1 Introduction

Machine learning can be broadly defined as computational methods using experience to improve performance or to make accurate predictions. Here, *experience* refers to the past information available to the learner, which typically takes the form of electronic data collected and made available for analysis. This data could be in the form of digitized human-labeled training sets, or other types of information obtained via interaction with the environment. In all cases, its quality and size are crucial to the success of the predictions made by the learner.

Machine learning consists of designing efficient and accurate prediction *algorithms*. As in other areas of computer science, some critical measures of the quality of these algorithms are their time and space complexity. But, in machine learning, we will need additionally a notion of *sample complexity* to evaluate the sample size required for the algorithm to learn a family of concepts. More generally, theoretical learning guarantees for an algorithm depend on the complexity of the concept classes considered and the size of the training sample.

Since the success of a learning algorithm depends on the data used, machine learning is inherently related to data analysis and statistics. More generally, learning techniques are data-driven methods combining fundamental concepts in computer science with ideas from statistics, probability and optimization.

1.1 Applications and problems

Learning algorithms have been successfully deployed in a variety of applications, including

- Text or document classification, e.g., spam detection;
- Natural language processing, e.g., morphological analysis, part-of-speech tagging, statistical parsing, named-entity recognition;
- Speech recognition, speech synthesis, speaker verification;
- Optical character recognition (OCR);
- Computational biology applications, e.g., protein function or structured predic-

tion;

- Computer vision tasks, e.g., image recognition, face detection;
- Fraud detection (credit card, telephone) and network intrusion;
- Games, e.g., chess, backgammon;
- Unassisted vehicle control (robots, navigation);
- Medical diagnosis;
- Recommendation systems, search engines, information extraction systems.

This list is by no means comprehensive, and learning algorithms are applied to new applications every day. Moreover, such applications correspond to a wide variety of learning problems. Some major classes of learning problems are:

- *Classification*: Assign a category to each item. For example, document classification may assign items with categories such as *politics, business, sports*, or *weather* while image classification may assign items with categories such as *landscape, portrait*, or *animal*. The number of categories in such tasks is often relatively small, but can be large in some difficult tasks and even unbounded as in OCR, text classification, or speech recognition.
- *Regression*: Predict a real value for each item. Examples of regression include prediction of stock values or variations of economic variables. In this problem, the penalty for an incorrect prediction depends on the magnitude of the difference between the true and predicted values, in contrast with the classification problem, where there is typically no notion of closeness between various categories.
- *Ranking*: Order items according to some criterion. Web search, e.g., returning web pages relevant to a search query, is the canonical ranking example. Many other similar ranking problems arise in the context of the design of information extraction or natural language processing systems.
- *Clustering*: Partition items into homogeneous regions. Clustering is often performed to analyze very large data sets. For example, in the context of social network analysis, clustering algorithms attempt to identify "communities" within large groups of people.
- *Dimensionality reduction* or *manifold learning*: Transform an initial representation of items into a lower-dimensional representation of these items while preserving some properties of the initial representation. A common example involves preprocessing digital images in computer vision tasks.

The main practical objectives of machine learning consist of generating accurate predictions for unseen items and of designing efficient and robust algorithms to produce these predictions, even for large-scale problems. To do so, a number of algorithmic and theoretical questions arise. Some fundamental questions include:

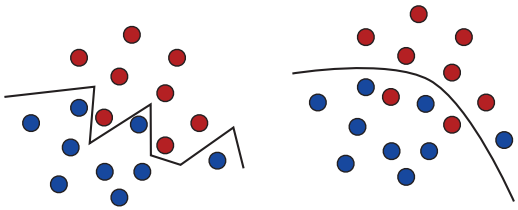

Figure 1.1 The zig-zag line on the left panel is consistent over the blue and red training sample, but it is a complex separation surface that is not likely to generalize well to unseen data. In contrast, the decision surface on the right panel is simpler and might generalize better in spite of its misclassification of a few points of the training sample.

Which concept families can actually be learned, and under what conditions? How well can these concepts be learned computationally?

1.2 Definitions and terminology

We will use the canonical problem of spam detection as a running example to illustrate some basic definitions and to describe the use and evaluation of machine learning algorithms in practice. Spam detection is the problem of learning to automatically classify email messages as either SPAM or non-SPAM.

- *Examples*: Items or instances of data used for learning or evaluation. In our spam problem, these examples correspond to the collection of email messages we will use for learning and testing.
- *Features*: The set of attributes, often represented as a vector, associated to an example. In the case of email messages, some relevant features may include the length of the message, the name of the sender, various characteristics of the header, the presence of certain keywords in the body of the message, and so on.
- *Labels*: Values or categories assigned to examples. In classification problems, examples are assigned specific categories, for instance, the SPAM and non-SPAM categories in our binary classification problem. In regression, items are assigned real-valued labels.
- *Training sample*: Examples used to train a learning algorithm. In our spam problem, the training sample consists of a set of email examples along with their associated labels. The training sample varies for different learning scenarios, as described in section 1.4.
- *Validation sample*: Examples used to tune the parameters of a learning algorithm

when working with labeled data. Learning algorithms typically have one or more free parameters, and the validation sample is used to select appropriate values for these model parameters.

- *Test sample*: Examples used to evaluate the performance of a learning algorithm. The test sample is separate from the training and validation data and is not made available in the learning stage. In the spam problem, the test sample consists of a collection of email examples for which the learning algorithm must predict labels based on features. These predictions are then compared with the labels of the test sample to measure the performance of the algorithm.

- *Loss function*: A function that measures the difference, or loss, between a predicted label and a true label. Denoting the set of all labels as $\mathcal{Y}$ and the set of possible predictions as $\mathcal{Y}'$, a loss function L is a mapping $L\colon \mathcal{Y} \times \mathcal{Y}' \to \mathbb{R}_+$. In most cases, $\mathcal{Y}' = \mathcal{Y}$ and the loss function is bounded, but these conditions do not always hold. Common examples of loss functions include the zero-one (or misclassification) loss defined over $\{-1, +1\} \times \{-1, +1\}$ by $L(y, y') = 1_{y' \neq y}$ and the squared loss defined over $I \times I$ by $L(y, y') = (y' - y)^2$, where $I \subseteq \mathbb{R}$ is typically a bounded interval.

- *Hypothesis set*: A set of functions mapping features (feature vectors) to the set of labels $\mathcal{Y}$. In our example, these may be a set of functions mapping email features to $\mathcal{Y} = \{\text{SPAM}, \text{non-SPAM}\}$. More generally, hypotheses may be functions mapping features to a different set $\mathcal{Y}'$. They could be linear functions mapping email feature vectors to real numbers interpreted as *scores* ($\mathcal{Y}' = \mathbb{R}$), with higher score values more indicative of SPAM than lower ones.

We now define the learning stages of our spam problem. We start with a given collection of labeled examples. We first randomly partition the data into a training sample, a validation sample, and a test sample. The size of each of these samples depends on a number of different considerations. For example, the amount of data reserved for validation depends on the number of free parameters of the algorithm. Also, when the labeled sample is relatively small, the amount of training data is often chosen to be larger than that of test data since the learning performance directly depends on the training sample.

Next, we associate relevant features to the examples. This is a critical step in the design of machine learning solutions. Useful features can effectively guide the learning algorithm, while poor or uninformative ones can be misleading. Although it is critical, to a large extent, the choice of the features is left to the user. This choice reflects the user's *prior knowledge* about the learning task which in practice can have a dramatic effect on the performance results.

Now, we use the features selected to train our learning algorithm by fixing different values of its free parameters. For each value of these parameters, the algorithm

selects a different hypothesis out of the hypothesis set. We choose among them the hypothesis resulting in the best performance on the validation sample. Finally, using that hypothesis, we predict the labels of the examples in the test sample. The performance of the algorithm is evaluated by using the loss function associated to the task, e.g., the zero-one loss in our spam detection task, to compare the predicted and true labels.

Thus, the performance of an algorithm is of course evaluated based on its test error and not its error on the training sample. A learning algorithm may be *consistent*, that is it may commit no error on the examples of the training data, and yet have a poor performance on the test data. This occurs for consistent learners defined by very complex decision surfaces, as illustrated in figure 1.1, which tend to memorize a relatively small training sample instead of seeking to generalize well. This highlights the key distinction between memorization and generalization, which is the fundamental property sought for an accurate learning algorithm. Theoretical guarantees for consistent learners will be discussed with great detail in chapter 2.

1.3 Cross-validation

In practice, the amount of labeled data available is often too small to set aside a validation sample since that would leave an insufficient amount of training data. Instead, a widely adopted method known as *n-fold cross-validation* is used to exploit the labeled data both for *model selection* (selection of the free parameters of the algorithm) and for training.

Let $\boldsymbol{\theta}$ denote the vector of free parameters of the algorithm. For a fixed value of $\boldsymbol{\theta}$, the method consists of first randomly partitioning a given sample S of m labeled examples into n subsamples, or folds. The ith fold is thus a labeled sample $((x_{i1}, y_{i1}), \ldots, (x_{im_i}, y_{im_i}))$ of size m_i. Then, for any $i \in [1, n]$, the learning algorithm is trained on all but the ith fold to generate a hypothesis h_i, and the performance of h_i is tested on the ith fold, as illustrated in figure 1.2a. The parameter value $\boldsymbol{\theta}$ is evaluated based on the average error of the hypotheses h_i, which is called the *cross-validation error*. This quantity is denoted by $\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta})$ and defined by

$$\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\frac{1}{m_i} \sum_{j=1}^{m_i} L(h_i(x_{ij}), y_{ij})}_{\text{error of } h_i \text{ on the } i\text{th fold}} .$$

The folds are generally chosen to have equal size, that is $m_i = m/n$ for all $i \in [1, n]$. How should n be chosen? The appropriate choice is subject to a trade-off and the topic of much learning theory research that we cannot address in this introductory

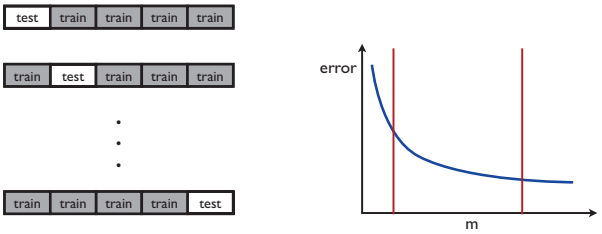

Figure 1.2 n-fold cross validation. (a) Illustration of the partitioning of the training data into 5 folds. (b) Typical plot of a classifier's prediction error as a function of the size of the training sample: the error decreases as a function of the number of training points.

chapter. For a large n, each training sample used in n-fold cross-validation has size $m - m/n = m(1-1/n)$ (illustrated by the right vertical red line in figure 1.2b), which is close to m, the size of the full sample, but the training samples are quite similar. Thus, the method tends to have a small bias but a large variance. In contrast, smaller values of n lead to more diverse training samples but their size (shown by the left vertical red line in figure 1.2b) is significantly less than m, thus the method tends to have a smaller variance but a larger bias.

In machine learning applications, n is typically chosen to be 5 or 10. n-fold cross validation is used as follows in model selection. The full labeled data is first split into a training and a test sample. The training sample of size m is then used to compute the n-fold cross-validation error $\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta})$ for a small number of possible values of $\boldsymbol{\theta}$. $\boldsymbol{\theta}$ is next set to the value $\boldsymbol{\theta}_0$ for which $\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta})$ is smallest and the algorithm is trained with the parameter setting $\boldsymbol{\theta}_0$ over the full training sample of size m. Its performance is evaluated on the test sample as already described in the previous section.

The special case of n-fold cross validation where $n = m$ is called *leave-one-out cross-validation*, since at each iteration exactly one instance is left out of the training sample. As shown in chapter 4, the average leave-one-out error is an approximately unbiased estimate of the average error of an algorithm and can be used to derive simple guarantees for some algorithms. In general, the leave-one-out error is very costly to compute, since it requires training n times on samples of size $m - 1$, but for some algorithms it admits a very efficient computation (see exercise 10.9).

In addition to model selection, n-fold cross validation is also commonly used for performance evaluation. In that case, for a fixed parameter setting $\boldsymbol{\theta}$, the full labeled sample is divided into n random folds with no distinction between training and test samples. The performance reported is the n-fold cross-validation on the full sample as well as the standard deviation of the errors measured on each fold.

1.4 Learning scenarios

We next briefly describe common machine learning scenarios. These scenarios differ in the types of training data available to the learner, the order and method by which training data is received and the test data used to evaluate the learning algorithm.

- *Supervised learning*: The learner receives a set of labeled examples as training data and makes predictions for all unseen points. This is the most common scenario associated with classification, regression, and ranking problems. The spam detection problem discussed in the previous section is an instance of supervised learning.
- *Unsupervised learning*: The learner exclusively receives unlabeled training data, and makes predictions for all unseen points. Since in general no labeled example is available in that setting, it can be difficult to quantitatively evaluate the performance of a learner. Clustering and dimensionality reduction are example of unsupervised learning problems.
- *Semi-supervised learning*: The learner receives a training sample consisting of both labeled and unlabeled data, and makes predictions for all unseen points. Semi-supervised learning is common in settings where unlabeled data is easily accessible but labels are expensive to obtain. Various types of problems arising in applications, including classification, regression, or ranking tasks, can be framed as instances of semi-supervised learning. The hope is that the distribution of unlabeled data accessible to the learner can help him achieve a better performance than in the supervised setting. The analysis of the conditions under which this can indeed be realized is the topic of much modern theoretical and applied machine learning research.
- *Transductive inference*: As in the semi-supervised scenario, the learner receives a labeled training sample along with a set of unlabeled test points. However, the objective of transductive inference is to predict labels only for these particular test points. Transductive inference appears to be an easier task and matches the scenario encountered in a variety of modern applications. However, as in the semi-supervised setting, the assumptions under which a better performance can be achieved in this setting are research questions that have not been fully resolved.
- *On-line learning*: In contrast with the previous scenarios, the online scenario involves multiple rounds and training and testing phases are intermixed. At each round, the learner receives an unlabeled training point, makes a prediction, receives the true label, and incurs a loss. The objective in the on-line setting is to minimize the cumulative loss over all rounds. Unlike the previous settings just discussed, no distributional assumption is made in on-line learning. In fact, instances and their labels may be chosen adversarially within this scenario.

- *Reinforcement learning*: The training and testing phases are also intermixed in reinforcement learning. To collect information, the learner actively interacts with the environment and in some cases affects the environment, and receives an immediate reward for each action. The object of the learner is to maximize his reward over a course of actions and iterations with the environment. However, no long-term reward feedback is provided by the environment, and the learner is faced with the *exploration versus exploitation* dilemma, since he must choose between exploring unknown actions to gain more information versus exploiting the information already collected.
- *Active learning*: The learner adaptively or interactively collects training examples, typically by querying an oracle to request labels for new points. The goal in active learning is to achieve a performance comparable to the standard supervised learning scenario, but with fewer labeled examples. Active learning is often used in applications where labels are expensive to obtain, for example computational biology applications.

In practice, many other intermediate and somewhat more complex learning scenarios may be encountered.

1.5 Outline

This book presents several fundamental and mathematically well-studied algorithms. It discusses in depth their theoretical foundations as well as their practical applications. The topics covered include:

- Probably approximately correct (PAC) learning framework; learning guarantees for finite hypothesis sets;
- Learning guarantees for infinite hypothesis sets, Rademacher complexity, VC-dimension;
- Support vector machines (SVMs), margin theory;
- Kernel methods, positive definite symmetric kernels, representer theorem, rational kernels;
- Boosting, analysis of empirical error, generalization error, margin bounds;
- Online learning, mistake bounds, the weighted majority algorithm, the exponential weighted average algorithm, the Perceptron and Winnow algorithms;
- Multi-class classification, multi-class SVMs, multi-class boosting, one-versus-all, one-versus-one, error-correction methods;
- Ranking, ranking with SVMs, RankBoost, bipartite ranking, preference-based

ranking;

- Regression, linear regression, kernel ridge regression, support vector regression, Lasso;
- Stability-based analysis, applications to classification and regression;
- Dimensionality reduction, principal component analysis (PCA), kernel PCA, Johnson-Lindenstrauss lemma;
- Learning automata and languages;
- Reinforcement learning, Markov decision processes, planning and learning problems.

The analyses in this book are self-contained, with relevant mathematical concepts related to linear algebra, convex optimization, probability and statistics included in the appendix.