1 Introduction and Problem Formulation

In this chapter, we provide an introduction to covariate shift adaptation toward machine learning in a non-stationary environment.

1.1 Machine Learning under Covariate Shift

Machine learning is an interdisciplinary field of science and engineering studying that studies mathematical foundations and practical applications of systems that learn. Depending on the type of learning, paradigms of machine learning can be categorized into three types:

- *Supervised learning* The goal of supervised learning is to infer an underlying input–output relation based on input–output samples. Once the underlying relation can be successfully learned, output values for unseen input points can be predicted. Thus, the learning machine can generalize to unexperienced situations. Studies of supervised learning are aimed at letting the learning machine acquire the best generalization performance from a small number of training samples. The supervised learning problem can be formulated as a function approximation problem from samples.
- *Unsupervised learning* In contrast to supervised learning, output values are not provided as training samples in unsupervised learning. The general goal of unsupervised learning is to extract valuable information hidden behind data. However, its specific goal depends heavily on the situation, and the unsupervised learning problem is sometimes not mathematically well-defined. Data clustering aimed at grouping similar data is a typical example. In data clustering, how to measure the similarity between data samples needs to be predetermined, but there is no objective criterion that can quantitatively evaluate the validity of the affinity measure; often it is merely subjectively determined.

• *Reinforcement learning* The goal of reinforcement learning is to acquire a policy function (a mapping from a state to an action) of a computer agent. The policy function is an input–output relation, so the goal of reinforcement learning is the same as that of supervised learning. However, unlike supervised learning, the output data cannot be observed directly. Therefore, the policy function needs to be learned without supervisors. However, in contrast to unsupervised learning, rewards are provided as training samples for an agent's action. Based on the reward information, reinforcement learning tries to learn the policy function in such a way that the sum of rewards the agent will receive in the future is maximized.

The purpose of this book is to provide a comprehensive overview of theory, algorithms, and applications of supervised learning under the situation called *covariate shift*.

When developing methods of supervised learning, it is commonly assumed that samples used as a training set and data points used for testing the generalization performance[1] follow the same probability distribution (e.g., [195, 20, 193, 42, 74, 141]). However, this common assumption is not fulfilled in recent real-world applications of machine learning such as robot control, brain signal analysis, and bioinformatics. Thus, there is a strong need for theories and algorithms of supervised learning under such a changing environment. However, if there is no connection between training data and test data, nothing about test data can be learned from training samples. This means that a reasonable assumption is necessary for relating training samples to test data.

Covariate shift is one of the assumptions in supervised learning. The situation where the training input points and test input points follow different probability distributions, but the conditional distributions of output values given input points are unchanged, is called the covariate shift [145]. This means that the target function we want to learn is unchanged between the training phase and the test phase, but the distributions of input points are different for training and test data.

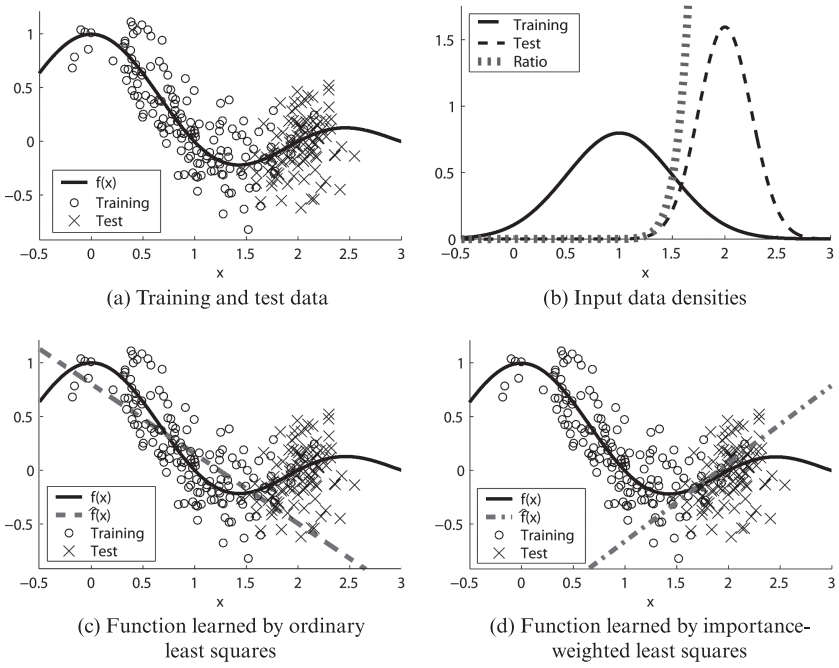
A situation of supervised learning where input-only samples are available in addition to input–output samples is called semisupervised learning [30]. The covariate shift adaptation techniques covered in this book fall into the category of semisupervised learning since input-only samples drawn from the test distribution are utilized for improving the generalization performance under covariate shift.

1. Such test points are not available during the training phase; they are given in the future after training has been completed.

1.2 Quick Tour of Covariate Shift Adaptation

Before going into the technical detail, in this section we briefly describe the core idea of covariate shift adaptation, using an illustative example. To this end, let us consider a regression problem of learning a function $f(\boldsymbol{x})$ from its samples $\{(\boldsymbol{x}_i^{\mathrm{tr}}, y_i^{\mathrm{tr}})\}_{i=1}^{n_{\mathrm{tr}}}$. Once a good approximation fuction $\widehat{f}(\boldsymbol{x})$ is obtained, we can predict the output value y^{te} at an unseen test input point $\boldsymbol{x}^{\mathrm{te}}$ by means of $\widehat{f}(\boldsymbol{x}^{\mathrm{te}})$.

Let us consider a covariate shift situation where the training and test input points follow different probability distributions, but the learning target function $f(\boldsymbol{x})$ is common to both training and test samples. In the toy regression example illustrated in figure 1.1a, training samples are located in the left-hand side of the graph and test samples are distributed in the right-hand side. Thus, this is an *extrapolation problem* where the test samples are located outside the



(a) Training and test data

(b) Input data densities

(c) Function learned by ordinary least squares

(d) Function learned by importance-weighted least squares

Figure 1.1
A regression example with covariate shift. (a) The learning target function $f(x)$ (the solid line), training samples ($\circ$), and test samples ($\times$). (b) Probability density functions of training and test input points and their ratio. (c) Learned function $\widehat{f}(x)$ (the dashed line) obtained by ordinary least squares. (d) Learned function $\widehat{f}(x)$ (the dashed-dotted line) obtained by importance-weighted least squares. Note that the test samples are not used for function learning.

training region. Note that the test samples are not given to us in the training phase; they are plotted in the graph only for illustration purposes. The probability densities of the training and test input points, $p_{\mathrm{tr}}(x)$ and $p_{\mathrm{te}}(x)$, are plotted in figure 1.1b.

Let us consider straight-line function fitting by the method of least squares:

$$\min_{\theta_1,\theta_2}\left[\sum_{i=1}^{n_{\mathrm{tr}}}\left(\widehat{f}(x_i^{\mathrm{tr}})-y_i^{\mathrm{tr}}\right)^2\right],$$

where

$$\widehat{f}(x)=\theta_1 x+\theta_2.$$

This ordinary least squares gives a function that goes through the training samples well, as illustrated in figure 1.1c. However, the function learned by least squares is not useful for predicting the output values of the test samples located in the right-hand side of the graph.

Intuitively, training samples that are far from the test region (say, training samples with $x<1$ in figure 1.1a) are less informative for predicting the output values of the test samples located in the right-hand side of the graph. This gives the idea that ignoring such less informative training samples and learning only from the training samples that are close to the test region (say, training samples with $x>1.2$ in figure 1.1a) is more promising. The key idea of covariate shift adaptation is to (softly) choose informative training samples in a systematic way, by considering the *importance* of each training sample in the prediction of test output values. More specifically, we use the ratio of training and test input densities (see figure 1.1b),

$$\frac{p_{\mathrm{te}}(x_i^{\mathrm{tr}})}{p_{\mathrm{tr}}(x_i^{\mathrm{tr}})},$$

as a weight for the i-th training sample in the least-squares fitting:

$$\min_{\theta_1,\theta_2}\left[\sum_{i=1}^{n_{\mathrm{tr}}}\frac{p_{\mathrm{te}}(x_i^{\mathrm{tr}})}{p_{\mathrm{tr}}(x_i^{\mathrm{tr}})}\left(\widehat{f}(x_i^{\mathrm{tr}})-y_i^{\mathrm{tr}}\right)^2\right].$$

Then we can obtain a function that extrapolates the test samples well (see figure 1.1d). Note that the test samples are *not* used for obtaining this function. In this example, the training samples located in the left-hand side of the graph (say, $x<1.2$) have almost zero importance (see figure 1.1b). Thus, these samples are essentially ignored in the above importance-weighted least-squares

method, and informative samples in the middle of the graph are automatically selected by importance weighting.

As illustrated above, importance weights play an essential role in covariate shift adaptation. Below, the problem of covariate shift adaptation is formulated more formally.

1.3 Problem Formulation

In this section, we formulate the supervised learning problem, which includes regression and classification. We pay particular attention to covariate shift and *model misspecification*; these two issues play the central roles in the following chapters.

1.3.1 Function Learning from Examples

Let us consider the supervised learning problem of estimating an unknown input–output dependency from training samples. Let

$$\{(\boldsymbol{x}_i^{\mathrm{tr}}, y_i^{\mathrm{tr}})\}_{i=1}^{n_{\mathrm{tr}}}$$

be the training samples, where the training input point

$$\boldsymbol{x}_i^{\mathrm{tr}} \in \mathcal{X} \subset \mathbb{R}^d, \quad i = 1, 2, \ldots, n_{\mathrm{tr}}$$

is an *independent and identically distributed* (i.i.d.) sample following a probability distribution $P_{\mathrm{tr}}(\boldsymbol{x})$ with density $p_{\mathrm{tr}}(\boldsymbol{x})$:

$$\{\boldsymbol{x}_i^{\mathrm{tr}}\}_{i=1}^{n_{\mathrm{tr}}} \overset{i.i.d.}{\sim} P_{\mathrm{tr}}(\boldsymbol{x}).$$

The training output value

$$y_i^{\mathrm{tr}} \in \mathcal{Y} \subset \mathbb{R}, \quad i = 1, 2, \ldots, n_{\mathrm{tr}}$$

follows a conditional probability distribution $P(y|\boldsymbol{x})$ with conditional density $p(y|\boldsymbol{x})$.

$$y_i^{\mathrm{tr}} \sim P(y|\boldsymbol{x} = \boldsymbol{x}_i^{\mathrm{tr}}).$$

$P(y|\boldsymbol{x})$ may be regarded as the superposition of the true output $f(\boldsymbol{x})$ and noise ϵ:

$$y = f(\boldsymbol{x}) + \epsilon.$$

We assume that noise ϵ has mean 0 and variance σ^2. Then the function $f(\boldsymbol{x})$ coincides with the conditional mean of y given $\boldsymbol{x}$.

The above formulation is summarized in figure 1.2.

1.3.2 Loss Functions

Let $\mathrm{loss}(\boldsymbol{x}, y, \widehat{y})$ be the loss function which measures the discrepancy between the true output value y at an input point $\boldsymbol{x}$ and its estimate $\widehat{y}$. In the regression scenarios where $\mathcal{Y}$ is continuous, the *squared loss* is often used.

$$\mathrm{loss}(\boldsymbol{x}, y, \widehat{y}) = (\widehat{y} - y)^2.$$

On the other hand, in the binary classification scenarios where $\mathcal{Y} = \{+1, -1\}$, the following *0/1-loss* is a typical choice since it corresponds to the *misclassification rate*.

$$\mathrm{loss}(\boldsymbol{x}, y, \widehat{y}) = \begin{cases} 0 & \text{if } \mathrm{sgn}(\widehat{y}) = y, \\ 1 & \text{otherwise,} \end{cases}$$

where $\mathrm{sgn}(\widehat{y})$ denotes the sign of $\widehat{y}$:

$$\mathrm{sgn}(\widehat{y}) := \begin{cases} +1 & \text{if } \widehat{y} > 0, \\ 0 & \text{if } \widehat{y} = 0, \\ -1 & \text{if } \widehat{y} < 0. \end{cases}$$

Although the above loss functions are independent of $\boldsymbol{x}$, the loss can generally depend on $\boldsymbol{x}$ [141].

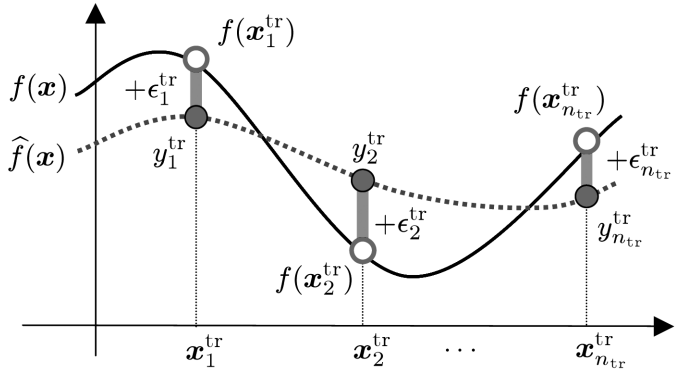


Figure 1.2
Framework of supervised learning.

1.3.3 Generalization Error

Let us consider a test sample $(\boldsymbol{x}^{\mathrm{te}}, y^{\mathrm{te}})$, which is not given in the training phase but will be given in the test phase. $\boldsymbol{x}^{\mathrm{te}} \in \mathcal{X}$ is a test input point following a test distribution $P_{\mathrm{te}}(\boldsymbol{x})$ with density $p_{\mathrm{te}}(\boldsymbol{x})$, and $y^{\mathrm{te}} \in \mathcal{Y}$ is a test output value following the conditional distribution $P(y|\boldsymbol{x} = \boldsymbol{x}^{\mathrm{te}})$ with conditional density $p(y|\boldsymbol{x} = \boldsymbol{x}^{\mathrm{te}})$. Note that the conditional distribution is common to both training and test samples. The test error expected over all test samples (or the *generalization error*) is expressed as

$$\mathrm{Gen} = \mathop{\mathbb{E}}_{\boldsymbol{x}^{\mathrm{te}}} \mathop{\mathbb{E}}_{y^{\mathrm{te}}} \left[\mathrm{loss}(\boldsymbol{x}^{\mathrm{te}}, y^{\mathrm{te}}, \widehat{f}(\boldsymbol{x}^{\mathrm{te}}; \boldsymbol{\theta})) \right],$$

where $\mathbb{E}_{\boldsymbol{x}^{\mathrm{te}}}$ denotes the expectation over $\boldsymbol{x}^{\mathrm{te}}$ drawn from $P_{\mathrm{te}}(\boldsymbol{x})$ and $\mathbb{E}_{y^{\mathrm{te}}}$ denotes the expectation over y^{te} drawn from $P(y|\boldsymbol{x} = \boldsymbol{x}^{\mathrm{te}})$. The goal of supervised learning is to determine the value of the parameter $\boldsymbol{\theta}$ so that the generalization error is minimized, that is, output values for unseen test input points can be accurately estimated in terms of the expected loss.

1.3.4 Covariate Shift

In standard supervised learning theories (e.g., [195,20,193,42,74,141,21]), the test input distribution $P_{\mathrm{te}}(\boldsymbol{x})$ is assumed to agree with the training input distribution $P_{\mathrm{tr}}(\boldsymbol{x})$. However, in this book we consider the situation under covariate shift [145], that is, the test input distribution $P_{\mathrm{te}}(\boldsymbol{x})$ and the training input distribution $P_{\mathrm{tr}}(\boldsymbol{x})$ are generally different:

$$P_{\mathrm{tr}}(\boldsymbol{x}) \neq P_{\mathrm{te}}(\boldsymbol{x}).$$

Under covariate shift, most of the standard machine learning techniques do not work properly due to the differing distributions. The main goal of this book is to provide machine learning methods that can mitigate the influence of covariate shift.

In the following chapters, we assume that the ratio of test to training input densities is bounded, that is,

$$\frac{p_{\mathrm{te}}(\boldsymbol{x})}{p_{\mathrm{tr}}(\boldsymbol{x})} < \infty \text{ for all } \boldsymbol{x} \in \mathcal{X}.$$

This means that the support of the test input distribution must be contained in that of the training input distribution. The above ratio is called the *importance* [51], and it plays a central role in covariate shift adaptation.

1.3.5 Models for Function Learning

Let us employ a parameterized function $\widehat{f}(\boldsymbol{x};\boldsymbol{\theta})$ for estimating the output value y, where

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_b)^\top \in \boldsymbol{\Theta} \subset \mathbb{R}^b.$$

Here, $^\top$ denotes the transpose of a vector or a matrix, and $\boldsymbol{\Theta}$ denotes the domain of parameter $\boldsymbol{\theta}$.

1.3.5.1 Linear-in-Input Model The simplest choice of parametric model would be the *linear-in-input model*:

$$\widehat{f}(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{k=1}^{d} \theta_k x^{(k)} + \theta_{d+1}, \tag{1.1}$$

where

$$\boldsymbol{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(d)})^\top.$$

This model has linearity in both input variable $\boldsymbol{x}$ and parameter $\boldsymbol{\theta}$, and the number b of parameters is $d+1$, where d is the dimensionality of $\boldsymbol{x}$. The linear-in-input model can represent only a linear input–output relation, so its expressibility is limited. However, since the effect of each input variable $x^{(k)}$ can be specified directly by the parameter θ_k, it would have high interpretability. For this reason, this simple model is still often used in many practical data analysis tasks such as natural language processing, bioinformatics, and computational chemistry.

1.3.5.2 Linear-in-Parameter Model A slight extension of the linear-in-input model is the *linear-in-parameter* model:

$$\widehat{f}(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{\ell=1}^{b} \theta_\ell \varphi_\ell(\boldsymbol{x}), \tag{1.2}$$

where $\{\varphi_\ell(\boldsymbol{x})\}_{\ell=1}^b$ are fixed, linearly independent functions. This model is linear in parameter $\boldsymbol{\theta}$, and we often refer to it as the *linear model*. Popular choices of basis functions include *polynomials* and *trigonometric polynomials*.

When the input dimensionality is $d = 1$, the polynomial basis functions are given by

$$\{\varphi_\ell(x)\}_{\ell=1}^b = \{1, x, x^2, \ldots, x^t\},$$

where $b = t + 1$. The trigonometric polynomial basis functions are given by

$$\{\varphi_\ell(x)\}_{\ell=1}^{b} = \{1, \sin x, \cos x, \sin 2x, \cos 2x, \ldots, \sin cx, \cos cx\},$$

where $b = 2c + 1$.

For multidimensional cases, basis functions are often built by combining one-dimensional basis functions. Popular choices include the *additive model* and the *multiplicative model*. The additive model is given by

$$\widehat{f}(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{k=1}^{d} \sum_{\ell=1}^{c} \theta_{k,\ell} \varphi_\ell(x^{(k)}).$$

Thus, a one-dimensional model for each dimension is combined with the others in an additive manner (figure 1.3a). The number of parameters in the additive model is

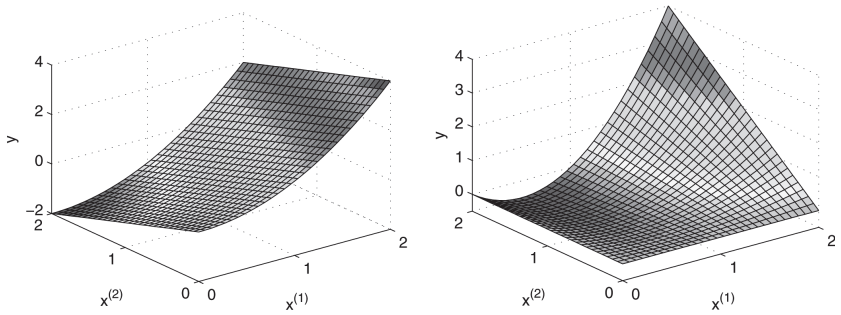
$$b = cd.$$

The multiplicative model is given by

$$\widehat{f}(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{\ell_1, \ell_2, \ldots, \ell_d = 1}^{c} \theta_{\ell_1, \ell_2, \ldots, \ell_d} \prod_{k=1}^{d} \varphi_{\ell_k}(x^{(k)}).$$

Thus, a one-dimensional model for each dimension is combined with the others in a multiplicative manner (figure 1.3b). The number of parameters in the multiplicative model is

$$b = c^d.$$



(a) Additive model (b) Multiplicative model

Figure 1.3
Examples of an additive model $\widehat{f}(\boldsymbol{x}) = (x^{(1)})^2 - x^{(2)}$ and of a multiplicative model $\widehat{f}(\boldsymbol{x}) = -x^{(1)}x^{(2)} + x^{(1)}(x^{(2)})^2$.

In general, the multiplicative model can represent more complex functions than the additive model (see figure 1.3). However, the multiplicative model contains exponentially many parameters with respect to the input dimensionality d—such a phenomenon is often referred to as the curse of dimensionality [12]. Thus, the multiplicative model is not tractable in high-dimensional problems. On the other hand, the number of parameters in the additive model increases only linearly with respect to the input dimensionality d, which is more preferable in high-dimensional cases [71].

1.3.5.3 Kernel Model The number of parameters in the linear-in-parameter model is related to the input dimensionality d. Another means for determining the number of parameters is to relate the number of parameters to the number of training samples, n_{tr}. The *kernel model* follows this idea, and is defined by

$$\widehat{f}(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{\ell=1}^{n_{\mathrm{tr}}} \theta_\ell K(\boldsymbol{x}, \boldsymbol{x}_\ell^{\mathrm{tr}}),$$

where $K(\cdot,\cdot)$ is a *kernel function*. The *Gaussian kernel* would be a typical choice (see figure 1.4):

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2h^2}\right), \tag{1.3}$$

where h (>0) controls the width of the Gaussian function.

In the kernel model, the number b of parameters is set to n_{tr}, which is independent of the input dimensionality d. For this reason, the kernel model is often preferred in high-dimensional problems. The kernel model is still linear in parameters, so it is a kind of linear-in-parameter model; indeed, letting

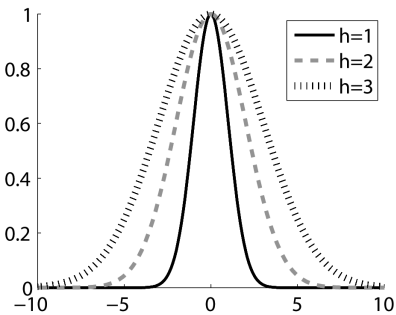


Figure 1.4
Gaussian functions (equation 1.3) centered at the origin with width h.

$b = n_{\mathrm{tr}}$ and $\varphi_\ell(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x}_\ell^{\mathrm{tr}})$ in the linear-in-parameter model (equation 1.2) yields the kernel model. Thus, many learning algorithms explained in this book could be applied to both models in the same way.

However, when we discuss convergence properties of the learned function $\widehat{f}(\boldsymbol{x}; \boldsymbol{\theta})$ when the number of training samples is increased to infinity, the kernel model should be treated differently from the linear-in-parameter model because the number of parameters increases as the number of training samples grows. In such a case, standard asymptotic analysis tools such as the *Cramér-Rao paradigm* are not applicable. For this reason, statisticians categorize the linear-in-parameter model and the kernel model in different classes: the linear-in-parameter model is categorized as a *parametric model*, whereas the kernel model is categorized as a *nonparametric model*. Analysis of the asymptotic behavior of nonparametric models is generally more difficult than that of parametric models, and highly sophisticated mathematical tools are needed (see, e.g., [191, 192, 69]).

A practical compromise would be to use a fixed number of kernel functions, that is, for fixed b,

$$\widehat{f}(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{\ell=1}^{b} \theta_\ell K(\boldsymbol{x}, \boldsymbol{c}_\ell),$$

where, for example, $\{\boldsymbol{c}_\ell\}_{\ell=1}^{b}$ are template points for example chosen randomly from the domain or from the training input points $\{\boldsymbol{x}_i^{\mathrm{tr}}\}_{i=1}^{n_{\mathrm{tr}}}$ without replacement.

1.3.6 Specification of Models

A model $\widehat{f}(\boldsymbol{x}; \boldsymbol{\theta})$ is said to be correctly specified if there exists a parameter $\boldsymbol{\theta}^*$ such that

$$\widehat{f}(\boldsymbol{x}; \boldsymbol{\theta}^*) = f(\boldsymbol{x}).$$

Otherwise, the model is said to be *misspecified*. In practice, the model used for learning would be misspecified to a greater or lesser extent since we do not generally have strong enough prior knowledge to correctly specify the model. Thus, it is important to consider misspecified models when developing machine learning algorithms.

On the other hand, it is meaningless to discuss properties of learning algorithms if the model is *totally* misspecified—for example, approximating highly nonlinearly fluctuated functions by a straight line does not provide meaningful prediction (figure 1.5). Thus, we effectively consider the situation where the model at hand is not correctly specified but is approximately correct.

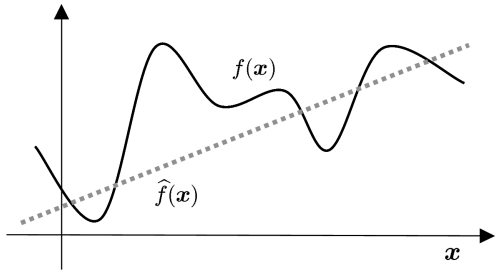


Figure 1.5
Approximating a highly nonlinear function $f(\boldsymbol{x})$ by the linear-in-input model $\widehat{f}(\boldsymbol{x})$, which is totally misspecified.

This approximate correctness plays an important role when designing model selection algorithms (chapter 3) and active learning algorithms (chapter 8).

1.4 Structure of This Book

This book covers issues related to the covariate shift problems, from fundamental learning algorithms to state-of-the-art applications.

Figure 1.6 summarizes the structure of chapters.

1.4.1 Part II: Learning under Covariate Shift

In part II, topics on learning under covariate shift are covered.

In chapter 2, function learning methods under covariate shift are introduced. Ordinary *empirical risk minimization* learning is not consistent under covariate shift for misspecified models, and this inconsistency issue can be resolved by considering *importance-weighted loss functions*. Here, various importance-weighted empirical risk minimization methods are introduced, including *least squares* and *Huber's method* for regression, and *Fisher discriminant analysis*, *logistic regression*, *support vector machines*, and *boosting* for classification. Their adaptive and regularized variants are also introduced. The numerical behavior of these importance-weighted learning methods is illustrated through experiments.

In chapter 3, the problem of model selection is addressed. Success of machine learning techniques depends heavily on the choice of *hyperparameters* such as *basis functions*, the *kernel bandwidth*, the *regularization parameter*, and the *importance-flattening parameter*. Thus, model selection is one of the most fundamental and crucial topics in machine learning. Standard model selection schemes such as the *Akaike information criterion*, *cross-validation*, and the *subspace information criterion* have their own theoretical justification

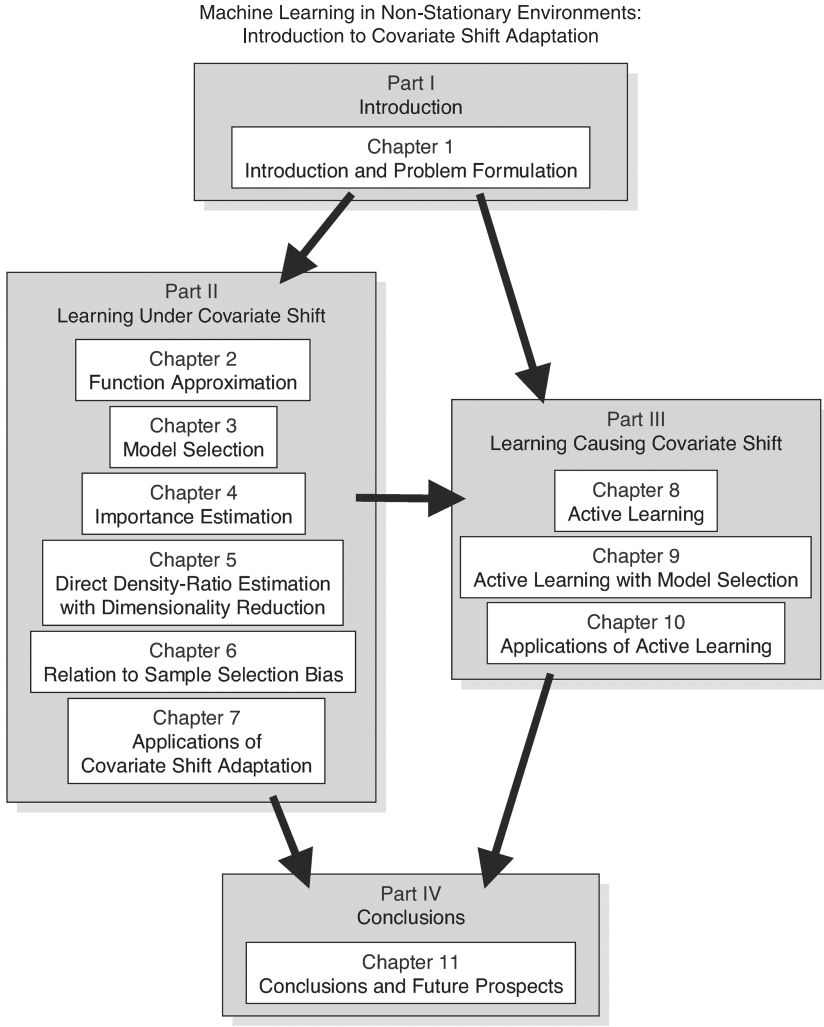


Figure 1.6
Structure of this book.

in terms of the unbiasedness as generalization error estimators. However, such theoretical guarantees are no longer valid under covariate shift. In this chapter, various their modified variants using importance-weighting techniques are introduced, and the modified methods are shown to be properly unbiased even under covariate shift. The usefulness of these modified model selection criteria is illustrated through numerical experiments.

In chapter 4, the problem of *importance estimation* is addressed. As shown in the preceding chapters, importance-weighting techniques play essential roles in covariate shift adaptation. However, the importance values are usually unknown a priori, so they must be estimated from data samples. In this chapter, importance estimation methods are introduced, including importance estimation via *kernel density estimation*, the *kernel mean matching* method, a *logistic regression* approach, the *Kullback–Leibler importance estimation procedure*, and the *least-squares importance fitting* methods. The latter methods allow one to estimate the importance weights without performing through density estimation. Since density estimation is known to be difficult, the direct importance estimation approaches would be more accurate and preferable in practice. The numerical behavior of direct importance estimation methods is illustrated through experiments. Characteristics of importance estimation methods are also discussed.

In chapter 5, a dimensionality reduction scheme for density-ratio estimation, called *direct density-ratio estimation with dimensionality reduction* (D^3; pronounced as "D-cube"), is introduced. The basic idea of D^3 is to find a low-dimensional subspace in which training and test densities are significantly different, and estimate the density ratio only in this subspace. A supervised dimensionality reduction technique called *local Fisher discriminant analysis* (LFDA) is employed for identifying such a subspace. The usefulness of the D^3 approach is illustrated through numerical experiments.

In chapter 6, the covariate shift approach is compared with related formulations called *sample selection bias*. Studies of correcting sample selection bias were initiated by Heckman [77,76], who received the Nobel Prize in economics for this achievement in 2000. We give a comprehensive review of Heckman's correction model, and discuss its relation to covariate shift adaptation.

In chapter 7, state-of-the-art applications of covariate shift adaptation techniques to various real-world problems are described. This chapter includes non-stationarity adaptation in brain–computer interfaces, speaker identification through change in voice quality, domain adaptation in natural language processing, age prediction from face images under changing illumination conditions, user adaptation in human activity recognition, and efficient sample reuse in autonomous robot control.

1.4.2 Part III: Learning Causing Covariate Shift

In part III, we discuss the situation where covariate shift is intentionally caused by users in order to improve generalization ability.

In chapter 8, the problem of *active learning* is addressed. The goal of active learning is to find the most "informative" training input points so that learning can be successfully achieved from only a small number of training samples. Active learning is particularly useful when the cost of data sampling is expensive. In the active learning scenario, covariate shift—mismatch of training and test input distributions—occurs naturally occurs since the training input distribution is designed by users, while the test input distribution is determined by the environment. Thus, covariate shift is inevitable in active learning. In this chapter, active learning methods for regression are introduced in light of covariate shift. Their mutual relation and numerical examples are also shown. Furthermore, these active learning methods are extended to the *pool-based scenarios*, where a set of input-only samples is provided in advance and users want to specify good input-only samples to gather output values.

In chapter 9, the problem of *active learning with model selection* is addressed. As explained in the previous chapters, model selection and active learning are two important challenges for successful learning. A natural desire is to perform model selection and active learning at the same time, that is, we want to choose the best model and the best training input points. However, this is actually a chicken-and-egg problem since training input samples should have been fixed for performing model selection and models should have been fixed for performing active learning. In this chapter, several compromise approaches, such as the sequential approach, the batch approach, and the ensemble approach, are discussed. Then, through numerical examples, limitations of the sequential and batch approaches are pointed out, and the usefulness of the *ensemble active learning* approach is demonstrated.

In chapter 10, applications of active learning techniques to real-world problems are shown. This chapter includes efficient exploration for autonomous robot control and efficient sensor design in semiconductor wafer alignment.