set and the test set into other quantities or properties we are interested in, such as the probability distribution, etc. The simplified VC bound is revisited as below (in the big *O* notation):

$$E_{out}(g) \le E_{in}(g) + O(\sqrt{\frac{d_{VC}}{N}\log N}), \text{ with probability} \ge 1-\delta \tag{10}$$

, where d_{VC} is the VC dimension used as a measure of model (hypothesis set *H* and objective function) complexity, and *N* is the training set size, and the definitions of $E_{in}(g)$ and $E_{out}(g)$ were illustrated in (2). Any combination of hypothesis sets and objective functions has its specific d_{VC}, and the VC bound serves as an estimate on how the learned *g* from the selected model will perform in the test or universal set. The term, $O(\sqrt{\frac{d_{VC}}{N}\log N})$, could be explained as the upper bound of the generalization gap between $E_{in}(g)$ and $E_{out}(g)$. **The details of the VC dimension d_{vc}** are beyond the scope of this tutorial, while we can think of it as **the power of models**. For example, nonlinear classifiers have higher VC dimension than linear classifiers because they could generate more flexible classification boundaries and achieve lower in-sample error rates. The VC bound is also called the "generalization error or performance", which emphasizes more on how the learned properties perform on the test set. Sometimes, these two terms are just referred to the generalization gap, while in this thesis, we prefer the first version.

Given a training set (*N* and *d* fixed), the **general relations** between the VC dimension and terms in the VC bound are formulated as follows:

$$d_{VC} \uparrow \Rightarrow E_{in}(g) \downarrow \tag{11}$$

$$d_{VC} \uparrow \Rightarrow O(\sqrt{\frac{d_{VC}}{N}\log N}) \uparrow . \tag{12}$$

From the VC bound, we know that the desired quantity to be minimized, $E_{out}(g)$, is dependent on both the terms in (11), (12), which means even $E_{in}(g)$ is small, an additional term $O(\sqrt{\frac{d_{VC}}{N}\log N})$ should also be kept small to ensure a small bound of $E_{out}(g)$. Unfortunately, by changing the elements (hypothesis sets, objective functions, and the optimization methods) in learning structures as well as changing d_{VC} to reduce one term in the VC bound, the other term will increase, and we don't know how $E_{in}(g)$ will vary.

In addition to d_{VC}, which strongly depends on the selected model, $E_{in}(g)$ and $O(\sqrt{\frac{d_{VC}}{N}\log N})$ are also affected by the training set characteristics, such as *N* and *d*:

$$N \uparrow \Rightarrow O(\sqrt{\frac{d_{VC}}{N}\log N}) \downarrow \tag{13}$$

$$d \uparrow \Rightarrow d_{VC} \uparrow \Rightarrow O(\sqrt{\frac{d_{VC}}{N}\log N}) \uparrow \tag{14}$$

$$d \uparrow \Rightarrow d_{VC} \uparrow \Rightarrow E_{in}(g) \downarrow \tag{15}$$

, where *d* is the dimensionality of feature vectors. The more samples the training set contains, the higher credibility the properties learned from it. Besides, **the feature dimensionality has some positive connection with the VC dimension** d_{VC}. Different feature dimensionalities will result in hypothesis sets with different dimensionalities or numbers of parameters, which indicates the change in model complexity. So when d increases, $E_{in}(g)$ decreases, while $O(\sqrt{\frac{d_{VC}}{N}\log N})$ will become larger. Later some illustrations are shown to give the readers more details.

4.2 Three Learning Effects

In Table 1, based on a specific model and a given training set, the concept of minimizing the objective function by *g* in the learning process and its relation to $E_{out}(g)$ is presented. And in this subsection, we will change the elements in the learning structure and see how it works on $E_{out}(g)$. Besides, the connection between the training set characteristics and $E_{out}(g)$ will also be discussed.

There are three general effects based on either the VC dimension d_{VC} or the training set characteristics (ex. *N* and *d*): over-fitting versus under-fitting, bias versus

variance, and the learning curve.

- **Over-fitting versus under-fitting (fixed *N*):** As shown in Fig. 14, this effect illustrates the relation between d_{VC}, model complexity, $E_{in}(g)$, and $E_{out}(g)$. Given a training set, if a too-easy model is used for learning, then both $E_{in}(g)$ and $E_{out}(g)$ will be really high, which is called "under-fitting". On the other hand, if a over-complicated model is exploited, although a really small $E_{in}(g)$ could probably be achieved, $E_{out}(g)$ will still be high due to a large $O(\sqrt{\frac{d_{VC}}{N}\log N})$, which is called "over-fitting". Based on this effect and observation, selecting a suitable model as well as a moderate d_{VC} plays an important role in machine learning. The VC dimension d_{VC} could be controlled by the type of hypothesis set, the objective function, and the feature dimensionality. Although the feature dimensionality d is given by the training set, several operations could be performed to reduce or increase it when feeding the training set into the learning structure (which will be further discussed in later sections).

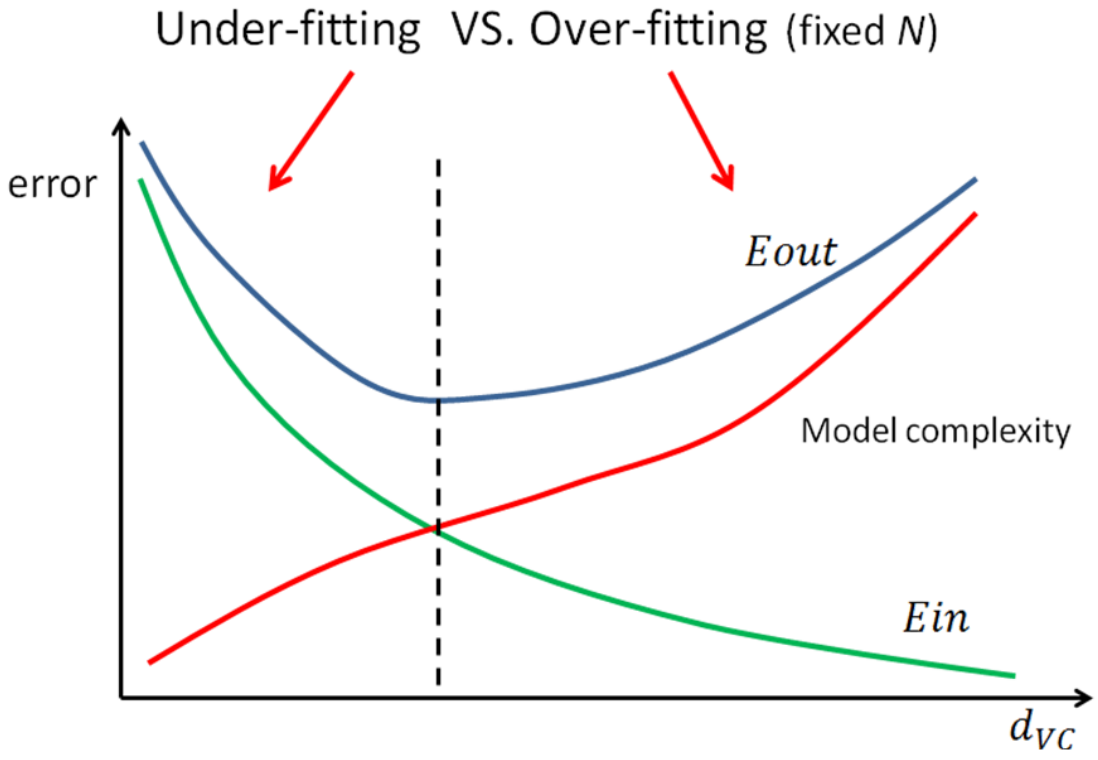


Fig. 14 The under-fitting versus over-fitting effect.

- **Bias versus variance (fixed *N*):** As shown in Fig. 15, the bias versus variance effect has a similar curve as the under-fitting versus over-fitting curve shown in

Fig. 14, while the explanation is different and it focuses more on statistics and regression analysis. Bias means the ability to fit the training set (the smaller the better), where stronger assumptions on the training set will result in a larger bias. For example, the bias of using linear classifiers is bigger than the bias of using nonlinear classifiers, because the set of nonlinear classifiers contains the set of linear classifiers and seems to be more general. On the other hand, the variance term means the variation of the final hypotheses when different training sets coming from the same universal set are given.

Now let me take a revisiting to the Hoeffding inequality and the VC bound mentioned in (1) and (10). The readers may have questions why there is a "probability" term in these two inequalities, and the reason comes from the quality of the training set. The desired goal of machine learning is to find the properties of the universal set, while the only thing we observe during learning is the training set. There exists an uncertainty that how representative the training set is for the universal set, and the probability term stands for the chance that a poor training set is observed.

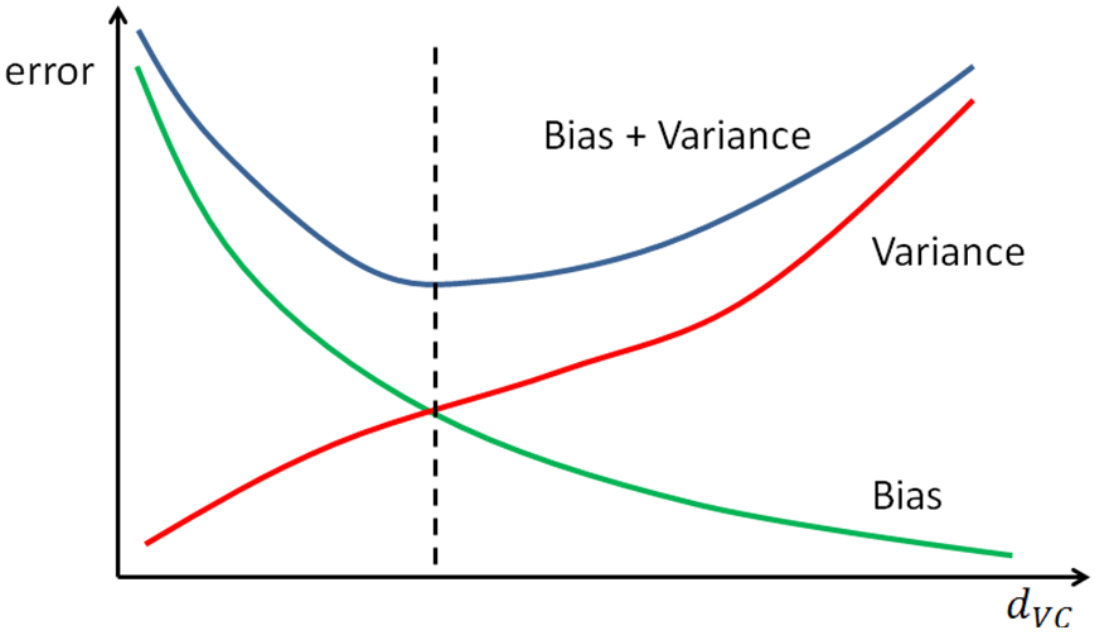


Fig. 15 The bias versus variance effect.

As the definitions of bias and variance go, a low bias model has strong abilities to fit the training set and reach low $E_{in}(g)$ as mentioned in (2) and (6). If the training set is representative, the final hypothesis g will be really closed to the target function f, while if the training set is poor, g can be really dissimilar

from f. These effects result in a large variance for a low bias model. In contrary, a high bias model has poor abilities to fit the training data, while the variance among the final hypotheses based on different training sets is small due to limited variation in the hypothesis sets. For statisticians and regression analysis, the balance between bias and variance is the key to judge the learning performance, and the relationship between d_{VC} and the bias versus variance effect is illustrated in Fig. 15. Although the shape of bias and variance looks really similar to $E_{in}(g)$ and $O(\sqrt{\frac{d_{VC}}{N}\log N})$, there is no strong yet direct relationship among them.

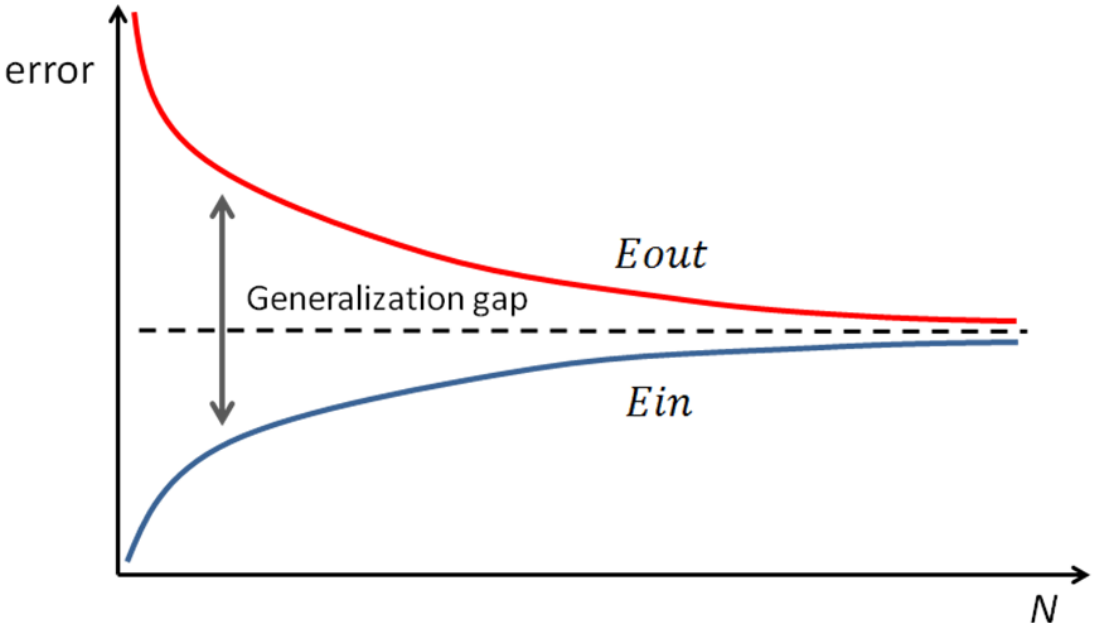


Fig. 16 The learning curve effect.

If we focus on the learning performance at a single sample $\boldsymbol{x}$ which is not necessary in the training set X, and assume that the universal set is a probability distribution where each possible feature vector is mapped to each possible label through a target probability distribution $f(y \mid \boldsymbol{x})$ (frequently used in regression analysis), then the equation combining bias and variance can be formulated as below:

$$
\begin{aligned}
& E_X\left[E_f\left[(y-g(\boldsymbol{x}))^2\right]\right] \\
&= E_X\left[E_f\left[(y-E_f(y)+E_f(y)-g(\boldsymbol{x}))^2\right]\right] \\
&= E_X\left[E_f\left[(y-E_f(y))^2\right]+E_f\left[(E_f(y)-g(\boldsymbol{x}))^2\right]\right] \\
&= E_X\left[\text{var}(y)+(E_f(y)-E_X[g(\boldsymbol{x})]+E_X[g(\boldsymbol{x})]-g(\boldsymbol{x}))^2\right] \\
&= \text{var}(y)+E_X\left[(E_f(y)-E_X[g(\boldsymbol{x})])^2\right]+E_X\left[(E_X[g(\boldsymbol{x})]-g(\boldsymbol{x}))^2\right] \\
&= \text{var}(y)+(E_f(y)-E_X[g(\boldsymbol{x})])^2+\text{var}(g(\boldsymbol{x})) \\
&= \text{var}(y)+\text{bias}(g(\boldsymbol{x}))+\text{var}(g(\boldsymbol{x}))
\end{aligned}
\tag{16}
$$

, where E_f means the expectation over the target distribution given $\boldsymbol{x}$, and E_X is the expectation over all possible training set (maybe poor or representative). The first term in the last row shows the intrinsic data variation coming from the target distribution *f*, and the two following terms stand for the bias and variance of the selected hypothesis set as well the objective function.

- **Learning curve (fixed d_{VC}):** The learning curve shown in Fig. 16 looks very different from the previous two figures, and the relationship considering in this effect is between the VC bound and the training set size *N*. As mentioned earlier in (13), when *N* increases, $O(\sqrt{\frac{d_{VC}}{N}\log N})$ decreases, and the learning curve mentioned here will show how *N* affects $E_{in}(g)$ and $E_{out}(g)$. When the data size is very small, a selected model has the chance to achieve extremely low $E_{in}(g)$, for example, if only two different 2-D features are included in the training set, then no matter what their labels are, linear classifiers could attain $E_{in}(g)=0$. While with *N* increasing, there will be more and more training samples that the selected model can't handle and results in wrong predictions. But surprisingly, the increasing speed of $E_{in}(g)$ is lower than the decreasing speed of generalization gap along *N*, which means increasing *N* generally improve the learning performance.

To summarize the three effects introduced above, we found that the selection of model is one of the most important parts in machine learning. A suitable model not only reaches an acceptable $E_{in}(g)$ but also limits the generalization gap as well as $O(\sqrt{\frac{d_{VC}}{N}\log N})$. An over-complicated model with an extremely low $E_{in}(g)$ may cause

"over-fitting" effect while a too-easy model with an extremely small $O(\sqrt{\frac{d_{VC}}{N}\log N})$ may result in "under-fitting" effect. These two cases both degrade the learning performance $E_{out}(g)$. Besides, when the model and feature dimensionality are fixed, increasing the size of the training set generally improves $E_{out}(g)$. Furthermore, when judging if a model is complicated or not for a given problem, not only d_{VC} but also N should be considered. Table 5 lists these important concepts based on VC bound.

Now we revise the learning process to a more generalized procedure. As summarized in Table 1, given a fixed model with its fixed d_{VC}, the objective function is minimized over the training set and the attained final hypothesis g is expected to induce low $E_{out}(g)$ through the VC bound relationship and used for further application. And if there are many possible models and a fixed training set at hand, **"model selection"** is a necessary and important step during learning, which aims at searching the best model with the **lowest generalization error**. To be noticed, we cannot perform model selection based on $E_{in}(g)$ or $E_{app}(g)$, because now each model has its specific d_{VC}. Unfortunately again, the only term we can measure during learning is $E_{in}(g)$, with the test set preserved and $O(\sqrt{\frac{d_{VC}}{N}\log N})$ nearly unachievable (just as the upper bound and d_{VC} is usually hard to define). In fact, $O(\sqrt{\frac{d_{VC}}{N}\log N})$ often serves as a theoretical adjustment and is used just as a guideline in model selection. Furthermore, not only the VC dimension affects the performance of learning, but **different hypothesis types and different objective functions having their specific learning properties would discover various aspects of knowledge and result in different performances for the given problem, even if they are of the same VC dimension** d_{VC}**.** According to these diversities of models, a more practical method for selecting suitable models is of high demand. In the next two subsections, several popular methods for generating model diversities and performing model selection are introduced and discussed.

Table 5 Important concepts based on VC bound

Problem	Modification
$E_{in}(g)$ is high	$d_{VC}\uparrow$
small $E_{in}(g)$, high $O(\sqrt{\frac{d_{VC}}{N}\log N})$	$d_{VC}\downarrow$ or $N\uparrow$

Practical usage of d_{VC} for a given N	• To maintain a same $O(\sqrt{\frac{d_{VC}}{N}\log N})$,when d_{VC} increases, N also increases. • $N \geq 10d_{VC}$ usually performs well

4.3 Feature Transform

Before further introduction, we expect the readers to catch the meaning, why we need model selection. As a matter of fact in classification, unless there is a feature vector mapping to more than one label in the training set, we can always find a machine learning structure to achieve $E_{in}(g) = 0$, such as the well-known decision trees. While, with a probably uncontrolled $O(\sqrt{\frac{d_{VC}}{N}\log N})$, there is no guarantee that this model could bring a low $E_{out}(g)$.

As mentioned in the previous subsection, given a training set, we are willing to find the model which could extract important knowledge, **avoids over-fitting and under-fitting**, and results in the best learning performance for the ongoing problem. Afore model selection, we need to know how achieve different models as well as different model complexities. There are generally three methods to reaches this goal:

- **Hypothesis set types and objective functions (Type I):** Different hypothesis set types (ex. *K*NN, decision trees, and linear classifiers) result in different models. Furthermore, even in the same class such as linear classifiers, different objective functions (ex. square error and hinge loss) come up with different learning performances.

- **Model parameter (Type II):** Even under the same hypothesis set type and objective function, there are still some free parameters to adjust the hypothesis set. For example, in *K*NN (*K*-nearest neighbors), different selections of *K* may result in different learning performances. The use of SVM and multi-layer perceptron also requires users to set some parameters before execution. Generally, these parameters have connections with model complexity and d_{VC}.

- **Feature transform (Type III):** The last but not the least, changing the dimensionality of feature vectors will result in different d_{VC} of the model. There are a bunch of methods to modify the feature vector dimensionality, and the general framework is formulated by **basis functions**:

$$\Phi(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), ..., \phi_{d_\Phi}(\boldsymbol{x})]^T \tag{17}$$

, where $\Phi(\boldsymbol{x})$ denotes a feature transform consist of a set of basis functions $\{\phi_j(\boldsymbol{x})\}_{j=1}^{d_\Phi}, \phi_j(\boldsymbol{x}) : \boldsymbol{x} \in R^d \mapsto$. For example, the 2nd-order (quadratic) transform performed on a 2-dimensional input can be modeled as:

$$\Phi_2(\boldsymbol{x}) = [1, x_1, x_2, {x_1}^2, x_1 x_2, {x_2}^2]^T \tag{18}$$

, where the added "1" is for offset when using linear classifiers as in (7). Table 6 and Table 7 list several useful feature transforms and their definitions, and as you can see, we can always perform feature transform before feeding feature vectors into the learning machine. In addition to these kinds of "geometry- or mathematics-driven" feature transforms, there are also "data-driven" feature transforms defining their basis functions form learning (ex. PCA and LDA) and "knowledge-driven" feature transforms based on the characteristics of problems (ex. DCT and DFT). We will mention these transforms in later sections.

Based on these three methods for achieving different models as well as different model complexities, now we can generate several models and perform model selection to choose the best model among them.

Table 6 The definition of feature transform and its usage

Definition	$\Phi : \boldsymbol{x} \in R^d \mapsto$ $\Phi(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), ..., \phi_{d_\Phi}(\boldsymbol{x})]^T$
Usage	Change the dimensionality of feature vectors to result in different d_{VC}, which will affect both $E_{in}(g)$ and $O(\sqrt{\frac{d_{VC}}{N}\log N})$.
Example	Linear classifier: $h(\Phi(\boldsymbol{x}^{(n)})) = sign(\boldsymbol{w}^T \Phi(\boldsymbol{x}^{(n)})) = sign(\sum_{i=1}^{d_\Phi} w_i \phi_i(\boldsymbol{x}))$

Table 7 Useful feature transforms on a 2-dimensional feature vector, which could be further extended in to arbitrary dimensionality

Feature transform	The transform formula

Decision stump	$\Phi_S(\boldsymbol{x}) = [1, x_j]^T$, where $1 \le j \le d$
1st-order	$\Phi_1(\boldsymbol{x}) = [1, x_1, x_2]^T$
2nd-order (quadratic)	$\Phi_2(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2]^T$
3rd-order (cubic)	$\Phi_3(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3]^T$

4.4 Model Selection

Model selection is performed to find the best model which could extract important knowledge, **avoids over-fitting and under-fitting**, and results in the best learning performance for the ongoing problem. There are generally two popular methods toward this goal: regularization and validation [8].

- **Regularization:** Regularization is performed to balance $E_{in}(g)$ and model complexity. In the previous two subsections, d_{VC} is defined over the hypothesis set and objective function for model complexity measurement. As a matter of fact, each hypothesis has its own hypothesis complexity and classification power. For example, a nonlinear hypothesis set (used in nonlinear classifiers) contains the linear hypothesis as a subset, which means we can also find a linear separating boundary (hyperplane) based on the nonlinear classifiers. From the perspective of machine learning, a nonlinear boundary (ex. curves or circles) has higher classification power, higher complexity, while higher risk of over-fitting than a line boundary. To search the hypothesis which minimizes E_{in} (or E_{app}) and the hypothesis complexity jointly, the regularization method is introduced and formulated as:

$$E_{obj}(h) = E_{app}(h) + \Omega(h) \tag{19}$$

, where the $\Omega(h)$ term is used to penalty hypothesis h with higher complexity and $E_{obj}(h)$ denotes the objective function to be minimized. As introduced in Section 3.4, the objective function is composed of a loss function as well as other

pursuits, such as the penalty function $\Omega(h)$, and the approximation functions introduced in Section 3.5 are indeed loss functions because they are defined for measuring classification error of the training set. In fact, regularization searches the best hypothesis insides a hypothesis set, not among several hypothesis sets. Several widely used penalty functions and the corresponding objective functions are list in Table 8, where λ is the model parameter (Type II defined in Section 3.3) used to balance classification error of the training set and the penalty term. The reason why "objective function could affect model complexity as well as d_{VC}" is because the penalty function introduced inside has abilities to control them.

Table 8 Several widely used penalty functions in machine learning

Penalty function name	The formula
Hard limitation (L^0)	$E_{obj}(h) = E_{app}(h) + \lambda \sum_i \ w_i \neq 0$
L^1 minimization	$E_{obj}(h) = E_{app}(h) + \lambda \sum_i \lvert w_i \rvert$
L^2 minimization	$E_{obj}(h) = E_{app}(h) + \frac{1}{2} \lambda \sum_i \lvert w_i \rvert^2$

Table 9 The general validation procedure in practice

Before learning	(1) M different models: $\{H_m, A_m\}_{m=1}^{M}$ (2) A training set: $\mathbb{D}$ $\mathbb{D}$ $\mathbb{D}$ (for base and validation) (3) $E_{obj}^{(m)}$(, $\mathbb{D}$ is the objective function of model m measured on $\mathbb{D}$ (4) E_{Val}( ) is the classification error measured on $\mathbb{D}$
Training	for $m = 1:M$ find $g_m = \arg\min_{h_m} E_{obj}^{(m)}(h_m, \mathbb{D}$ end (Find the final hypothesis for each model m)
Validation	Find the model l that $l = \arg\min_m E_{Val}(g_m)$ (Run these M final hypotheses on the validation set, and select the model with the lowest validation error.)

Retrain	find $g_l = \arg\min_{h_l} E_{obj}^{(l)}(h_l, \mathbb{D})$, and used it for prediction (Retrain the selected model on the whole training set)

- **Validation:** In contrast to regularization, validation does select model from different hypothesis sets and different objective functions. Indeed, we can view a hypothesis set with different model parameter λ as different models in validation. The validation exploits the idea of **finding the best model based on** $E_{out}(g)$, where the training set is separated into a "base set" and a "validation set" (The base set contains *N*-*K* samples and the validation set contains *K* samples). During learning, each model is only trained on the base set to find its final hypothesis *g*, and these fixed final hypotheses are further tested on the validation

Table 10 Different kinds of validation strategies

Validation type	The formula
One-shot	As mentioned in Table 9
Multi-shot	for $v = 1:V$ (a) randomly generate $\mathbb{D}$ and $\mathbb{D}$ (b) perform the training step as in Table 9 (c) record $E_{Val}(g_m, v)$, where $E_{Val}(g_m, v)$ defines the validation error of model *m* in round *v* end Find the model *l* that, $l = \arg\min_m \frac{1}{V}\sum_{v=1}^{V} E_{Val}(g_m, v)$, then retrain
Cross	Uniformly cut $\mathbb{D}$ into *V* folds ($3 \le V \le 10$ in practice, a 5-fold CV is also called a 4-1 fold CV) for $v = 1:V$ (a) the fold *v* as the validation set and the others as the base set (b) perform steps 2 and 3 in the for loop of the multi-shot validation end Find the model *l* that, $l = \arg\min_m \frac{1}{V}\sum_{v=1}^{V} E_{Val}(g_m, v)$, then retrain
Leave-one-out	for $n = 1:N$ (a) Use sample *n* as the validation set and the others *N*-1 samples as the base set (b) perform steps 2 and 3 in the for loop of the multi-shot

	validation end Find the model *l* that, $l = \arg\min_m \frac{1}{N}\sum_{n=1}^{N} E_{Val}(g_m, n)$, then retrain

set. The model with the best g (achieving the lowest classification error $E_{Val}(g)$) is selected as the winner model for the ongoing problem and expected to perform well on unseen data. Table 9 describes this procedure in more detailed. There are generally four kinds of validation strategies: **One shot validation, multi-shot validation, cross validation (CV), and leave-one-out (LOO) validation**, as listed in Table 10. Besides the LOO and the cross validation, the other two strategies have "$K = (10\% \sim 40\%) \times N$" in practice. The availability of validation is based on some theoretical proofs, which is beyond the scope of this thesis. In recent pattern recognition researches, validation is the most popular methods for performance comparison.

4.5 Three Learning Principles

From the previous two subsections, how to generate several models and select the best one among them are discussed, and in this subsection, three principles that the machine learning users should keep in mind in order to prevent poor performance are introduced:

- **Occam's razor:** The simplest model that fits the data is also the most plausible, which means if two models could achieve the same expected $E_{in}(g)$, then the simpler one is the suitable model.

- **Sampling bias:** If the sampling data is sampled in a biased way, then learning will produce a similarly biased outcome. For example, if an examination of
  "how Internet affects your life?"
  is performed **on-line**, the statistical result has risk to over-estimate the goodness of Internet because people who don't like to use Internet are likely to miss this test.

- **Data snooping:** If a data set has affected any step in the learning process, it cannot be fully trusted in assessing the outcome. During learning, the hypothesis g which best fits the training data through minimizing the objective function is

selected, and during testing, we test how this learned hypothesis could be generalized in the test data. The reason why there exists a generalization gap is because the learned hypothesis *g* is biased by and may over-fit to the training data. But if a data set both affects the learning and test phase, we cannot correctly detect the generalization gap and will over-estimate the performance of the model.

4.6 Practical Usage: The First Glance

In this subsection, we take a first glance on some issues in practical usage, especially for dataset processing. As mentioned in the before sections, we make no assumption if the training data is sampled from the universal set with or without noise, while in practical case, noise can be easily generated during the data acquisition process. When facing noisy data, some preprocessing steps such as noise reduction and outlier deletion are required to perform [16], and these steps are often designed according to the domain knowledge. Besides, the regularization process is experimented to find a suitable final hypothesis *g* in noisy data which has better generalization performance than just minimizing the loss function. The reason is that a more complicated hypothesis or model has higher ability to fit not only the training data but also the noise, so a penalty term on the hypothesis complexity could guide the learning algorithm to find the final hypothesis *g* with less possibility to over-fit the noise.

Another frequently faced problem is the missing of data [17], which means each *d*-dimensional feature vector in both training and test set may have several elements lost. This problem generally occurs in data collection process, especially when data comes from questionnaire or survey where people may ignore some questions they don't want to answer. Some machine learning techniques are capable of dealing with this problem, for example, decision tree usually generates braches based on one feature element, so elements with missing values are prevented to be used when building a decision tree. Probabilistic graphical models [15] (a framework for generative models) can also handle this problem by viewing missing elements as unobserved values and perform learning and inference through marginalization over these elements.

For machine learning techniques which cannot work with missing data, data imputation [17] is necessary as a preprocessing step. Data imputation is to fill in these missing elements with values, and there are some simple methods such as filling in values with element means over the whole training set, finding the nearest neighbor in

the training set then fill in the missing elements with the corresponding values in this neighbor, and replacing missing elements with random values according to the distribution of these elements. In the missing data survey proposed by J. Schafer et al. [17], maximum likelihood (ML) and Bayesian multiple imputation (MI) are two highly recommended techniques towards this problem.

In recent machine learning competition such as KDD Cup 2009 [18], the provided data commonly contains high noise and a large portion of missing values. For extended understanding, the readers could referred to papers of KDD Cup 2009 [19][20][21].

5. Techniques of Supervised Learning

The previous two sections see machine learning as a tool or a concept and discuss its categorization, basic ideas, effects, and some practical aspects, and from this section on, we start to introduce machine learning techniques, including supervised and unsupervised learning. Categorization will be defined for each learning task, and some outstanding or widely-used techniques are described in more details.

In this section, we focus on supervised learning. The overview and categorization are provided in Section 5.1, and later the linear models are described in more detailed in Section 5.2. In Section 5.3, a broad summary and conclusion is given.

5.1 Supervised Learning Overview

In this thesis, four categories of supervised learning based on different hypothesis set types are considered, including **linear models**, **non-parametric models**, **non- metric models** and **parametric models**. This categorization method is also used in the book chapters of [8], while the order is permuted in this section. The **aggregation method** which aims at combining several models together can further improve the learning performance and has been widely applied in machine learning contests. To be noticed, many hypothesis types can handle both classification and regression problems, so we don't separate these two tasks into two sections. The categorization strategy is described as below, and Table 11 lists the categorization as well as some important techniques of each category:

- **Linear model:** The classifier which can be formulated as:

$$h(\boldsymbol{x}^{(n)}) = sign(\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)}), \text{ or} \tag{20}$$

$$h(\Phi(\boldsymbol{x}^{(n)})) = sign(\boldsymbol{w}^T \Phi(\boldsymbol{x}^{(n)})) = sign(\sum_{i=1}^{d_\Phi} w_i \phi_i(\boldsymbol{x})) \tag{21}$$

or built on blocks of these forms is categorized as a linear classifier, where $\boldsymbol{w}$ is a (d+1)-dimensional parameter vector $\boldsymbol{w} = [w_0, w_1, ..., w_d]^T$, $\underline{\boldsymbol{x}} = [1, x_1, x_2, ..., x_d]^T$ is the extended version of $\boldsymbol{x}$ also with (d+1) dimensions, and $\Phi(\boldsymbol{x})$ stands for a feature transform consist of a set of basis functions $\{\phi_j(\boldsymbol{x})\}_{j=1}^{d_\Phi}, \phi_j(\boldsymbol{x}) : \boldsymbol{x} \in R^d \mapsto$. For regression problems, the $sign(.)$ function is replaced by other continuous functions.

- **Parametric model:** For clear description of the four categories, the parametric model is introduced before the other two methods. A model is called "parametric" as it is built on **well-defined probabilistic distribution model**, which means when the parameters of the distribution model is learned from the training set, we could discard the training set and only reserve these parameters for testing and prediction. Generally speaking, when the type of probabilistic model is set, no matter how many samples are in the training set, if the number of parameters (values the model needs to remember) doesn't change, then the model is called a parametric model.

- **Non-parametric model:** The non-parametric model is also built on the idea of probability. While compared to the parametric model, it makes no assumption on the density or distribution type of the training and universal set, but assumes that similar feature vectors have similar labels. Based on this idea, for a new coming feature sample, the model finds the similar feature samples (instances) in the training set using a suitable measure and interpolates from them to determine the final output label. In fact, there is nearly no learning process for non-parametric learning, and all the training set are preserved for prediction purpose, which means the number of values the model needs to remember depends on the number of training samples. Non-parametric model is also called instance-based, memory- based, or lazy learning method.

- **Non-metric model:** For the previous three categories, each element of feature vector is assumed to contain comparable information, for example, 10 is closer to 8 than to 2. While suppose a supervised learning problem involves nominal data, where features are discrete and without any natural notion of **similarity metric** or even ordering, the previous three methods might be out of function. For example, if a feature contains three kinds of possibilities, red, blue, and green, then there is merely no clear notion of similarity among them. For this nominal case, the non-metric model is a suitable choice for learning, which can depend or not depends on the feature metric to build the model.

Readers might also have questions on what the relationship between the one-shot / two-stage strategies mentioned in Section 3.6 and the four categories defined in this section is. In Sec. 3.6, we discussed about what information a classifier (regressor) can provide as well as the optimization criteria during learning, while in this section, these four categories are defined based on their basic ideas on the hypothesis set types. Indeed, a linear classifier which is usually categorized as a one-shot method can also be modified into a probabilistic version based on some probabilistic model assumption, which means each category in this section may contains both one-shot and two-shot classifiers (regressors). As a consequence, the one-shot and two-shot strategies are not explicitly mentioned in this section.

Table 11 Overview and categorization of supervised learning techniques

Category	Important methods
Linear model	• Perceptron, Multi-layer perceptron(MLP) • Support vector machine (SVM) (free on-line, LIBSVM) • Support vector regression (SVR) (LIBSVM) • Linear regressor, Rigid regression • Logistic regression (LIBLINEAR)
Non-parametric model	• *K*-nearest neighbors • Kernel density estimation • Kernel regression ,Local regression
Non-metric model	• Classification and regression tree (CART), decision tree
Parametric model	• Naïve Bayes • Gaussian discriminant analysis (GDA) (free on-line) • Hidden Markov models (HMM) • Probabilistic graphical models

Mixed method	• Bagging (bootstrap + aggregation) • Adaboost • Random forest (code is available in Matlab2010)

5.2 Linear Model (Numerical Functions)

As described in the previous subsection, a linear model could be characterized by the parametric vector $\boldsymbol{w}$. And from Section 3.4 and 3.5, we have known that the original objective function for ERM learning strategy is to minimize the in-sample error of classification. Because it is a non-continuous function, several approximated loss functions are proposed to simplify the optimization procedure. In this subsection, we describe these approximated loss functions in more detailed and provide either pseudocode or useful toolbox of each method, including perceptron learning algorithm (PLA), Adaline, support vector machine (SVM). On the other hand, for regression problem, the most widely-used idea is the mean square error (MSE), and three popular regressors, linear regression, rigid regression, and support vector regression (SVR) are introduced. For convenience, the notation in **錯誤! 找不到參照來源。** is preferred. Besides, in the classification case, only the two-class problem ($y^{(n)} \in [-1,1]$) is discussed, and its extension to multi-class will be introduced at the end of this subsection.

Table 12 The **perceptron** learning

Presetting:

- Training set: $X = \{\boldsymbol{x}^{(n)} \in R^d\}_{n=1}^{N}, Y = \{y^{(n)} \in R\}_{n=1}^{N}, y^{(n)} \in [-1,1]$
- The loss function of each training sample is

  $y^{(n)} \neq sign(\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)})$ -> 1 $y^{(n)} = sign(\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)})$ -> 0
- Preset $\boldsymbol{w}(1)$, usually assume (d+1)-dimensional zero vector.
- $h_{\boldsymbol{w}(t+1)}$ is the hypothesis with parameter vector $\boldsymbol{w}(t+1)$

Learning:

for $t = 1:\infty$

randomly pick a $\{\boldsymbol{x}^{(n)}, y^{(n)}\}$ where $h_{\boldsymbol{w}(t)}(\boldsymbol{x}^{(n)}) = sign(\boldsymbol{w}(t)^T \underline{\boldsymbol{x}}^{(n)}) \neq y^{(n)}$

$\boldsymbol{w}(t+1) \leftarrow \boldsymbol{w}(t) + y^{(n)} \underline{\boldsymbol{x}}^{(n)}$ Key process

if $E_{in}(h_{\boldsymbol{w}(t+1)}) = 0$

$g \leftarrow h_{\boldsymbol{w}(t+1)}$, break

end

end

5.2.1 Perceptron Learning Algorithm (PLA) - Classification

Given a two-class training set, the perceptron learning algorithm (PLA) directly seeks $\boldsymbol{w}$ that results in zero in-sample error without any approximation function. This algorithm is iteratively updating the parameter vector until zero in-sample error is achieved, and the pseudocode is provided in Table 12.

If the training set is linear separable, this algorithm is proved to reach a $\boldsymbol{w}$ with zero in-sample error in limited iterations, while for non-linear-separable cases, PLA won't converge and the updated hypothesis has no guarantee to reach lower in-sample error than the previous hypotheses. To solve this problem, a modified PLA algorithm called the pocket algorithm is proposed and summarized in Table 13.

Table 13 The **pocket** algorithm

Presetting:

- The presetting of PLA in Table 12 is included
- The maximum number of iterations T
- A buffer $\boldsymbol{w}^* = \boldsymbol{w}(1)$

Learning:

for $t = 1 : T$

randomly pick a $\{\boldsymbol{x}^{(n)}, y^{(n)}\}$ where $h_{\boldsymbol{w}(t)}(\boldsymbol{x}^{(n)}) = sign(\boldsymbol{w}(t)^T \underline{\boldsymbol{x}}^{(n)}) \neq y^{(n)}$

$\boldsymbol{w}(t+1) \leftarrow \boldsymbol{w}(t) + y^{(n)} \underline{\boldsymbol{x}}^{(n)}$

if $E_{in}(h_{\boldsymbol{w}(t+1)}) < E_{in}(h_{\boldsymbol{w}^*})$

$\boldsymbol{w}^* = \boldsymbol{w}(t+1)$

end

Extra process

if $E_{in}(h_{\boldsymbol{w}^*}) = 0$ or $t = T$

$g \leftarrow h_{\boldsymbol{w}^*}$, break

end

end

The pocket algorithm can find the best hypothesis which reaches the minimum in-sample error in T iterations, while for a non-linear-separable training set, there is no guarantee that within how large T a plausibly low in-sample error could be achieved. Besides, because linear classifiers can only generate linear classification boundaries, the pocket algorithm still cannot solve non-linear-separable training set very well, especially when the class boundary of the training set is far away from a line. To solve this problem, we show in the next subsection that performing feature transform can make the linear classifier available for non-linear boundary cases.

5.2.2 From Linear to Nonlinear

As mentioned in Section 4.3, the feature transform shown as below can change the input dimensionality d into d_Φ :

$$\Phi(\boldsymbol{x}) = [\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), ..., \phi_{d_\Phi}(\boldsymbol{x})]^T. \tag{22}$$

Based on this transformed feature vectors, a linear classifier with d_Φ-dimensional parameter vector $\boldsymbol{w}$ can be trained using the same learning algorithm. The feature transform provides the ability that a non-linear-separable training set in the original d-dimensional feature space may become a linear-separable training set in the transformed d_Φ-dimensional feature space. Then, a linear classifier trained in the transformed feature space could find the perfect separating boundary. In Fig. 17, a training set that can be separated by a circle $x_1^2 + x_2^2 = 1$ is presented. If the linear classifier is performed without feature transform, the achieved final hypothesis $sign(x_1 - x_2 + 0)$ (gray dotted line) seems far away from the separating boundary. Now

with the 2nd-order feature transform introduced in Table 7:

$$\Phi_2(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_1x_2, x_2^2]^T \tag{23}$$

, the final hypothesis with $\boldsymbol{w} = [1\ 0\ 0\ -1\ 0\ -1]$ (black dotted line) could be achieved by PLA, which reaches zero in-sample error.

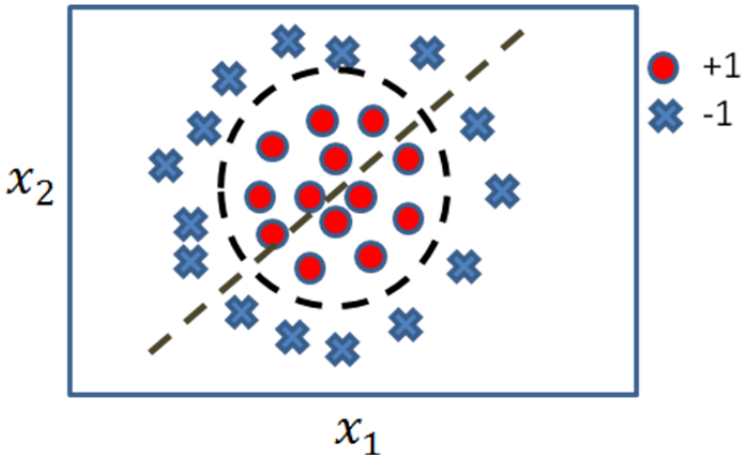


Fig. 17 A non-linear separable training set which could be separated by a linear classifier with feature transform. The gray dotted line is the achieved classifier with no feature transform (1st-order), while the black dotted line is the one with the 2nd-order feature transform.

The feature transform does bring linear classifiers into nonlinear-separable cases, while what feature transform should be used is yet a problem. A higher-order feature transform has a bigger chance to achieve linear-separable boundary, while it may cause over-fitting problem. On the other hand, if the transformed training set is still not linear-separable, the pocket algorithm has no guarantee to achieve a plausibly low in-sample error in T iterations because the updating rule of PLA doesn't ensure monotonic decreasing of in-sample error. In order to speed-up the learning of linear classifiers and confirm its stability, modification on the non-continuous objective function to make other optimization methods available is required.

Table 14 The Adaline algorithm

Presetting:

- Training set: $X = \{\boldsymbol{x}^{(n)} \in R^d\}_{n=1}^N, Y = \{y^{(n)} \in R\}_{n=1}^N, y^{(n)} \in [-1,1]$
- The loss function of a sample is shown in (24), and the objective function of the

training set is defined as: $E_{app}(h_{\boldsymbol{w}}) = \frac{1}{N}\sum_{n=1}^{N}(y^{(n)} - \boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)})^2$

- Preset $\boldsymbol{w}(1)$, usually assume (d+1)-dimensional zero vector.
- The maximum number of iterations T
- Preset the learning step η, ex. $\eta = 0.01$

Learning:

for $t = 1:T$

randomly pick a $\{\boldsymbol{x}^{(n)}, y^{(n)}\}$

if $y^{(n)}(\boldsymbol{w}(t)^T \underline{\boldsymbol{x}}^{(n)}) \le 1$

$\boldsymbol{w}(t+1) \leftarrow \boldsymbol{w}(t) + \eta(y^{(n)} - \boldsymbol{w}(t)^T \underline{\boldsymbol{x}}^{(n)})\underline{\boldsymbol{x}}^{(n)}$

end

end

$g \leftarrow h_{\boldsymbol{w}(T+1)}$

(The learning process is derived from Fig. 12 and Table 16)

5.2.3 Adaptive Perceptron Learning Algorithm- Classification

From this subsection on, several approximated loss functions as well as their optimization procedures and methods will be introduced. The first approximated loss function is a variant of the so-called Adaline (Adaptive linear neuron) algorithm for perceptron learning. The loss function of each sample is modified as:

$$loss^{(n)}(h_{\boldsymbol{w}}) = \begin{cases} (y^{(n)} - \boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)})^2, & \text{if } y^{(n)}(\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)}) \le 1 \\ 0, & \text{otherwise} \end{cases} \quad (24)$$

, which is both continuous and differential at any $\boldsymbol{w}$ with a given data pair, so

Table 15 Concept of gradient descent

Presetting:

- Assume a function $E(\boldsymbol{w})$ is to be minimized, $\boldsymbol{w}$ is k-dimensional
- $\boldsymbol{w}^* = \arg(\nabla E(\boldsymbol{w}) = 0)$ is a solution while sometimes hard to compute due to coupling across parameters and a large summation caused by the training set size.
- Assume now we are at $\boldsymbol{w} = \boldsymbol{w}_0$, and we want to take a small modification $\Delta\boldsymbol{w}$, which makes $E(\boldsymbol{w}_0 + \Delta\boldsymbol{w}) \le E(\boldsymbol{w}_0)$, then Taylor's expansion can be applied.

Taylor's expansion:

scalar w: $E(w_0+\Delta w)=E(w_0)+E'(w_0)\Delta w+\frac{E''(w_0)}{2!}\Delta w^2+H.O.T$ (high order terms)

vector $\boldsymbol{w}$: $E(\boldsymbol{w}_0+\Delta\boldsymbol{w})=E(\boldsymbol{w}_0)+J(\boldsymbol{w}_0)\Delta\boldsymbol{w}+\frac{1}{2!}\Delta\boldsymbol{w}^T H(\boldsymbol{w}_0)\Delta\boldsymbol{w}^T+H.O.T$

, where $J(\boldsymbol{w})$ is the Jacobian matrix: $J(\boldsymbol{w})=\nabla E(\boldsymbol{w})=\begin{bmatrix}\frac{\partial E(\boldsymbol{w})}{\partial w_1}\\ \vdots \\ \frac{\partial E(\boldsymbol{w})}{\partial w_k}\end{bmatrix}$

, and $H(\boldsymbol{w})$ is the Hessian matrix: $H(\boldsymbol{w})=\nabla^2 E(\boldsymbol{w})=\begin{bmatrix}\frac{\partial^2 E(\boldsymbol{w})}{\partial w_1\partial w_1} & \cdots & \frac{\partial^2 E(\boldsymbol{w})}{\partial w_1\partial w_k}\\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\boldsymbol{w})}{\partial w_k\partial w_1} & \cdots & \frac{\partial^2 E(\boldsymbol{w})}{\partial w_k\partial w_k}\end{bmatrix}$

In machine learning, the H.O.T is often discarded.

Concepts of gradient descent:

keep the first two terms: $E(\boldsymbol{w}_0+\Delta\boldsymbol{w})\approx E(\boldsymbol{w}_0)+\nabla E(\boldsymbol{w}_0)\Delta\boldsymbol{w}$

$\Delta\boldsymbol{w}^*=\arg\min_{\|\Delta\boldsymbol{w}\|=\beta} E(\boldsymbol{w}_0+\Delta\boldsymbol{w})\approx\arg\min_{\|\Delta\boldsymbol{w}\|=\beta} E(\boldsymbol{w}_0)+\nabla E(\boldsymbol{w}_0)\Delta\boldsymbol{w}$

$\Delta\boldsymbol{w}^*=-\beta\frac{\nabla E(\boldsymbol{w}_0)}{\|\nabla E(\boldsymbol{w}_0)\|}=-\eta\nabla E(\boldsymbol{w}_0)$, η is set as a small constant for convenience

$E(\boldsymbol{w}_0+\Delta\boldsymbol{w}^*)-E(\boldsymbol{w}_0)\approx\nabla E(\boldsymbol{w}_0)\Delta\boldsymbol{w}^*=-\eta\|\nabla E(\boldsymbol{w}_0)\|^2$

$\boldsymbol{w}_0^{(new)}\leftarrow\boldsymbol{w}_0^{(old)}-\eta\nabla E(\boldsymbol{w}_0)$

Table 16 gradient descent and stochastic gradient descent

Presetting:

- Assume a function $E(\boldsymbol{w})=\frac{1}{N}\sum_{n=1}^{N} loss^{(n)}(\boldsymbol{w})$ is to be minimized.
- Preset $\boldsymbol{w}(1)$, usually assume (d+1)-dimensional zero vector.
- The maximum number of iterations T
- Preset the learning step η, ex. $\eta=0.01$

Algorithm of gradient descent (GD):

for $t = 1:T$

$$\boldsymbol{w}(t+1) \leftarrow \boldsymbol{w}(t) + \eta \nabla E(\boldsymbol{w}(t)) = \boldsymbol{w}(t) + \eta \sum_{n=1}^{N} \nabla loss^{(n)}(\boldsymbol{w}(t))$$

end

$g \leftarrow \boldsymbol{w}(T+1)$

Algorithm of stochastic gradient descent (SGD):

$$E(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} loss^{(n)}(\boldsymbol{w}) = E[loss^{(n)}(\boldsymbol{w})]$$

for $t = 1:T$

randomly choose a n^*

$$\boldsymbol{w}(t+1) \leftarrow \boldsymbol{w}(t) + \eta \nabla loss^{(n^*)}(\boldsymbol{w}(t))$$

end

$g \leftarrow h_{\boldsymbol{w}(T+1)}$

(SGD only perform GD on one sample each time, and perform iteratively.)

differentiation- based optimization methods are available now. Adaline uses stochastic gradient descent (SGD) to search the best hypothesis which minimizes the objective function (without any penalty term). In Table 14, the pseudocode of Adaline algorithm is presented, and in Adaptive Perceptron Learning Algorithm- Classification

From this subsection on, several approximated loss functions as well as their optimization procedures and methods will be introduced. The first approximated loss function is a variant of the so-called Adaline (Adaptive linear neuron) algorithm for perceptron learning. The loss function of each sample is modified as:

$$loss^{(n)}(h_{\boldsymbol{w}}) = \begin{cases} (y^{(n)} - \boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)})^2, & \text{if } y^{(n)}(\boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)}) \le 1 \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

, which is both continuous and differential at any $\boldsymbol{w}$ with a given data pair, so Table 15 and , both the Adaline algorithm and SGD are described in detailed.

From these tables, the mechanism of using gradient descent for optimization is presented. There are several adjustable items of gradient descent algorithm, such as the learning step η and the number of iterations *T*. The learning step should be kept small to fit the requirement of Taylor's expansion. A too small learning step takes more number of iterations towards convergence. On the other hand, a large learning step may spend less number of iterations to converge, while it has chances to diverge out or reach a wrong solution. The number of iterations can be defined before or

during learning, where the modification between $\boldsymbol{w}(t+1)$ and $\boldsymbol{w}(t)$ is used as a measure of convergence, and generally, SGD takes more iteration than GD towards convergence.

GD, SGD, and $\boldsymbol{w}^* = \arg(\nabla E(\boldsymbol{w}) = 0)$ are searching for the "local minima" of $E(\boldsymbol{w})$, which means the achieved final hypothesis *g* may not actually minimizes the objective function. While if $E(\boldsymbol{w})$ is a convex function of $\boldsymbol{w}$, then any local minima of $E(\boldsymbol{w})$ is exactly the global minima of $E(\boldsymbol{w})$. The definition of convexity can be referred to the textbook written by S. Boyd et al. [22]. In fact, the stochastic gradient descent can't even achieve the local minima, but vibrates around it after a number of iterations.

Adaline provides a much stable learning algorithm than PLA. Although the function minimized by Adaline is just an approximated loss function, not directly the in-sample error of the ERM strategy, the in-sample error resulting from the final hypothesis *g* is usually not far away from the minimum value.

5.2.4 Linear Regressor-Regression

As mentioned in Section 5.2, the linear model for classification can also be used for regression by replacing the binary $sign(.)$ function in (20) and (21) into some continuous functions, and the simplest choice is the identity function $f(x) = x$. By doing so, the linear regressor is formulized as:

$$h(\boldsymbol{x}^{(n)}) = \boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)}. \tag{25}$$

After achieving the regression model, we also need to define the objective function and optimization method for training. As mentioned in Section 3.4, the most widely-used criterion for regression is to minimize the root mean square error (RMSE):

$$E(h) = \frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)} - h(\boldsymbol{x}^{(n)})\right|^2 = \frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)} - \boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)}\right|^2. \tag{26}$$

This equation is definitely continuous and differentiable, so gradient descent or stochastic gradient descent can be applied for optimization. Furthermore, because $\left|y^{(n)} - \boldsymbol{w}^T \underline{\boldsymbol{x}}^{(n)}\right|^2$ is a convex function on $\boldsymbol{w}$ and the positive weighted summation of a set of convex functions is still convex [22], the final solution provided by gradient descent is a global minimum solution (or very close to the global minima due to the iteration limitation).

Besides applying the general differentiation optimization methods, there exists a closed form solution for linear regressors with the RMSE criterion. This closed form

solution is quite general, which can be applied not only for linear regressors but also for many optimization problems with the RMSE criterion. In Table 17, we summarize this formulation. To be noticed, for linear regressors with other kinds of objective functions, this closed form solution may not exist. To get more understanding on linear regression and other kinds of objective functions, the textbook [16] is recommended.

Table 17 The closed form solution for linear regression with the RMSE criterion

Presetting:

- A function $E(h)=\frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)}-h(\boldsymbol{x}^{(n)})\right|^2=\frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)}-\boldsymbol{w}^T\underline{\boldsymbol{x}}^{(n)}\right|^2$ is to be minimized.
- $\boldsymbol{w}^*=\arg(\nabla E(h_{\boldsymbol{w}})=0)$ is the global optimal solution.

Closed form solution and its justification:

- $$\frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)}-\boldsymbol{w}^T\underline{\boldsymbol{x}}^{(n)}\right|^2=\frac{1}{N}\left(Y-\boldsymbol{w}^T\underline{X}\right)\left(Y-\boldsymbol{w}^T\underline{X}\right)^T$$
$$=\frac{1}{N}YY^T-2\boldsymbol{w}^T\underline{X}Y^T+\boldsymbol{w}^T\underline{X}\underline{X}^T\boldsymbol{w}$$
- $\nabla_{\boldsymbol{w}}\left\{\frac{1}{N}YY^T-2\boldsymbol{w}^T\underline{X}Y^T+\boldsymbol{w}^T\underline{X}\underline{X}^T\boldsymbol{w}\right\}=-2\underline{X}Y^T+2(\underline{X}\underline{X}^T)\boldsymbol{w}$
- $-2\underline{X}Y^T+2(\underline{X}\underline{X}^T)\boldsymbol{w}^*=0\ \rightarrow\ (\underline{X}\underline{X}^T)\boldsymbol{w}^*=\underline{X}Y^T$, where $\underline{X}\underline{X}^T$ is $d\times d$
- If $\underline{X}\underline{X}^T$ is nonsingular (when $N>d$, it is usually the case), $\boldsymbol{w}^*=(\underline{X}\underline{X}^T)^{-1}\underline{X}Y^T$.
- If $\underline{X}\underline{X}^T$ is singular, other treatments like pseudo-inverse or SVD can be applied.

5.2.5 Rigid Regression-Regression

Linear regressors with the RMSE criterion directly minimize the in-sample RMSE. While in some situations such as high noise in the data or a small sample size, they may suffer from over-fitting and result in poor out-of-sample performances. To deal with this problem, the concept of regularization introduced in Section 4.4 can be applied. In general, linear regression with L^1 regularization is called Lasso regression [16], and linear regression with L^2 regularization is called rigid regression. Lasso regression has the property to find a sparse $\boldsymbol{w}^*$ and can be optimized through linear programming, while in this subsection we mainly focus on rigid regression.

The formulation of rigid regression is as follows:

$$E(h)=\frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)}-\boldsymbol{w}^T\underline{\boldsymbol{x}}^{(n)}\right|^2+\frac{1}{2}\lambda\sum_i|w_i|^2 \tag{27}$$

, where λ is a tradeoff term between the loss function and the penalty function. The penalty term penalizes $\boldsymbol{w}$ with large components, and the algorithm of rigid regression is summarized in Table 18.

Table 18 The rigid regression algorithm

Presetting:

- A function $E(h)=\frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)}-\boldsymbol{w}^T\underline{\boldsymbol{x}}^{(n)}\right|^2+\frac{1}{2}\lambda\sum_i|w_i|^2$ is to be minimized.
- $\boldsymbol{w}^*=\arg(\nabla E(h_{\boldsymbol{w}})=0)$ is the global optimal solution.

Closed form solution and its justification:

- $$\frac{1}{N}\sum_{n=1}^{N}\left|y^{(n)}-\boldsymbol{w}^T\underline{\boldsymbol{x}}^{(n)}\right|^2+\frac{1}{2}\lambda\sum_i|w_i|^2=\frac{1}{N}\left(Y-\boldsymbol{w}^T\underline{X}\right)\left(Y-\boldsymbol{w}^T\underline{X}\right)^T+\frac{1}{2}\lambda\boldsymbol{w}^T\boldsymbol{w}$$
$$=\frac{1}{N}YY^T-2\boldsymbol{w}^T\underline{X}Y^T+\boldsymbol{w}^T\left(\underline{X}\underline{X}^T+\lambda I\right)\boldsymbol{w}$$
- $\nabla_{\boldsymbol{w}}\left\{\frac{1}{N}YY^T-2\boldsymbol{w}^T\underline{X}Y^T+\boldsymbol{w}^T\left(\underline{X}\underline{X}^T+\lambda I\right)\boldsymbol{w}\right\}=-2\underline{X}Y^T+2\left(\underline{X}\underline{X}^T+\lambda I\right)\boldsymbol{w}$
- $-2\underline{X}Y^T+2\left(\underline{X}\underline{X}^T+\lambda I\right)\boldsymbol{w}^*=0\ \rightarrow\ \left(\underline{X}\underline{X}^T+\lambda I\right)\boldsymbol{w}^*=\underline{X}Y^T$, where $\underline{X}\underline{X}^T$ is $d\times d$

$$\boldsymbol{w}^*=\left(\underline{X}\underline{X}^T+\lambda I\right)^{-1}\underline{X}Y^T$$

5.2.6 Support Vector Machine (SVM) and Regression (SVR)

Support vector machine (SVM) which combines the L^2 regularized term as well as the hinge loss function has attracts a significant amount of attention during the past ten years. In many classification and recognition problems, SVM now become the baseline for performance comparison. From the geometrical perspective, SVM tends to find the separating hyperplane that has the largest margin to the nearest positive and negative samples, which gives a unique final hypothesis against the traditional perceptron learning algorithm. And from the theoretical perspective, SVM searches the final hypothesis in a much smaller hypothesis set due to the regularized term. With the help of feature transform, SVM can achieve a nonlinear separating hyperplane.

The original form of SVM aims at finding a separating hyperplane that not only

achieve zero in-sample error but also results in the largest margin, which is usually called the hard-margin SVM. While the error-free constraint often results in a much complicated hyperplane and may over-fit the training data, especially when noise occurs. To deal with this problem, slack variables are introduced to release the error-free constraint. This modification results in a more complicated objective function of SVM, but bring a smoother separating hyperplane and makes the training of SVM more robust to outliers and noise. This type of SVM is usually called the soft-margin SVM.

The objective function of SVM can be modeled as a quadratic programming problem or a constraint quadratic problem (primal form), where the computational time is based on the number of features. So when the feature transform is performed, SVM takes more time for training. While with the dual modification, SVM can be modeled as another constraint quadratic problem whose computational time is based on the number of samples, and the feature transform can be embedded in a more efficient form called the kernel trick. Through the dual modification and kernel trick, complicated feature transforms can be performed on SVM while the computational time won't increase explicitly. In addition, with the large margin concept (regularized term), SVM has lower risk to suffer from over-fitting even with a complicated feature transform.

The idea of kernel trick, large margin, and dual modification can be extended to other problems, such as regression and density estimation. The support vector regression (SVR) which combines the L^2 regularized term and the L^1 loss function (also called ε-sensitive loss function) is a successive example. The LIBSVM [23] developed by C. Chang and C. Lin which exploits sequential minimal optimization (SMO) [24][25] for fast optimization is an efficient and easily-used toolbox for SVM and SVR training, and can be downloaded freely for academic usage.

5.2.7 Extension to Multi-Class Problems

In previous subsections, linear classifiers are defined only for binary classification problems. And in this subsection, we introduce two methods to extend binary classifiers into multi-class classifiers: one-versus-one (OVO) and one-versus-all (OVA) [26].

- **One-versus-one (OVO):** Assume that there are totally c classes. OVO builds a binary classifier for each pair of classes, which means totally $c(c-1)/2$ binary classifiers are built. When given an input sample $\boldsymbol{x}$, each classifier predicts a

possible class label, and the final predicted label is the one with the most votes among all $c(c-1)/2$ classifiers.

- **One-versus-all (OVA):** OVA build a binary classifier for each class (positive) against all other classes (negative), which means totally c binary classifiers are built. When given an input sample $\boldsymbol{x}$, the class label corresponded to the classifier which gives a positive decision on $\boldsymbol{x}$ is selected as the final predicted label.

In general, the OVA method suffers from two problems. The first problem is that there may be more than one positive class or no positive class, and the second one is the unbalance problem. The unbalance problem means the number of positive training samples is either much larger or smaller than the number of negative training samples. In this condition, the trained classifier will tend to always predict the class with more training samples and lead to poor performances for unseen samples. For example, if there are 100 positive training samples and 9900 negative samples, always predicting the negative class could simply results in a 0.01 in-sample error. Although the OVO method doesn't suffer from these problems, it needs to build more binary classifiers ($(c-1)/2$ times more) than the OVA method, which is of highly computational cost especially when c is large.

There are also other methods to extend binary linear classifiers into multi-class linear classifiers, either based on a similar concept of OVO and OVA or from theoretical modifications. And for non-metric, non-parametric, and parametric models, multi-class classifiers are usually embedded in the basic formulation without extra modifications.

5.3 Conclusion and Summary

In this section, we give an overview of supervised learning techniques. Generally, supervised learning techniques can be categorized into linear models, parametric models, nonparametric models, and non-metric models, and the aggregation methods that combine several classifiers or regressors together usually improve the learning performance. The linear model is probably the most fundamental while powerful supervised learning technique, and several techniques as well as their optimization methods are introduced and listed in this section. At the end, two simple treatments to extend binary classifiers into multi-class classifiers are presented.

6. Techniques of Unsupervised Learning

In this section, we briefly introduce the techniques of unsupervised learning and its categorization. Compared to supervised learning, unsupervised learning only acquires the feature set, not the label set. As mentioned in Section 3.2, the main goal of unsupervised learning can be categorized into clustering, probability density estimation, and dimensionality reduction:

- **Clustering:** Given a set of samples, clustering aims to separate them into several groups based on some kinds of similarity / distance measures, and the basic criterion for doing so is to minimize the intra-group distance while maximize the inter-group distance. Clustering can discover the underlying structure in the samples, which is very important in applications such as business and medical issues: Separating the customers or patients into groups based on their attributes and designing specific strategies or treatments for each group. In addition, the discovered groups can be used as the label of each sample, and then the supervised learning techniques can be applied for further applications.

- **Probability estimation:** Given a set of samples, probability estimation aims to estimate the appearing probability of each sample vector, either via parametric models, semi-parametric models, or non-parametric models. By doing so, the underlying mechanism to generate these samples can be discovered, and the learned probability distribution can further be used for outlier and special event detections.

- **Dimensionality reduction:** Given a set of samples, dimensionality reduction aims to learn the relationship between features and discover the latent factors that control the feature values of a sample. By doing so, a compact representation can be derived for each sample, which are more informative for the subsequent applications such as pattern recognition and retrieval and can prevent the possible over-fitting problems during learning. We will discuss this category in more detailed in another suvery.

In Table 19, several important techniques for each category as well as the important references are listed for further studying.

Table 19 The category and important techniques for unsupervised learning

Category	Techniques	Reference
Clustering	K-means clustering	[8]
	Spectral clustering	[27][28]
Density Estimation	Gaussian mixture model (GMM)	[9]
	Graphical models	[9][15]
Dimensionality reduction	Principal component analysis (PCA)	[8]
	Factor analysis	[8]

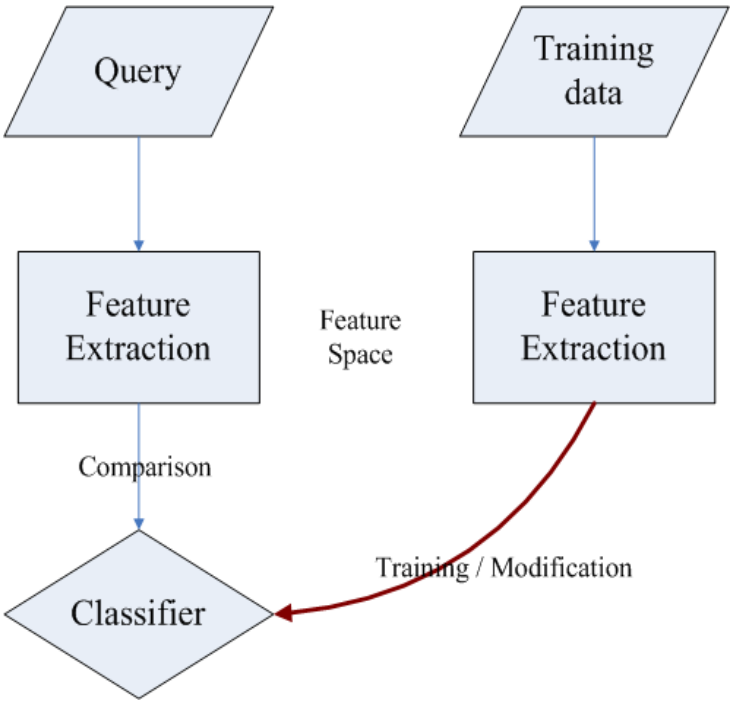


Fig. 18 The general framework of pattern recognition.

7. Practical Usage: Pattern Recognition

There are many applications of machine learning, such as economic, information retrieval, data mining, and pattern recognition. In this section, we briefly introduce the framework of pattern recognition and discuss how machine learning techniques can be applied in this framework.

The general framework of pattern recognition is presented in Fig. 18. Given a set of observations or raw data, the first step of pattern recognition is to extract important features from these observations, either based on domain knowledge or data-driven

knowledge. This step not only discards irrelevant information for recognition, but also reshapes these observations into a unified form (ex. vector or matrix). After the feature extraction step, the next step of pattern recognition is to build a classifier or recognizer that can perform class prediction or detection based on extracted features. Now when a new observation (query) is acquired, the pre-defined feature extraction step and classifier are performed to predict the corresponding class label or event occurrence.

In this framework, machine learning techniques can be applied in both the two steps. For example, in the feature extraction step, the dimensionality reduction techniques can be applied to extract relevant features and reduce the dimensionality of raw data. And in the second step, the supervised learning techniques introduced in Section **錯誤! 找不到參照來源。** can be directly used for classifier training.

However, when applying machine learning techniques in pattern recognition, some issues should be considered. At first, the observations or raw data often contain high noise and high dimensionality, which can degrade the overall learning performance and result in over-fitting. Second, the form of raw data may not be in the vector form. For example, images are in the form of 2D matrices, and videos are 3D matrices. These types of data have their intrinsic data structures (spatial and temporal relationship), so how to transfer them into the standard form of machine learning while maintaining the intrinsic structures is a key issue in pattern recognition. In general, when doing pattern recognition researches, both the domain knowledge and the theory of machine learning should be considered to achieve better recognition performance.

8. Conclusion

In this tutorial, a broad overview of machine learning containing both theoretical and practical aspects is presented. Machine learning is generally composed of modeling (hypothesis set + objective function) and optimization, and the necessary part to perform machine learning is a suitable dataset for knowledge learning. For the theoretical aspect, we introduce the basic idea, categorization, structure, and criteria of machine learning. And for the practical aspect, several principles and techniques of both unsupervised and supervised learning are presented in this tutorial.

9. Reference

[1] M. Turk and A. Pentland, “Eigenfaces for recognition,” *Journal of Cognitive*

Neuroscience, vol. 3, no.1, pp. 72-86, 1991.

[2] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, 711-720, 1997.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 511 -518, 2001.

[4] P. Viola and M. Jones, "Robust real-time face detection," *Int'l Journal of Computer Vision*, vol. 57, no. 2, pp. 137-154, 2004.

[5] Flicker: http://www.flickr.com/

[6] Facebook: http://www.facebook.com/

[7] Youtube: http://www.youtube.com/

[8] E. Alpaydin, *Introduction to machine learning*, 2nd ed., The MIT Press, 2010.

[9] C. M. Bishop, *Pattern recognition and machine learning*, Springer, 2006.

[10] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables," *American Statistical Association Journal*, vol. 58, pp. 13-30, March 1963.

[11] K. Sayood, *Introduction to Data Compression*, Morgan Kaufmann Publishers, 1996.

[12] R. Xu, D.I.I. Wunsch, "Survey of clustering algorithms," *IEEE Trans.Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[13] I.K. Fodor, "A survey of dimension reduction techniques," Technical report UCRL-ID-148494, LLNL, 2002.

[14] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," J. Artif. Intell. Res. 4, pp. 237-285, 1996.

[15] D. Koller and N. Friedman, *Probabilistic Graphical Models*, MIT Press, 2009.

[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer, 2005.

[17] J. L. Schafer, J. W. Graham, "Missing data: our view of the state of the art," *Psychological Methods*, vol. 7, no. 2, pp. 147-177, 2002.

[18] KDD Cup 2009: http://www.kddcup-orange.com/

[19] I. Guyon, V. Lemaire, G. Dror, and D. Vogel, "Analysis of the KDD cup 2009: Fast scoring on a large orange customer database," *JMLR*: Workshop and Conference Proceedings, vol. 7, pp. 1-22, 2009.

[20] H. Y. Lo et al, "An ensemble of three classifiers for KDD cup 2009: Expanded linear model, heterogeneous boosting, and selective naive Bayes," *In JMLR W&CP*, vol.7, KDD cup 2009, Paris, 2009.

[21] A. Niculescu-Mizil, C. Perlich, G. Swirszcz, V. Sindhwani, Y. Liu, P. Melville, D. Wang, J. Xiao, J. Hu, M. Singh, et al, "Winning the KDD Cup Orange Challenge with Ensemble Selection," In KDD Cup and Workshop in conjunction with KDD 2009, 2009.

[22] S. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, 2004.

[23] C. C. Chang and C. J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[24] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," in *Advances in Kernel Methods - Support Vector Learning*, MIT Press, pp. 185-208, 1999.

[25] R. E. Fan, P. H. Chen, and C. J. Lin, "Working set selection using second order information for training SVM," *Journal of Machine Learning Research*, 6:1889{1918, 2005.

[26] C. W. Hsu and C. J. Lin, "A comparison of methods for multi-class support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415-425, 2002.

[27] U. Von Luxburg, "A Tutorial on Spectral Clustering," Tech. Rep. TR-149, Max Plank Institute for Biological Cybernetics, 2006.

[28] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: analysis and an algorithm," *Advances in Neural Information Processing Systems*, vol. 14, 2002.