Statistical Computational Learning

Antoine Cornuejols, Frédéric Koriche and Richard Nock

Abstract Statistical computational learning is the branch of Machine Learning that defines and analyzes the performance of learning algorithms using two metrics: sample complexity and runtime complexity. This chapter is a short introduction to this important area of research, geared toward the reader interested in developing learning algorithms for AI models. We first provide the formal background about statistical learning problems, captured by three basic ingredients: tasks, models and loss functions. We next examine the PAC learning framework and its generalizations, used to capture the concepts of statistical learnability and computational (or efficient) learnability. Based on this framework, the conditions of statistical learnability are investigated through the properties of uniform convergence and algorithmic stability. We also survey several theoretical results and algorithms in the topics of concept learning and convex learning, which take a central place in statistical computational learning. We then conclude this survey with some trends and open questions in learning AI models, by mainly focusing on sparse models, probabilistic models, preference models and deep neural models.

1 Introduction

The cognitive ability of *learning* has long fascinated philosophers, psychologists, statisticians, computer scientists and, of course, the parents of young children. In Computer Science, Turing already speculated in Turing (1950) that learning would be used to build machines that think. Since then, the field of Machine Learn-

A. Cornuejols (✉)
AgroParisTech, Paris, France
e-mail: antoine.cornuejols@agroparistech.fr

F. Koriche
CRIL-CNRS and Université d'Artois, Lens, France
e-mail: frederic.koriche@cril.univ-artois.fr

R. Nock
NICTA, ANU College of Engineering and Computer Science, Canberra, Australia
e-mail: richard.nock@nicta.com.au

P. Marquis et al. (eds.), *A Guided Tour of Artificial Intelligence Research*,
https://doi.org/10.1007/978-3-030-06164-7_11

ing has flourished with the development of various learning frameworks, theories, algorithms, and practical applications. In fact, we are nowadays surrounded by learning-based computer technologies: our smartphones learn to recognize voice commands, our digital cameras learn to identify faces, antispam softwares learn to filter our email messages, and recommender systems learn our preferences about daily consumer objects. Learning algorithms are also widely used in scientific applications such as astronomy, bioinformatics, medicine, economy and robotics.

Broadly speaking, the main concern of Machine Learning is to study *how computer algorithms can improve automatically through experience*. Virtually all machine learning activities involve a *task* we wish to solve, a set of candidate prediction *models* for solving this task, and an *objective function* for measuring the performance of a model at solving the task. In this setting, the term "experience" refers to the information provided to the learning algorithm for assessing the quality of candidate models, and ultimately, choosing the right one.

To illustrate these aspects with a concrete example, consider the common task of classifying incoming email messages as either SPAM or non-SPAM. As electronic messages usually contain a text in natural language, possibly coupled with graphical elements and URL links, the problem of recognizing whether an incoming email is a spam, or not, is far from easy. So, in order to facilitate the learning process, each electronic message is associated with a set (or vector) of *features*, which capture informative properties of the message, such as its size, its text-to-image ratio, the presence of some domain names in the header, or the occurrence of certain regular expressions in the content. Based on this feature representation, the task of spam filtering is essentially to map email messages, described by their features, to the set of labels {SPAM, non−SPAM}. Any such mapping is called *hypothesis* or *model*, and the set of candidate models available to the learner is called the *hypothesis class*. Since spam filtering is a binary classification task, various models can be used, such as decision trees, separating hyperplanes, or Bayesian classifiers. Finally, we need to assess the performance of the chosen model at filtering incoming messages. Here, a natural objective function is the "zero-one" loss function, which simply counts the number of mistakes made by the model in labeling messages.

Based on the three ingredients, tasks, models and objective functions, the goal of a learning algorithm is essentially to find, in its hypothesis class, a model that optimizes some given objective function for the task at hand. To achieve this goal, the learner has usually access to a *training set*, that is, a sequence of data instances upon which the quality of candidate models can be measured. In spam filtering, the training set is a pool of email messages, each described by its features, and labeled by SPAM or by non-SPAM. Importantly, this training set captures only a small fragment of emails we are expected to receive. So, the learning problem is not to find a model that makes few mistakes on the training set, but to extrapolate from observed instances a model that accurately classifies *new*, incoming messages. In a nutshell, the key characteristic of learning algorithms lies in their ability to *generalize*, that is, to predict from observed data, the outcome of future data.

This chapter focuses on *statistical computational learning*, the branch of Machine Learning that lies at the intersection of statistical modeling and computational learning theory. In this setting, the generalization ability of learning algorithms is defined and analyzed through two key metrics: sample complexity and runtime complexity. Because statistical computational learning has long been recognized as the mainstream theoretical framework for analyzing the performance of learning algorithms, a detailed survey of this research field and its applications would require a whole book! In fact, there are already excellent printed works on statistical and computational learning, targeted to various audiences (Natarajan 1991; Kearns and Vazirani 1994; Anthony and Biggs 1997; Vapnik 1998; Engel and Broeck 2001; Hastie et al. 2009; DasGupta 2011; Kulkarni and Harman 2011; Webb and Copsey 2011; Devroye et al. 2013; James et al. 2013; Vapnik 2013; Sugiyama 2015). Furthermore, many introductory books in Machine Learning are devoting a significant part to statistical and/or computational learning theory (Mitchell 1997; Bishop 2006; Alpaydin 2009; Flach 2012; Mohri et al. 2012; Murphy 2012; Shalev-Shwartz and Ben-David 2014; Theodoridis 2015). So, this chapter is an elementary introduction to statistical computational learning, geared toward readers who have familiar with AI models, such as logical representations, geometric descriptions, and graphical models.

We introduce in Sect. 2 the formal background about statistical learning problems. The central notions of *statistical learnability* and *computational learnability* are defined in Sect. 3. The related optimization principles and conditions of learnability are examined in Sect. 4. With these theoretical notions in hand, the important topics of *concept learning* and *convex learning* are surveyed in Sects. 5 and 6, respectively. Finally, we conclude this chapter by discussing about some trends and open questions in statistical learning with sparse models, probabilistic models, preference models, and neural networks.

Notation. For the sake of clarity we shall use as much as possible the standard notation in Machine Learning. Scalars and vectors are denoted by lowercase letters. Sets, matrices, sequences, and distributions are denoted by uppercase letters. We use boldface letters for vectors and matrices. For an integer n, we use $[n]$ as an abbreviation of $\{1, \ldots, n\}$. Given a sequence S of m vectors $(\boldsymbol{x}_1, \ldots, \mathbf{x}_m)$, we use $x_{i,j}$ to denote the jth element of $\boldsymbol{x}_i$. The inner product of two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ is denoted $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$, and for any $p \in [1, \infty]$, the ℓ_p norm of $\boldsymbol{x}$ is denoted $\|\boldsymbol{x}\|_p$. In other words,

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \text{ and in particular } \|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i| \text{ and } \|\boldsymbol{x}\|_\infty = \max_{i \in [n]} |x_i|$$

We omit the subscript from the standard ℓ_2 (Euclidean) norm when it is clear from the context. The number of nonzero coordinates in $\boldsymbol{x}$, often called ℓ_0 *pseudo-norm* of $\boldsymbol{x}$, is denoted $\|\boldsymbol{x}\|_0$. For a set of scalars $X \subseteq \mathbb{R}$, the greatest lower bound of X and the least upper bound of X are denoted $\inf X$ and $\sup X$, respectively. Finally, we shall assume throughout this chapter that the sample space of any probability distribution is equipped with an implicit σ-algebra upon which the distribution is defined. Given a probability distribution $\mathscr{D}$ over a sample space $\mathscr{X} \subseteq \mathbb{R}^n$, we use $\boldsymbol{x} \sim \mathscr{X}$ to indicate that $\boldsymbol{x}$ is sampled according to $\mathscr{D}$. Probabilities and expectations over $\mathscr{D}$ are denoted $\mathbb{P}$ and $\mathbb{E}$, respectively.

2 Statistical Learning Problems

In order to provide a clear definition of "statistical computational learning", we need to capture in a formal way the three aforementioned ingredients: *tasks*, *models*, and *objective functions*. We start this section by discussing about these notions, and then describe the statistical learning framework upon which the rest of chapter is built.

2.1 Tasks

As Machine Learning can be considered as a data-driven approach to problem solving, the notion of "task" is described through its data instances. Specifically, an *instance space* is a (possibly infinite) subset $\mathscr{Z}$ of $\mathbb{R}^d$. Each coordinate $i \in [d]$ represents a distinct feature, and each instance $\boldsymbol{z} \in \mathscr{Z}$ is a vector of d feature values.

Learning algorithms can solve a wide variety of tasks and, for this reason, it may be useful to separate them into categories. A first separation, commonly advocated in the Machine Learning literature, is to distinguish *supervised* learning tasks from *unsupervised* ones.

Basically, supervised learning tasks capture applications for which we need to predict the dependence of an outcome $\boldsymbol{y} \in \mathscr{Y}$ on an observed information $\boldsymbol{x} \in \mathscr{X}$. Here, $\mathscr{Z}$ is the Cartesian product $\mathscr{X} \times \mathscr{Y}$ of a *domain set* $\mathscr{X}$ and a *target set* $\mathscr{Y}$. Pairs of the form $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ are often referred to as *labeled instances* or *examples*. The dimensions of $\mathscr{X}$ and $\mathscr{Y}$ are denoted n and p, respectively. A supervised learning task is *uni-dimensional* if $p = 1$, and *multi-dimensional* if $p > 1$. Some of the most common supervised learning tasks include the following:

- *Classification:* $\mathscr{Y}$ is a finite subset of $\mathbb{Z}$, encoding a collection of labels. The spam filtering task mentioned in the introduction of this chapter is an example of *binary* classification problem, where $\mathscr{Y}$ is usually defined by $\{0, 1\}$ or $\{-1, +1\}$. Classification problems with more than two labels are often referred to as *multi-class* or *multi-nominal* classification tasks.
- *Regression:* $\mathscr{Y}$ is a (typically bounded) subset of $\mathbb{R}$, capturing the domain of some real-valued variable. A common example of regression task is to estimate the revenue of a company, using historical accounting data.
- *Multi-label classification:* $\mathscr{Y}$ is a subset of $\{0, 1\}^p$ or $\{-1, +1\}^p$ for $p > 1$. Here, the learner as access to p distinct labels, and the goal is to map each input vector to a *subset* of these labels. A common example of multi-label classification in document analysis is to "tag" incoming news according to their most relevant topics (e.g. sports, entertainment, politics, science).

- *Multi-variate regression:* By analogy with multi-label classification, $\mathscr{Y}$ is a subset of $\mathbb{R}^p$ for $p > 1$. A well-studied example in ecological modeling is to simultaneously predict multiple target variables describing the condition or quality or plant species.
- *Structured prediction:* This setting covers multi-dimensional prediction tasks in which target variables are organized into some *structure*, such as a permutation, a tree, or a bipartite graph. One example is *parsing*, the task of mapping a natural language sentence into a tree that predicts its grammatical structure. Another example is *label ranking*, the task of mapping a feature vector (e.g. a user profile) into a permutation of items (e.g. movies).

In contrast with supervised tasks, there is *no* target set $\mathscr{Y}$ in unsupervised tasks. Here, $\mathscr{Z}$ is a set $\mathscr{X}$ of unlabeled instances. The overall goal of unsupervised learning is to extract from observed data some regularities or patterns which are likely to be found in future data. Two of the most popular unsupervised learning tasks are:

- *k-Means clustering*: The goal is to partition the instance space $\mathscr{X}$ into k clusters, each identified by a centroid $\boldsymbol{c}$ in $\mathscr{Z}$. Any incoming instance $\boldsymbol{x}$ is mapped to the centroid $\boldsymbol{c}$ that minimizes the squared distance $\|\boldsymbol{x} - \boldsymbol{c}\|^2$.
- *Density estimation*: Here, the task is to find a probability distribution over $\mathscr{X}$ that estimates the likeliness of incoming instances. This distribution can be viewed as a maximum likelihood estimator of the data instances supplied to the learner.

2.2 Models

In order to solve a given task, the learner has access to a set of candidate hypotheses, called the *hypothesis class*, and denoted $\mathscr{H}$. From a general viewpoint, any hypothesis in $\mathscr{H}$ can be viewed as a mapping $h : \mathscr{X} \to \mathscr{Y}^\dagger$, where $\mathscr{X}$ is the set of input observations, and $\mathscr{Y}^\dagger$ is a set of decisions. By analogy with learning tasks, hypotheses can be separated into *discriminative* models and *descriptive* models. Basically, discriminative models are dedicated to supervised learning tasks. Here, the decision set $\mathscr{Y}^\dagger$ coincides with the target set $\mathscr{Y}$, and hence, any class $\mathscr{H}$ of discriminative models is a subset of the function space $\mathscr{Y}^{\mathscr{X}}$. By contrast, descriptive models are used to explain observations by extracting regularities or patterns. For those models, the choice of $\mathscr{Y}^\dagger$ depends on how observations are explained. An important subclass of descriptive models is the family of *generative* models, where $\mathscr{Y}^\dagger = [0, 1]$, and $\mathscr{H}$ is a set of probability distributions over $\mathscr{X}$. While generative models are mainly devoted to unsupervised learning tasks, they may be applied to supervised learning problems by first extracting from examples a probabilistic model that estimates the underlying distribution, and then using this model for solving various tasks.

Some of the most common families of hypothesis classes which have been examined in Machine Learning include:

- *Logical models:* $\mathscr{H}$ is typically a set of functions of the form $h : \{0, 1\}^n \to \{0, 1\}$. In other words, the domain of a logical model is a set of Boolean features, and its range is a Boolean variable. Simple logical models are constructed using a single logical operator; they include *monomials* (conjunctions of literals), *clauses* (disjunctions of literals), and *XOR clauses* (exclusive-or of literals). More complex functions are built using at least two logical operators. They include, among others, *DNF formulas* (disjunctions of monomials), *decision trees* (disjunctions of monomials organized into a tree), and *decision lists* (sets of monomials organized into a preference list). The main learning task considered for logical models is binary classification; this problem had long been considered as a central topic in computational learning theory (Natarajan 1991; Anthony 2010; Kearns et al. 1994a; Kearns and Vazirani 1994). Besides Boolean functions, logical models investigated in Machine Learning include *relational models*, defined over structured domain spaces (Getoor and Taskar 2007; De Raedt 2008).

- *Geometric models:* $\mathscr{H}$ is a set of geometric objects or functions over $\mathscr{X} \subseteq \mathbb{R}^n$. Arguably, the simplest hypothesis class in the family of geometric models is the class of *separating hyperplanes*, also known as *linear threshold functions*, which has been studied since the very start of Machine Learning (Rosenblatt 1958). Here, each hypothesis $h : \mathbb{R}^n \to \{-1, +1\}$ is represented by a pair $(\boldsymbol{w}, b)$, where $\boldsymbol{w} \in \mathbb{R}^n$ is a weight vector, and $b \in \mathbb{R}$ is a threshold value. The label assigned to any input object $\boldsymbol{x} \in \mathbb{R}^n$ is given by

$$h(\boldsymbol{x}) = \begin{cases} +1 & \text{if } \langle \boldsymbol{w}, \boldsymbol{x} \rangle > b \\ -1 & \text{otherwise.} \end{cases} \tag{1}$$

For *zero-threshold* or *homogeneous* linear functions, h is simply described by its weight vector $\boldsymbol{w}$, and defined by $h(\boldsymbol{x}) = \operatorname{sign} \langle \boldsymbol{w}, \boldsymbol{x} \rangle$. More complex geometric objects may be defined using a weight vector $\boldsymbol{w} \in \mathbb{R}^p$, together with a *feature expansion* mapping: $\boldsymbol{\phi} : \mathscr{X} \to \mathscr{X}^\dagger$, where $\mathscr{X}^\dagger$ is an Euclidean or Hilbert space. In this general setting,

$$h(\boldsymbol{x}) = \begin{cases} +1 & \text{if } \langle \boldsymbol{w}, \phi(\boldsymbol{x}) \rangle > b \\ -1 & \text{otherwise.} \end{cases} \tag{2}$$

Linear functions and their feature expansions can be extended, in a natural way, to regression tasks, multi-nominal classification tasks, and even multi-dimensional prediction tasks. Besides hyperplanes, *manifolds* and *distances* take also an important place in geometric learning. Namely, manifolds are used for extracting a low-dimensional structure from a high-dimensional domain (Ma and Fu 2011), and distance functions are commonly used in classification, regression, and clustering (Aggarwal and Reddy 2013).

- *Probabilistic models:* $\mathscr{H}$ is a set of probability distributions over an instance space $\mathscr{X} \subseteq \mathbb{R}^d$. Of particular importance are probabilistic *graphical* models, which encode high-dimensional probability distributions in a compact and intuitive way (Koller and Friedman 2009; Murphy 2012). Here, each hypothesis is represented by a pair $(G, \boldsymbol{\theta})$, where G is a graph over $[d]$ nodes, and $\boldsymbol{\theta}$ is a vector of parameters which together determine a probability distribution over $\mathscr{X}$. In *directed* graphical models, also known as *Bayesian networks* (Pearl 1988; Darwiche 2009), G is a directed acyclic graph, and $\boldsymbol{\theta}$ is a set of conditional probability tables associated with the nodes of G. In *undirected* graphical models (Wainwright and Jordan 2008), such as *factor graphs* and *Markov networks*, G is an undirected graph and $\boldsymbol{\theta}$ is a vector of energy functions defined on the edges (for factor graphs) or the cliques (for Markov networks) of the graph. Probabilistic graphical models can be applied to a wide variety of learning tasks, including density estimation and structured prediction.

- *Preference models:* $\mathscr{H}$ is a set of functions from $\mathscr{X}$ to $\mathscr{Y}$, where $\mathscr{X}$ is a set of objects, possibly coupled with user profiles, and $\mathscr{Y}$ is a partial or total ordering over some reference set $\mathscr{I}$. In preference learning (Fürnkranz and Hüllermeier 2010), the family of models may be organized into different subclasses, depending on the type of reference set $\mathscr{I}$, and the type of preference relation $\mathscr{Y}$. In *object ranking* (Cohen et al. 1999), $\mathscr{I}$ is a set of objects in $\mathscr{X}$, while in *label ranking* (Vembu and Gärtner 2010), $\mathscr{I}$ is a set of labels associated with objects in $\mathscr{X}$. Orthogonally, *total rankings* are permutations over $\mathscr{I}$, while *partial rankings* are pre-orderings on $\mathscr{I}$. For example, in the task of *top-k object ranking* commonly used in information retrieval, the goal is to find a total ordering over the k best objects in $\mathscr{X}$, while others objects are considered indifferent. Similarly, the task of *bipartite ranking* is to separate objects in $\mathscr{X}$ in two categories: the most preferred objects, and the less preferred ones (Clémençon and Vayatis 2007). Common preference models advocated in the Machine Learning literature include the *Placket-Luce model* (Plackett 1975), the *Mallows model* (Mallows 1957), and their extensions (Fligner and Verducci 1986; Lebanon and Lafferty 2002; Meila and Chen 2010; Lu and Boutilier 2014; Zhao et al. 2016).
- *Neural models:* $\mathscr{H}$ is a class of artificial neural networks, inspired from the structure of neural networks in the brain. A *feedforward neural netwok* is defined by a labeled and weighted directed acyclic graph. Each node in the graph a simple model of neuron, labeled by an activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$. Common scalar functions include the sign function $\sigma(a) = \text{sign}(a)$, the threshold function given by (1), and the sigmoid function $\sigma(a) = {}^{1}/{}_{1+\exp(-a)}$. Each edge in the graph, linking the output of some neuron to the input of another neuron, is associated with a weight that reflects the strength of the signal joining both neurons. The input of a neuron is obtained by taking the weighted sum of the outputs of its incident neurons. It is often assumed that neurons are organized in *layers*. Namely, the set of nodes in the graph is partitioned into $d + 1$ subsets $\{V_0, V_1, \dots, V_d\}$, where V_0 is the input layer, V_d is the output layer, and $\{V_1, \dots, V_{d-1}\}$ are the *hidden* layers. The *depth* and *width* of the network are given by d and $\max_i |V_i|$, respectively. Based on this layer structure, the output of the layer V_t is given by:

$$\boldsymbol{x}_t = \boldsymbol{\sigma}\left(\boldsymbol{W}_t^\top \boldsymbol{x}_{t-1} + \mathbf{b}_t\right)$$

where $\boldsymbol{x}_{t-1}$ is the input of the tth layer, $\boldsymbol{\sigma}$ is the (possibly rectified) activation function for this layer, $\boldsymbol{W}_t$ is the weight matrix capturing weighted edges between the layers V_{t-1} and V_t, and $\mathbf{b}_t$ is a bias vector. The family of hypothesis classes of artificial neural networks is very expressive: notably, Boolean functions of polynomial circuit complexity can be represented by neural networks of polynomial size (Parberry 1994). For this reason, neural networks have been a subject of extensive research in statistical computational learning (Anthony and Barlett 1999; Anthony 2001; Du and Swamy 2013). *Deep networks*, characterized by more than two layers, have recently shown very impressive practical performance on a wide variety of learning tasks (Goodfellow et al. 2016).

2.3 Objective Functions

In Machine Learning, the connexion between tasks and models is established through *objective* or *loss* functions. Formally, a loss function is a map ℓ from $\mathscr{H} \times \mathscr{Z}$ to $\mathbb{R}$ that penalizes a model $h \in \mathscr{H}$ picked by the learner when it observes the instance $z \in \mathscr{Z}$. In other words, $\ell(h, z)$ is the cost incurred by h on z. Some of the most common loss functions include the following:

- *Zero-one loss:* Applied to binary classification, this function measures whether a binary hypothesis is misclassifying a labeled instance. Formally, $\ell(h, (\boldsymbol{x}, y)) = 1$ if $h(\boldsymbol{x}) \neq y$, and $\ell(h, (\boldsymbol{x}, y)) = 0$ otherwise.
- *Quadratic loss:* This function, commonly used in regression tasks, measures the squared distance between a predicted value and the target value. Namely, $\ell(h, (\boldsymbol{x}, y)) = (h(\boldsymbol{x}) - y)^2$.
- *Hinge loss:* This function is a convex surrogate of the zero-one loss in linear classification. For a zero-threshold separating hyperplane h, represented by its weight vector $\boldsymbol{w}$, the hinge loss of h on some example $(\boldsymbol{x}, y)$ is given by

$$\ell(h, (\boldsymbol{x}, y)) = \max\{0, 1 - y \langle \boldsymbol{w}, \boldsymbol{x} \rangle\}$$

- *Log-loss:* Used in density estimation, this function measures the negative log-likelihood of a probabilistic model $h : \mathscr{X} \to [0, 1]$ given an incoming instance $\boldsymbol{x}$. Formally, $\ell(h, \boldsymbol{x}) = -\ln[h(\boldsymbol{x})]$.
- *Conditional log-loss:* As a direct extension of the log-loss, this function is often used in structured prediction. Given a conditional probabilistic model h that maps each input object $\boldsymbol{x}$ to a probability distribution $h(\cdot \mid \boldsymbol{x})$ over $\mathscr{Y}$, the conditional log-loss of h with respect to an example $(\boldsymbol{x}, \boldsymbol{y})$ is $\ell(h, (\boldsymbol{x}, \boldsymbol{y})) = -\ln[h(\boldsymbol{y} \mid \boldsymbol{x})]$.

2.4 The Framework

The three components - tasks, models, and objective functions - are common to many machine learning frameworks. The specificity of statistical learning lies in a fourth component that captures how data instances are generated. Here, it is assumed that instances are independently and identically distributed (i.i.d.) according to some probability distribution $\mathscr{D}$ over $\mathscr{Z}$. Importantly, $\mathscr{D}$ is an *arbitrary* but *hidden* distribution: the incoming data can be generated according to any possible distribution over the sample space $\mathscr{Z}$, and the learning algorithm has no prior information about this distribution. Instead, the learner has access to $\mathscr{D}$ through a procedure $\text{EX}(\mathscr{D})$, that runs in unit time, and on each call returns an instance $z \in \mathscr{Z}$ drawn randomly and independently according to $\mathscr{D}$. This procedure, referred to as *example oracle*, is used to generate a *training set*, that is, a sequence $S = (z_1, \dots, z_m)$ of instances which are i.i.d. according to $\mathscr{D}$.

Recall that in supervised learning, the instance space $\mathscr{Z}$ is the Cartesian product of a domain set $\mathscr{X}$ and a target set $\mathscr{Y}$. So, in this setting, $\mathscr{D}$ is a joint distribution over $\mathscr{X} \times \mathscr{Y}$. Equivalently, this distribution can be viewed as the conditional probability of observing the labeled object $(\boldsymbol{x}, \boldsymbol{y})$ given an unlabeled object $\boldsymbol{x}$. For instance, in the spam filtering task, $\mathscr{D}$ specifies the probability of encountering a spam message, given a feature description of this message. In unsupervised learning, the learner has only access to unlabeled observations, and its goal is essentially to predict the data-generation model $\mathscr{D}$ using a limited number of calls to $\text{EX}(\mathscr{D})$.

With these components in hand, we are now in position to describe the learning framework upon which the remaining sections are built. Formally, a *statistical learning problem* is defined as follows:

Given:

- A task described by its instance space $\mathscr{Z}$
- A hypothesis class $\mathscr{H}$ for $\mathscr{Z}$
- A loss function $\ell : \mathscr{H} \times \mathscr{Z} \to \mathbb{R}$
- A distribution $\mathscr{D}$ accessible through the example oracle $\text{EX}(\mathscr{D})$

Find a hypothesis $h \in \mathscr{H}$ that minimizes

$$L_{\mathscr{D}}(h) = \mathbb{E}_{\mathbf{z} \sim \mathscr{D}}[\ell(h, \mathbf{z})] \tag{3}$$

The objective function $L_{\mathscr{D}} : \mathscr{H} \to \mathbb{R}$ in (3) is called the *true risk*, or *risk* for short. It measures the expected loss of a hypothesis $h \in \mathscr{H}$ with respect to the probability distribution $\mathscr{D}$ over $\mathscr{Z}$. Since the learner has only access to a sample of data instances picked randomly according to $\mathscr{D}$, we define the *empirical risk* of a hypothesis h with respect to a training set $S = (z_1, \dots, z_m)$ as:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, z_i) \tag{4}$$

The main difficulty of statistical learning is to estimate the unknown true risk according to the known empirical risk. The intimate relation between $L_{\mathscr{D}}$ and L_S will be discussed in Sect. 4.

In practice, how do we analyze the performance of learning algorithms? There is no simple answer to this question, since the instance space $\mathscr{X}$ of most learning problems is immense, or infinite. In practice, we typically have a limited "dataset" for the task we wish to solve. If the dataset is already separated into a training sample S and a test sample S', then we just have to train our algorithm on S, and to measure the empirical risk of its output model on S'. Yet, if the dataset does not include a predefined test sample, we need to resort on a statistical validation technique for assessing the performance of the learner. The following *k-fold cross-validation* procedure is often applied: randomly partition the dataset S in k parts or "folds" $S_1, \ldots, S_k$, pick one fold S_j for testing, train the algorithm on the complementary set $S \backslash S_j$, and evaluate the resulting hypothesis h_j on the test fold S_j. This process is repeated k times, until each fold has been picked for testing once. The *cross-validation risk* of the k hypotheses $(h_1, \ldots, h_k)$ returned by the algorithm is given by

$$L_{\mathrm{CV}}(h_1, \ldots, h_k) = \frac{1}{k} \sum_{j=1}^{k} L_{S_j}(h_j)$$

3 Complexity Measures

As mentioned above, a statistical learning problem involves a task, described by its instance space $\mathscr{X}$, a hypothesis class $\mathscr{H}$ for $\mathscr{X}$, a loss function ℓ, and a hidden distribution $\mathscr{D}$ over $\mathscr{X}$ which is only accessible through an example oracle $\mathrm{EX}(\mathscr{D})$. The goal is to to find a model with good generalization performance, that is, a hypothesis $h \in \mathscr{H}$ for which the true risk $L_{\mathscr{D}}(h)$ is as small as possible. Based on this formulation, there are two main sources of complexity in the computational approach to statistical learning. The first, *sample complexity*, measures the inherent difficulty of generalizing from examples: it is the number of calls to $\mathrm{EX}(\mathscr{D})$ which are required to find a good hypothesis. The second, *runtime complexity*, measures the amount of computational steps required to find such a model. This section explores in more detail both sources of complexity which are related to the concept of *learnability*.

3.1 Sample Complexity

As indicated above, sample complexity is the amount of information learning requires. to find a "good" hypothesis. In order to capture this metric in a more rigorous way, we need a formal model of *learnability*, that explains the ability of algorithms to predict with respect to a hypothesis class, given access to training samples.

We begin with a conceptually simple notion of learnability, introduced by (Valiant 1984) and thoughtfully detailed in various textbooks about computational learning theory (Natarajan 1991; Kearns and Vazirani 1994; Anthony and Biggs 1997). In *Probably Approximately Correct* (PAC) learning, we are concerned with supervised learning tasks, where the instance space $\mathscr{Z}$ is the product of a domain set $\mathscr{X}$ and a target set $\mathscr{Y}$. Originally, the PAC learning framework was defined for binary classification tasks, but we can easily extend the framework to other discriminative tasks, using an appropriate loss function. The key assumption in PAC learning, often called *realizability condition*, is to consider that the hypothesis class $\mathscr{H}$ includes at least one model, say h^*, which correctly solves the task at hand. In other words, the outcome $\boldsymbol{y}$ of any input object $\boldsymbol{x}$ is given by $\boldsymbol{y} = h^*(\boldsymbol{x})$. The realizability assumption can be captured using a restricted example oracle $\textsc{ex}(h^*, \mathscr{D})$ which returns, on each call, a labeled example $(\boldsymbol{x}, h^*(\boldsymbol{x}))$, where $\boldsymbol{x}$ is drawn at random according to a hidden distribution $\mathscr{D}$ over $\mathscr{X}$.

A PAC learning algorithm takes as input a *confidence* parameter and a *accuracy* parameter, denoted δ and ε, respectively. These parameters are use to control two types failures which are inherent to learn from samples drawn at random according to an unknown distribution $\mathscr{D}$. The confidence parameter is necessary since there is always a chance that the training set picked by the learner is not representative of $\mathscr{D}$. For example, the learner might be very unlucky by picking a sample consisting of repeated draws of the same object in $\mathscr{X}$, despite the fact that the distribution is spread evenly over all the domain set $\mathscr{X}$. The accuracy parameter is also necessary since, even with a training set that is representative of $\mathscr{D}$, some objects in $\mathscr{X}$ may have a very low probability under $\mathscr{D}$, and hence, the learning algorithm will not see the target function's behavior on those objects. So, the best we can hope is that the likeliness of both types of failure can be made arbitrary small, at the cost of increasing the size of the training set.

Definition 1 (*PAC Learning*) Let $\mathscr{Z} = \mathscr{X} \times \mathscr{Y}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \to \mathbb{R}$ be a loss function. Then, $\mathscr{H}$ is PAC *learnable* with respect to ℓ if there exist an algorithm $\textsc{learn}$ with the following property: for any hypothesis $h^* \in \mathscr{H}$, any distribution $\mathscr{D}$ over $\mathscr{X}$, and any pair $(\delta, \varepsilon) \in (0, 1)^2$, if $\textsc{learn}$ is given inputs δ and ε, and access to $\textsc{ex}(h^*, \mathscr{D})$, then $\textsc{learn}$ returns a hypothesis $h \in \mathscr{H}$ that satisfies $L_{\mathscr{D}}(h) \leq \varepsilon$ with probability $1 - \delta$.

In essence, PAC learning is a *distribution-free* model of statistical learning: for any possible distribution $\mathscr{D}$ over the domain set $\mathscr{X}$, the algorithm $\textsc{learn}$ must be "approximately correct" with high probability. The sample complexity of $\textsc{learn}$ is the number of calls to the example oracle $\textsc{ex}(h^*, \mathscr{D})$, that is, the number m of training examples required to output with confidence $1 - \delta$, an ε-accurate hypothesis. If m is polynomial in $1/\delta$ and $1/\varepsilon$, then the hypothesis class $\mathscr{H}$ is called PAC *learnable with polynomial sample complexity*.

Though conceptually elegant, the PAC learning framework relies on some realizability condition which is unrealistic in practice. Indeed in many, if not most, statistical learning problems, there is no well-defined target model that perfectly labels incoming instances. For example, if we choose the class $\mathscr{H}$ of decision trees for learning to filter spam messages, we are not guaranteed that $\mathscr{H}$ will always include a decision tree that accurately filters any possible electronic message. This realizability assumption is relaxed in the *agnostic* PAC learning framework, which is general enough to cover both supervised and unsupervised learning tasks involving arbitrary distributions over their instance space (Haussler 1992). In essence, the agnostic PAC learning framework follows the general setting of statistical learning, investigated by (Vapnik, 1998, 2013).

Definition 2 (*Agnostic PAC Learning*) Let $\mathscr{Z}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \rightarrow \mathbb{R}$ be a loss function. Then, $\mathscr{H}$ is *agnostic* PAC *learnable* with respect to ℓ if there exist an algorithm LEARN with the following property: for any distribution $\mathscr{D}$ over $\mathscr{Z}$, and any $(\delta, \varepsilon) \in (0, 1)^2$, if LEARN is given inputs δ and ε, and access to $\text{EX}(\mathscr{D})$, then LEARN returns a hypothesis $h \in \mathscr{H}$ that satisfies, with probability $1 - \delta$,

$$L_{\mathscr{D}}(h) - \inf_{h' \in \mathscr{H}} L_{\mathscr{D}}(h') \leq \varepsilon \tag{5}$$

Again, agnostic PAC learning is a distribution-free model: for every distribution over the instance space, the learner is ask to find with high probability, a model whose performance is near to that of the best model in its hypothesis class. It is important to emphasize that, in the agnostic case, $\mathscr{D}$ is a arbitrary distribution over the *whole* instance space $\mathscr{Z}$, and $\text{EX}(\mathscr{D})$ is a procedure that returns on each call an instance $z \in \mathscr{Z}$ drawn independently at random according to $\mathscr{D}$. In particular, if $\mathscr{Z}$ is the instance space $\mathscr{X} \times \mathscr{Y}$ of a supervised learning task, then $\mathscr{D}$ is an arbitrary joint distribution over $\mathscr{X} \times \mathscr{Y}$, and $\text{EX}(\mathscr{D})$ generates a sample $(\boldsymbol{x}, \mathbf{y})$ where $\boldsymbol{y}$ is not determined by some hypothetical target function, but drawn at random with probability $\mathscr{D}(\boldsymbol{y} \mid \boldsymbol{x})$, whenever $\boldsymbol{x}$ is drawn at random with probability $\mathscr{D}(\boldsymbol{x})$.

3.2 Runtime Complexity

Based on the definition of agnostic PAC learnability, we might be tempted to characterize the runtime complexity of a PAC algorithm LEARN as the amount of computation it performs for returning with probability $1 - \delta$ a hypothesis whose risk is ε-close to the best possible risk. Yet, this measure is not really satisfactory, because we have swept under the rug two key issues.

The first issue is related to the input of the learning algorithm A. Typically, the runtime complexity of A does not only depend on the accuracy (ε) and confidence (δ) parameters, but also on the *dimension* d of the learning task. A natural approach for incorporating this parameter in the input of a statistical learning problem is consider *stratified* classes parameterized by d. Formally, a stratified instance space is a set $\mathscr{Z} = \bigcup_{d \in \mathbb{N}} \mathscr{Z}_d$, where each $\mathscr{Z}_d$ is a subset of $\mathbb{R}^d$. A stratified hypothesis class is defined in a similar way using $\mathscr{H} = \bigcup_{d \in \mathbb{N}} \mathscr{H}_d$. For example, if $\mathscr{H}$ is the class of hyperplanes of arbitrary dimension, then $\mathscr{H}_1$ is the set of points in the line $\mathbb{R}$, $\mathscr{H}_2$ is the set of lines in the plane $\mathbb{R}^2$, $\mathscr{H}_3$ is the set of planes in the space $\mathbb{R}^3$, and so on. By extension, a stratified class of loss functions is a set $\mathscr{L} = \{\ell_d\}_{d\in\mathbb{N}}$, where each ℓ_d is a mapping $\mathscr{H}_d \times \mathscr{Z}_d \to \mathbb{R}$. Based on these stratified classes, the generalization ability of the algorithm LEARN is analyzed for every dimension d, every confidence δ and accuracy ε, and every distribution $\mathscr{D}$ over $\mathscr{Z}_d$.

The second issue is related to the output of the learner. Specifically, the model returned by a computer algorithm is not an abstract function $h \in \mathscr{H}$, but a symbolic *representation* of this mathematical object. From a computational viewpoint, this representation would be of little use if an exponential amount of computational resources was needed for inferring $h(\boldsymbol{x})$ given some incoming instance $\boldsymbol{x}$. So, to alleviate this issue, each candidate hypothesis $h \in \mathscr{H}$ should be associated with a representation for which the inference task is tractable. To this end, let $\mathscr{R}$ be a set of finite strings defined over some alphabet Σ. Then, $\mathscr{R}$ is called a *representation class* for $\mathscr{H}$ if there exist a surjective function from $\mathscr{R}$ to $\mathscr{H}$: each representation $\boldsymbol{r} \in \mathscr{R}$ is associated with exactly one hypothesis in $\mathscr{H}$ denoted $h_{\boldsymbol{r}}$, and each hypothesis $h \in \mathscr{H}$ is associated with at least one representation $\boldsymbol{r} \in \mathscr{R}$ such that $h_{\boldsymbol{r}} = h$. By extension, $\mathscr{R} = \bigcup_{d\in\mathbb{N}} \mathscr{R}_d$ is called a *stratified representation class* for $\mathscr{H} = \bigcup_{d\in\mathbb{N}} \mathscr{H}_d$ if for each dimension d, $\mathscr{R}_d$ is a representation class of $\mathscr{H}_d$.

With these notions in hand, we are now in position to provide a formal model of computationally efficient learnability. The next definition is essentially a variant of the computational learning models presented in Kearns et al. (1994b), Shalev-Shwartz and Ben-David (2014).

Definition 3 (*Efficient Agnostic PAC Learning*) Let $\mathscr{Z}$ be a stratified instance space, $\mathscr{H}$ be a stratified hypothesis class, and $\mathscr{L}$ be a stratified class of loss functions over $\mathscr{H}$ and $\mathscr{Z}$. In addition, let $\mathscr{R}$ be a stratified representation class for $\mathscr{H}$. Then, $\mathscr{H}$ is *efficiently agnostic* PAC *learnable* with respect to ℓ and $\mathscr{R}$ if both the following conditions hold:

- **Polynomial inference**: There exist an algorithm EVAL such that for every positive integer d, every representation $\boldsymbol{r} \in \mathscr{R}_d$, and every instance $\boldsymbol{x} \in \mathscr{Z}_d$, if EVAL is given inputs $\boldsymbol{r}$ and $\boldsymbol{x}$, then EVAL returns $h_{\boldsymbol{r}}(\boldsymbol{x})$ in time polynomial in d.
- **Polynomial convergence**: There exist an algorithm LEARN such that for every positive integer d, every distribution $\mathscr{D}$ over $\mathscr{Z}_d$, and every $(\delta, \varepsilon) \in (0,1)^2$, if LEARN is given inputs d, δ and ε, and access to EX$(\mathscr{D})$, then LEARN returns in time polynomial in d, $1/\delta$ and $1/\varepsilon$ a representation $\boldsymbol{r} \in \mathscr{R}_d$ that satisfies, with probability $1-\delta$,

$$L_{\mathscr{D}}(h_{\boldsymbol{r}}) - \inf_{h' \in \mathscr{H}_d} L_{\mathscr{D}}(h') \leq \varepsilon$$

Conceptually, there is a fundamental difference between "statistical learnability" specified in Definition 2, and "computational learnability" characterized by Definition 3. On the one hand, a hypothesis class $\mathscr{H}$ is statistically learnable if we can find an algorithm that converges with high probability to the best hypothesis in $\mathscr{H}$, using a *finite* number of calls to the example oracle. On the other hand, computational learnability imposes much stronger conditions. In order to establish that $\mathscr{H}$ is efficiently learnable, we must not only prove that $\mathscr{H}$ is tractable for inference, but also find an algorithm that converges in probability to the best model, using a polynomial amount of operations. Since each call to the example oracle takes unit time, this directly implies that $\mathscr{H}$ must be learnable with polynomial sample complexity.

This crucial difference will be illustrated in the forthcoming sections. Many hypothesis classes of interest in the AI literature are learnable, even in the agnostic case, if computational considerations are not taken into account. By contrast, very few of them are *efficiently* learnable. An important class of statistical learning problems satisfying the property of efficient learnability is the family of convex learning problems, examined in Sect. 6.

4 Learning as Optimization

Arguably, statistical learning shares strong similarities with optimization problems. Based on the framework presented in Sect. 2.4, any statistical learning problem can be viewed as a *stochastic optimization problem*, involving a decision variable h defined over $\mathscr{H}$, a random variable z specified by a probability distribution $\mathscr{D}$ over $\mathscr{Z}$, and a loss function $\ell : \mathscr{H} \times \mathscr{Z} \to \mathbb{R}$. The problem is to

$$\begin{aligned} &\text{minimize} \;\; \mathbb{E}_{z\sim\mathscr{D}}[\ell(h,z)] \\ &\text{subject to} \;\; h \in \mathscr{H} \end{aligned} \tag{6}$$

Recall that the expression $\mathbb{E}_{z\sim\mathscr{D}}[\ell(h,z)]$ is the true risk of h, denoted $L_{\mathscr{D}}(h)$. The key specificity - and difficulty - of statistical learning lies in the fact that this objective function cannot be directly evaluated, since the underlying distribution $\mathscr{D}$ is unknown. In other words, statistical learning is a *black-box* stochastic optimization problem, for which the objective function can only be approximated using a limited number of calls to an example oracle $\text{Ex}(\mathscr{D})$. In the statistical learning literature, various optimization principles have been proposed for replacing the unknown risk function (6) with a known, evaluable objective function. In this section, we begin to review several optimization principles, and next, we examine some general conditions for learnability which justify the use of these principles, and open the door to new optimization strategies.

4.1 Optimization Principles

In statistical learning, the data generation process $\mathscr{D}$ is unknown, but we still do have access to a training sample S, given explicitly by a dataset, or implicitly through an example oracle EX($\mathscr{D}$). Let $\mathscr{S}$ denote the set of all finite training sets over $\mathscr{Z}$, that is, $\mathscr{S} = \bigcup_{m \in \mathbb{N}} \mathscr{Z}^m$. The main idea behind most optimization principles in statistical learning is to replace the unknown objective function (6) defined over $\mathscr{D}$, with an evaluable objective function f_S defined for every training set $S \in \mathscr{S}$. The corresponding optimization problem is to

$$\begin{aligned} \text{minimize} \quad & f_S(h) \\ \text{subject to} \quad & h \in \mathscr{H} \end{aligned} \tag{7}$$

A *learning rule* is a map $A : \mathscr{S} \to \mathscr{H}$ that takes as input a training set $S \in \mathscr{S}$, and returns as output a hypothesis $A(S) \in \mathscr{H}$. We note in passing that any agnostic PAC learning algorithm LEARN can be unambiguously specified by a learning rule A and an integer-valued function $m : (0, 1)^2 \to \mathbb{N}$. Namely, given as input a desired confidence δ and a desired accuracy ε, the algorithm LEARN starts by picking a training set $S \in \mathscr{S}$ by calling $m(\delta, \varepsilon)$ times the example oracle EX($\mathscr{D}$), and then uses the learning rule A with S in order to produce a model $A(S) \in \mathscr{H}$. Here, $m(\delta, \varepsilon)$ captures the sample complexity of LEARN.

Based on these considerations, we say that a learning rule $A : \mathscr{S} \to \mathscr{H}$ *solves* the optimization task (7) if for every input $S \in \mathscr{S}$, the algorithm A returns as output a hypothesis $A(S) \in \mathscr{H}$ satisfying $f_S(A(S)) = \inf_{h \in \mathscr{H}} f_S(h)$. If in addition A runs in time polynomial in the dimension d of the training instances, and the size m of the training set S, then we say that A *efficiently solves* the optimization task (7).

4.1.1 Empirical Risk Minimization

Perhaps the most common approach for handling statistical learning problems is to replace the true risk function $L_{\mathscr{D}}$ by the empirical risk function L_S that measures the average loss of a model on the observed instances (Vapnik 1998; Zhang 2010). Based on this principle, called *Empirical Risk Minimization* (ERM), the objective function f_S, defined for a training set $S = (z_1, \ldots, z_m)$, is given by

$$f_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, z_i)] = L_S(h) \tag{8}$$

Correspondingly, any learning rule $A : \mathscr{S} \to \mathscr{H}$ that solves the optimization task (7), using (8) as objective function, is called an *empirical risk minimizer*.

Borrowing the terminology of stochastic optimization, the ERM principle is equivalent to the paradigm of *sample average approximation*, which aims at approximating the expected value function by a sample average function (Birge and Louveaux 2011). Though this idea is conceptually simple, and statistically justified by the law of large numbers, we must keep in mind that L_S is only an estimator of $L_{\mathscr{D}}$, In practice, the divergence between these objective functions depend on the choice of the hypothesis class $\mathscr{H}$ and the available training set S. More precisely, the true risk of the hypothesis $A(S)$ returned by an empirical risk minimizer A can be decomposed as the sum of two terms:

$$L_{\mathscr{D}}(A(S)) = \underbrace{\inf_{h \in \mathscr{H}} L_{\mathscr{D}}(h)}_{\text{approximation}} + \underbrace{\left[L_{\mathscr{D}}(A(S)) - \inf_{h \in \mathscr{H}} L_{\mathscr{D}}(h) \right]}_{\text{estimation}}$$

The approximation term measures the minimum risk achievable by any possible model in the hypothesis class $\mathscr{H}$. The estimation term evaluates the performance of the hypothesis $A(S)$ chosen by the learning rule A, relatively to the best model in $\mathscr{H}$. By minimizing the sum of both terms, we are faced with a dilemma between approximation and estimation, called *bias-complexity trade-off* (Shalev-Shwartz and Ben-David 2014). On the one hand, if we choose a very rich hypothesis class $\mathscr{H}$, then we decrease the approximation error by covering good models for the task at hand but, at the same time, we increase the sample complexity required to guarantee that, with high probability, training sets are representative of the underlying distribution $\mathscr{D}$. Thus, if the available training set S is too small for achieving this guarantee, the objective function f_S is likely to be a poor estimator of $L_{\mathscr{D}}$, and hence, the hypothesis $A(S)$ is prone to *overfitting*, by having an optimal performance on training data, but a poor performance on test data. On the other hand, if we choose a very small hypothesis class $\mathscr{H}$, then we increase the odds that the available training set is representative, but we also increase the approximation error by missing good models for the learning task. So here, $A(S)$ is prone to *underfitting*, by exhibiting a relatively stable, but low performance, on both training data and test data.

4.1.2 Structural Risk Minimization

A natural idea to prevent overfitting situations is to penalize complex hypotheses, in favor of simpler ones, whenever they share the same empirical risk. This idea follows the well-known law of parsimony, according to which *plurality should not be posited without necessity*. This law, called *Occam's razor* after the philosopher William of Ockham, gives precedence to simplicity: of two competing theories, the simpler explanation of an entity is to be preferred.

In the paradigm of *Structural Risk Minimization* (SRM), due to Vapnik and Chervonenkis (Vapnik and Chervonenkis 1974), it is assumed that the hypothesis class $\mathscr{H}$ is associated with a stratified representation class $\mathscr{R} = \bigcup_{k \in \mathbb{N}} \mathscr{R}_k$, where k is a structural parameter. For instance, if $\mathscr{H}$ is the class of all (zero-threshold) separating hyperplanes over the domain set $\mathscr{X} \subseteq \mathbb{R}^n$, then its representation class $\mathscr{R} \subseteq \mathbb{R}^n$ can be stratified by the number k of nonzero weights. Namely, each stratum $\mathscr{R}_k$ is the set of all weight vectors $\boldsymbol{w} \in \mathbb{R}^n$ such that $\|\boldsymbol{w}\|_0 \leq k$. Given a model $h \in \mathscr{H}$, we use k_h to denote the smallest integer k such that $h = h_{\boldsymbol{r}}$ for at least one representation $\boldsymbol{r} \in \mathscr{R}_k$. Based on these notions, the objective function is a mapping of the form

$$f_S(h) = L_S(h) + \text{pen}_{m,\delta}(k_h) \tag{9}$$

where $\text{pen}_{m,\delta}$ is a penalty function that depends on the size m of the training set, and a confidence parameter $\delta \in (0, 1)$. Ideally, the penalty function should satisfy the condition that for every confidence $\delta \in (0, 1)$ and every distribution $\mathscr{D}$ over the instance space, with probability $1 - \delta$ over the choice of $S \sim \mathscr{D}^m$, the following bound holds for any hypothesis $h \in \mathscr{H}$:

$$|L_{\mathscr{D}}(h) - L_S(h)| \leq \text{pen}_{m,\delta}(k_h) \tag{10}$$

If this condition is indeed satisfied, then the estimation error of h is bounded by $L_S(h) + \text{pen}_{m,\delta}(k_h)$. In other words, the SRM principle handles the bias-complexity trade-off by giving preference to simple hypotheses (with small penalty value) which behave well on the training set.

A closely related paradigm is the *Minimum Description Length* (MDL) principle, due to Rissanen (1983, 1985), and surveyed in detail by Grünwald (2007). Here, it is assumed that the hypothesis class $\mathscr{H}$ is associated with a *prefix-free* representation class $\mathscr{R}$. Namely, $\mathscr{R}$ is a prefix-free language if no representation $\boldsymbol{r} \in \mathscr{R}$ is the prefix of a distinct representation $\boldsymbol{r}' \in \mathscr{R}$. Notice that $\mathscr{R}$ can be viewed as a stratified representation class $\bigcup_{k \in \mathbb{N}} \mathscr{R}_k$, where $\mathscr{R}_k$ is the set of all representations, or "codewords", of length k. Based on this observation, h_k measures the length of the smallest codeword $\boldsymbol{r}$ such that $h = h_{\boldsymbol{r}}$, and it is simply denoted $|h|$. In the MDL principle, the objective function is given by

$$f_S(h) = L_S(h) + \text{pen}_{m,\delta}(|h|) \text{ where } \text{pen}_{m,\delta}(|h|) = \sqrt{\frac{|h| + \ln \frac{2}{\delta}}{2m}} \tag{11}$$

Notably, using the well-known Kraft's inequality property of prefix-free languages, it can be shown that the penalty function $\text{pen}_{m,\delta}(|h|)$ satisfies the condition (10). A detailed proof is given in Shalev-Shwartz and Ben-David (2014).

To sum up, the MDL paradigm provides an elegant way to circumvent the pitfall of overfitting in rich hypothesis classes, by penalizing models with their code length. However, the MDL principle does not come without practical issues: a key difficulty in the design of MDL-based learning algorithms is to find an appropriate prefix free representation language for the hypothesis class at hand. Another important issue is the runtime complexity of the optimization task. Notably, if the loss function ℓ is convex, then ERM objective (8) remains convex, but the MDL objective (11) is generally not convex due to the additional, non-convex, penalty term. Similar computational issues arise for the more general SRM principle, for which penalty functions in (9) are typically not convex.

4.1.3 Regularized Risk Minimization

For hypothesis classes $\mathscr{H}$ represented by linear functions, the predominant approach to penalize complex models is through "regularizing" their representation. In this setting, called *Regularized Risk Minimization* (RRM), the representation class of $\mathscr{H}$ is a set of weight vectors, denoted here $\mathscr{W}$. The objective function f_S takes the following form:

$$f_S(h) = L_S(h) + \mathrm{reg}(\boldsymbol{w}) \tag{12}$$

where $\boldsymbol{w}$ is the vector representation of h, and reg : $\mathscr{W} \to \mathbb{R}$ is a regularization term that penalizes hypotheses according to the "complexity" of their vector representation. The complexity of vectors is typically measured using some norm over $\mathscr{W}$. For example, the regularizer $\mathrm{reg}(\boldsymbol{w}) = \lambda \, \|\boldsymbol{w}\|_2^2$ due to Tikhonov (1943), penalizes weights with large magnitudes. Alternatively, the regularizer $\mathrm{reg}(\boldsymbol{w}) = \lambda \, \|\boldsymbol{w}\|_1$ gives preference to parsimonious models involving few nonzero weights. In both expressions, the parameter λ is a positive scalar that controls the regularization effect. We emphasize that regularization functions are not always defined through norms. For instance, the entropic regularizer $\mathrm{reg}(\boldsymbol{w}) = \lambda \sum_i w_i \ln {}^1\!/_{w_i}$ is often used when the representation class $\mathscr{W}$ is a probability simplex.

Obviously, the RRM paradigm shares strong similarities with the SRM principle: both approaches aim at preventing overfitting issues by penalizing models which are excessively complex for the task at hand. From a pragmatic viewpoint, there are, yet, important differences related to the formulation of the statistical learning problem as an optimization task, and the resolution of this optimization task. The regularization term in RRM is often specified by a simple analytic form, while the penalty term in SRM is typically much more difficult to characterize. For example, the penalty term in (11) is defined using the code length $|h|$ of a model $h \in \mathscr{H}$, which requires a prefix-free representation language for $\mathscr{H}$. Furthermore, most regularization terms in the Machine Learning literature are convex functions. If, in addition, the representation class $\mathscr{W}$ is convex, and the loss function is convex for $\mathscr{W}$, then the optimization task (7) using (12) as objective function is a convex optimization problem, which can be efficiently solved by a wide variety of algorithms. As mentioned above, objective functions for SRM and MDL principles typically lead to intractable optimization tasks, due to the non-convex nature of penalty terms.

4.2 Conditions for Learnability

The overall goal of optimization principles in statistical learning is to reformulate the black-box stochastic optimization task (6) as a standard, well-formed, optimization task (7). If we put aside computational considerations, there is still an important question that emerges from those principles: under which conditions an optimization algorithm for (7) is guaranteed, with high probability, to solve (6)?

In the statistical learning literature, various conditions for learnability have been proposed, in order to characterize the key relationships between learning and optimization (Vapnik 1998; Bousquet and Elisseeff 2002; Poffio et al. 2004; Mohammadi and van de Geer 2005; Mukherjee et al. 2006; Watanabe 2009; Wibisono et al. 2009; Shalev-Shwartz et al. 2010; Liu et al. 2017). We shall concentrate on two of them, namely, *uniform convergence* and *stability*, which play a central role in statistical learning theory.

To this end, we need some additional definitions. A *rate function* is a monotone decreasing mapping $\epsilon \colon \mathbb{N} \to \mathbb{R}$ that converges to 0 as m tends to infinite. With these notions in hand, a learning rule A is called *(universally) consistent* with rate ϵ_{cons} if for any $m \in \mathbb{N}$ and any distribution $\mathscr{D}$ over $\mathscr{Z}$,

$$\mathbb{E}_{S\sim\mathscr{D}^m}[L_{\mathscr{D}}(A(S))] - \inf_{h\in\mathscr{H}} L_{\mathscr{D}}(h) \leq \epsilon_{\text{cons}}(m)$$

The next result, derived from Shalev-Shwartz et al. (2010), Sridharan (2012), states that consistency is a necessary and sufficient condition for achieving learnability in the setting of *bounded* loss functions, that is, cost functions of the form $\ell : \mathscr{H} \times \mathscr{Z} \to [0, b]$, where b is a positive scalar.

Theorem 1 (Learnability as Consistency) *Let $\mathscr{Z}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \to [0, b]$ be a bounded loss function. Then, $\mathscr{H}$ is (agnostic PAC) learnable with respect to ℓ if and only if there is a learning rule A for $\mathscr{H}$ and a rate function ϵ_{cons} such that A is consistent with rate ϵ_{cons}.*

4.2.1 Uniform Convergence

For bounded loss functions, the statistical learning problem is to find a learning rule that achieves a uniform rate for all distributions. To this point, it is well-known that the empirical risk minimizer is consistent, provided that its hypothesis class satisfies the *uniform convergence* property (Vapnik 1998, 2013). This key condition for learnability can be formalized in the following way.

Definition 4 (*Uniform Convergence*) Let $\mathscr{Z}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \to \mathbb{R}$ be a loss function. Then, $\mathscr{H}$ has the *uniform convergence property* with respect to ℓ if for every distribution $\mathscr{D}$ over $\mathscr{Z}$,

$$\lim_{m\to\infty} \mathbb{E}_{S\sim\mathscr{D}^m}\left[\sup_{h\in\mathscr{H}} |L_{\mathscr{D}}(h) - L_S(h)|\right] = 0$$

Intuitively, the quantity $\sup_{h\in\mathscr{H}} |L_{\mathscr{D}}(h) - L_S(h)|$ measures the ability of a training set S to adequately represent the underlying distribution $\mathscr{D}$ for the task at hand. Given an accuracy parameter ε, the training set S is called ε *-representative* if for all hypotheses $h \in \mathscr{H}$, we have $|L_{\mathscr{D}}(h) - L_S(h)| \leq \varepsilon$. Based on this notion, a hypothesis class $\mathscr{H}$ has the uniform convergence property if there exist an integer-valued function $m_{\mathscr{H}} : (0, 1)^2 \to \mathbb{N}$ such that, for every pair $(\delta, \varepsilon) \in (0, 1)^2$, and every distribution $\mathscr{D}$ over $\mathscr{Z}$, if the example oracle $\textsc{ex}(\mathscr{D})$ is called $m \geq m_{\mathscr{H}}(\delta, \varepsilon)$ times, then the resulting sample $S \in \mathscr{Z}^m$ is ε-representative with probability $1 - \delta$. Based on this reformulation of uniform convergence, the metric $m_{\mathscr{H}}$ shares similarities with the sample complexity of learning. Specifically, $m_{\mathscr{H}}(\delta, \varepsilon)$ is the amount of information needed to ensure that, with probability $1 - \delta$, the training set S supplied to the learner is ε-representative. Thus, if S is sufficiently large, then the empirical risk of hypotheses is a faithful approximation of their true risk. The ERM principle (8) can therefore be used without the need of penalty or regularization terms.

Theorem 2 (Learnability via Uniform Convergence) *Let $\mathscr{Z}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \to [0, b]$ be a bounded loss function. If $\mathscr{H}$ has the uniform convergence property with sample complexity $m_{\mathscr{H}}(\delta, \varepsilon)$, then $\mathscr{H}$ is (agnostic PAC) learnable with sample complexity $m_{\mathscr{H}}(\delta, \varepsilon/2)$, and the empirical risk minimizer is consistent.*

Interestingly, for supervised classification and regression tasks, a converse result also holds; namely, $\mathscr{H}$ is learnable *if and only if* it enjoys the uniform convergence property (Blumer et al. 1989; Alon et al. 1997).

For rich hypothesis classes $\mathscr{H}$, the sample complexity $m_{\mathscr{H}}(\delta, \varepsilon)$ required to ensure uniform convergence can be much larger than the size of training sets available in practice, and hence, the ERM rule is prone to overfitting. So, we need here a weaker form of uniform convergence that justifies the use of alternative principles, such as SRM. To this end, assume that $\mathscr{H}$ is associated with a stratified representation class $\mathscr{R} = \bigcup_{k \in \mathbb{N}} \mathscr{R}_k$, and let $\mathscr{H}_k$ be the set of models represented by $\mathscr{R}_k$. Then, $\mathscr{H}$ is said to have the *locally uniform convergence* property if each $\mathscr{H}_k$ enjoys the uniform convergence condition with sample complexity $m_{\mathscr{H}_k}$. Intuitively, the quantity $m_{\mathscr{H}_k}(\delta, \varepsilon)$ is small for simple hypothesis classes $\mathscr{H}_k$, and increases with the structural parameter k. Given a sample size m, let $\varepsilon_k(m, \delta)$ be the minimum value of $\varepsilon \in (0, 1)$ for which $m_{\mathscr{H}_k}(\delta, \varepsilon) \leq k$. Since $\mathscr{H}_k$ has the uniform convergence property, it follows that any training sample $S \sim \mathscr{D}^m$ is $\varepsilon_k(m, \delta)$-representative with probability $1 - \delta$. Thus, the penalty rule

$$\text{pen}_{m,\delta}(k_h) = \varepsilon\left(m, \frac{\delta}{2^{k_h}}\right)$$

satisfies the condition (10), which in turn implies that any structural risk minimizer defined on this penalty rule is consistent. In a nutshell, the locally uniform convergence property is a sufficient condition for learnability using the SRM paradigm.

4.2.2 Stability

In contrast with uniform convergence, a condition defined for hypothesis classes, stability is a property related to learning rules. Intuitively, a learning algorithm is characterized by an overfitting behavior when it overreacts to small fluctuations in the training data. Put another way, a learning rule $A : \mathscr{S} \rightarrow \mathscr{H}$ is stable if a small change of the input $S \in \mathscr{S}$ will only induce a small change of the output $h \in \mathscr{H}$.

The next definition of stability, often referred to as *average replace-one stability*, is based on replacing one instance in the training set, with a new instance drawn at random according to the underlying distribution. Given a sample $S = (z_1, \ldots, z_m)$ and an instance $z' \in \mathscr{Z}$, let $S_{z_i \leftarrow z'} = (z_1, \ldots, z_{i-1}, z', z_{i+1}, \ldots, z_m)$ be the sequence obtained by replacing the ith observation of S with the instance z'.

Definition 5 (*Stability*) Let $\mathscr{Z}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \rightarrow \mathbb{R}$ be a loss function. Then, a learning rule A for $\mathscr{H}$ is *(on average replace-one) stable* with rate ϵ_{stable} if for any distribution $\mathscr{D}$ over $\mathscr{Z}$,

$$\frac{1}{m}\left|\sum_{i=1}^{m} \mathbb{E}_{S\sim\mathscr{D}^m,(z'_1,\ldots,z'_m)\sim\mathscr{D}^m}\left[\ell\big(A(S_{z_i\leftarrow z'_i}); z'_i\big) - \ell\big(A(S); z'_i\big)\right]\right| \leq \epsilon_{\text{stable}}\ (m)$$

For stable learning rules, the ERM principle is *not* a necessary condition for ensuring learnability. Instead, the learner is only required to converge toward the ERM minimizer when the number m of training instances tends to infinite. Formally, a learning rule A is an *Asymptotic Empirical Risk Minimizer* (AERM) with rate ϵ_{erm} if for any distribution $\mathscr{D}$ over $\mathscr{Z}$,

$$\mathbb{E}_{S\sim\mathscr{D}^m}\left[L_S(A(S)) - \inf_{h\in\mathscr{H}} L_S(h)\right] \leq \epsilon_{\text{erm}}\ (m)$$

The next result, demonstrated in Shalev-Shwartz et al. (2010), establishes an equivalence between statistical learnability and stable AERM rules.

Theorem 3 (Learnability via Stability) *Let $\mathscr{Z}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \rightarrow [0, b]$ be a bounded loss function. Then $\mathscr{H}$ is (agnostic PAC) learnable if and only if there exists a stable* AERM *for $\mathscr{H}$. In particular, if a learning rule A is stable with rate ϵ_{stable} and* AERM *with rate ϵ_{erm}, then A is consistent with rate*

$$\epsilon_{\text{cons}}\ (m) \leq \epsilon_{\text{stable}}\ (m) + \epsilon_{\text{erm}}\ (m)$$

In a nutshell, uniform convergence and stability provide different mathematical tools for building learning algorithms. If the hypothesis class $\mathscr{H}$ is endowed with uniform convergence, then Empirical Risk Minimization is the paradigm of choice for designing a learning rule with a good generalization ability. Yet, $\mathscr{H}$ may be learnable even if it does not satisfy the uniform convergence property: in this case, stable asymptotic empirical risk minimizers are guaranteed to work. For convex learning problems described in Sect. 6, such learning rules can be constructed in a simple and intuitive manner using the Regularized Risk Minimization principle.

5 Concept Learning

Basically, the problem of concept learning is to extrapolate, from a series of positive and negative examples, a model that accurately separate future, unseen instances. In other words, concept learning problems are binary classification tasks whose objective function is the zero-one loss. The instance space $\mathscr{Z}$ is a set $\mathscr{X} \times \{0, 1\}$ of instances labeled as negative (0) or positive (1). A concept is a subset of $\mathscr{X}$, or equivalently, an indicator function h mapping $\mathscr{X}$ to $\{0, 1\}$. By extension, a concept class is a subset $\mathscr{H} \subseteq \{0, 1\}^{\mathscr{X}}$. Recall that the zero-one loss function ℓ over $\mathscr{H}$ and $\mathscr{Z}$ is given by:

$$\ell(h; (\boldsymbol{x}, y)) = \begin{cases} 0 & \text{if } h(\boldsymbol{x}) = y \\ 1 & \text{otherwise} \end{cases}$$

Based on this objective function, the true risk and the empirical risk of a concept can be viewed as error measures. Namely, $L_{\mathscr{D}}(h)$ captures the probability that the concept h is making a mistake on a labeled instance $(\boldsymbol{x}, y)$ drawn at random according to $\mathscr{D}$. $L_S(h)$ is the proportions of mistakes made by h on the training set S.

In this section, we begin to examine the *Vapnik-Chervonenkis dimension* of concept classes, an important notion related to their sample complexity. We next survey some theoretical results related to learning concepts in the realizable case and the agnostic case. We close this section by briefly discussing about *bagging* and *boosting*, two efficient techniques for learning combinations of models.

5.1 VC-Dimension

As explained in Sect. 4.2, the uniform convergence property is a sufficient condition for establishing the learnability of hypothesis classes. In concept learning, this property is intrinsically related to the classification power of the concept class, called *Vapnik-Chervonenkis* (VC) *dimension* (Vapnik and Chervonenkis 1974). Intuitively, the VC-dimension of $\mathscr{H}$ is the maximum size of any set of input objects which can be labeled in any possible way using concepts taken from $\mathscr{H}$. More formally, let $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\} \subseteq \mathscr{X}$ be a set of m input objects, and let

$$\mathscr{H}_S = \{(h(\boldsymbol{x}_1), \ldots, h(\boldsymbol{x}_m)) : h \in \mathscr{H}\}$$

be the restriction of $\mathscr{H}$ to S, that is, the set of functions from S to $\{0, 1\}$ which can be derived from $\mathscr{H}$. Then, S is called *shattered* by $\mathscr{H}$ if $\mathscr{H}_S$ is the set of all possible Boolean functions from S to $\{0, 1\}$, that is, $|\mathscr{H}_S| = 2^{|S|}$.

Definition 6 (*VC-dimension*) Let $\mathscr{X}$ be a set, and $\mathscr{H}$ be a set of functions from $\mathscr{X}$ to $\{0, 1\}$. Then, the VC-*dimension* of $\mathscr{H}$, denoted $\text{VCdim}(\mathscr{H})$, is the maximal size

Table 1 VC-dimension of some concept classes. A k-term DNF formula is a disjunction of at most k monomials, and a k-DNF formula is a disjunction of monomials, with at most k literals per term

Concept class	VCdim
Monotone monomials on $\{0,1\}^n$	n
Homogeneous Linear functions on $\mathbb{R}^n$	n
Linear threshold functions on $\mathbb{R}^n$	$n+1$
Feedforward linear threshold neural networks with E edges on $\mathbb{R}^n$	$6E\log_2 E$
k-term DNF formulas on $\{0,1\}^n$	$\Theta(kn)$
k-DNF formulas on $\{0,1\}^n$	$\Theta(n^k)$
Polynomial threshold functions of degree k on $\mathbb{R}^n$	$\binom{n+k}{k}$
Arbitrary DNF formulas on $\{0,1\}^n$	2^n
Arbitrary functions from $\mathbb{R}^n$ to $\{0,1\}$	∞

of any set $S \subseteq \mathscr{X}$ that is shattered by $\mathscr{H}$. If $\mathscr{H}$ can shatter sets of arbitrary large size, then $\text{VCdim}(\mathscr{H}) = \infty$.

We mention in passing that for a finite class $\mathscr{H}$, a set S of instances cannot by shattered by $\mathscr{H}$ if $|\mathscr{H}| < 2^{|S|}$. It follows that

$$\text{VCdim}(\mathscr{H}) \leq \log_2 |\mathscr{H}|$$

Actually, the VC-dimension of finite concept classes $\mathscr{H}$ can be much smaller than the logarithm of their size. Consider for example the class $\mathscr{H} = \{h_1, \ldots, h_n\}$ of Boolean functions from $\{0,1\}^n \to \{0,1\}$, defined as follows: $h_i(\boldsymbol{x}) = 1$ if and only if all features in $\boldsymbol{x}$ ranging from i to n are set to 1. Clearly, $\mathscr{H}$ can shatter a singleton set $S = \{\boldsymbol{x}\}$ using $x_1 = 0$ and $x_2 = 1$. Yet, $\mathscr{H}$ cannot shatter any pair of input objects $S = \{\boldsymbol{x}, \boldsymbol{x}'\}$, because there is no pair of hypotheses $\mathscr{H}$ for which the first gives the labeling $(0,1)$ and the second gives the opposite labeling $(1,0)$. So, the VC-dimension of $\mathscr{H}$ is 1, and since n can be arbitrary large, the gap between $\text{VCdim}(\mathscr{H})$ and $\log_2 |\mathscr{H}|$ may be arbitrary large.

The VC-dimension of several concept classes is reported on Table 1; the proofs may be found in Anthony (2001, 2010). It is important to keep in mind that some infinite classes, such as linear threshold functions and feedforward neural networks, have a finite (and sometimes low) VC-dimension. The next theorem is a standard result in statistical learning theory, and its proof can be found in various textbooks (Anthony and Barlett 1999; Vapnik 2013; Shalev-Shwartz and Ben-David 2014).

Theorem 4 (Learnability of Concept Classes) *Let $\mathscr{H}$ be a hypothesis class from a domain $\mathscr{X}$ to $\{0,1\}$, and let ℓ be the zero-one loss function. Then, then following are equivalent:*

- *$\mathscr{H}$ has a finite VC-dimension.*
- *$\mathscr{H}$ has the uniform convergence property.*
- *$\mathscr{H}$ is agnostic PAC learnable.*

In particular, if $\mathrm{VCdim}(\mathscr{H}) \leq d$*, then the sample complexity of* $\mathscr{H}$ *is in* $\mathscr{O}(\frac{d+\ln(1/\delta)}{\varepsilon^2})$.

An important notion related to the VC-dimension is the *growth function* of a hypothesis class, which measures the number of different functions from a set S of size m to $\{0, 1\}$ that can be obtained by restricting $\mathscr{H}$ to S. Formally, the growth function of $\mathscr{H}$, is the mapping $\Pi_{\mathscr{H}} : \mathbb{N} \to \mathbb{N}$ given by

$$\Pi_{\mathscr{H}}(m) = \max_{S \subseteq \mathscr{X} : |S| = m} |\mathscr{H}_S|$$

Clearly, if the VC-dimension of $\mathscr{H}$ is d, then $\Pi_{\mathscr{H}}(m) = 2^d$ for all $m \leq d$. More precisely, by Sauer's Lemma (1972), the growth function of a concept class $\mathscr{H}$ for which the VC-dimension is upper-bounded by d satisfies $\Pi_{\mathscr{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}$ for all $m \in \mathbb{N}$. In particular, when m is becoming larger than d, the growth function is bounded by $(em/d)^d$, that is, $\Pi_{\mathscr{H}}$ increases polynomially with m. As a direct corollary of Theorem 4, if $\mathscr{H}$ has a finite VC-dimension, then $\mathscr{H}$ is agnostic PAC learnable with a sample complexity that is logarithmic in $\Pi_{\mathscr{H}}$.

5.2 *Realizable Concept Learning*

We first explore the PAC learnability of concept classes in the *realizable* setting, where a target function in the concept class is labeling the instances supplied to the learner. A useful algebraic tool in realizable PAC learning is the notion of *version space*, due to Mitchell (1982). Given a concept class $\mathscr{H}$ and a training sample $S \subseteq \mathscr{X} \times \{0, 1\}$, the version space of $\mathscr{H}$ with respect to S is given by

$$\mathrm{VS}(\mathscr{H}, S) = \{h \in \mathscr{H} \mid h(\boldsymbol{x}) = y \text{ for all } (\boldsymbol{x}, y) \in S\}$$

Let $\mathscr{D}$ denote the hidden distribution over $\mathscr{X}$, and $h^* \in \mathscr{H}$ denote the hidden target concept. Given a desired accuracy $\varepsilon \in (0, 1)$, the version space of $\mathscr{H}$ with respect to S is called ε*-exhausted* if $L_{\mathscr{D}}(h) \leq \varepsilon$ for any hypothesis in $\mathrm{VS}(\mathscr{H}, S)$. In other words, all candidate concepts in an ε-exhausted version space have error at most ε with respect to h^*. The following result, established in Blumer et al. (1989), Haussler (1988), provides a relation between ε-exhausted version spaces and the growth function of the concept class.

Theorem 5 *Let* $\mathscr{H}$ *be a hypothesis class from a domain* $\mathscr{X}$ *to* $\{0, 1\}$*, and let* ℓ *be the zero-one loss function. In addition, let* $\mathscr{D}$ *be a arbitrary distribution over* $\mathscr{X}$*, and* $h^* \in \mathscr{H}$ *be a target concept. Then for any* $\varepsilon \in (0, 1)$ *and any training sample* S *of size* m *drawn from* $\mathscr{D}$ *and labeled by* h^**, the probability that* $\mathrm{VS}(\mathscr{H}, S)$ *is not* ε*-exhausted is at most*

$$2\Pi_{\mathscr{H}}(2m)2^{-\varepsilon m/2}$$

As a corollary, if the size m of the training sample S is at least

$$\frac{4}{\varepsilon}\left[\mathrm{VCdim}(\mathscr{H})\log_2\left(\frac{12}{\varepsilon}\right)+\log_2\left(\frac{2}{\delta}\right)\right] \tag{13}$$

the version space is ε-exhausted with probability $1-\delta$. Consequently, the concept class $\mathscr{H}$ is PAC learnable with a sample complexity which is linear in the VC-dimension of $\mathscr{H}$. So, in order to show that logical concept classes of polynomial VC-dimension are *efficiently* PAC learnable in the realizable case, we simply need to devise an algorithm that returns in polynomial time an element in the version space $\mathrm{VS}(\mathscr{H}, S)$, given as input a training sample S of size at least (13). In other words, realizable PAC learning is essentially a consistency (or feasibility) problem: given a set of labeled instances, find a concept that correctly labels all instances.

For simple concept classes, the consistency problem is relatively straightforward. For example, monomials and clauses may be learned using a standard variable elimination algorithm (Mitchell 1982; Kearns et al. 1987). Parity functions represented by XOR clauses can be learned using a closure algorithm (Helmbold et al. 1992). For linear threshold functions, the feasibility problem can be cast as a standard Linear Programming (LP) task, and hence, may be solved in polynomial time using an LP method. Here, the incremental Perceptron algorithm (Rosenblatt 1958) is more attractive in practice, but it is not generally efficient, because the number of its iterations depends on the margin of the training set, which can be exponential in the input dimension n (Anthony and Shawe-Taylor 1993).

Much less obvious is the consistency issue of expressive concept classes. On the one hand, k-DNF are efficiently PAC learnable using a simple extension of the variable elimination algorithm, and decision lists with clauses of size at most k can be efficiently learned using Rivest's algorithm (1987). On the other hand, for k-term DNF formulas, the consistency problem is NP-hard (even for $k = 3$) (Pitt and Valiant 1988). Similar hardness results have been found for expressive classes of geometric models: the consistency problem is NP-hard for k intersections of halfspaces (even for $k = 2$) (Megiddo 1988; Blum and Rivest 1992).

The above negative results hold for realizable and *proper* PAC learning; the concept returned by the learner must be a representation of a model in the hypothesis class $\mathscr{H}$. What about relaxing this condition? Namely, the computational issue of finding a representation of a model in $\mathscr{H}$ that is consistent with the data may be circumvented by allowing the learner to output in polynomial time a representation of a model in some larger concept class $\mathscr{H}'$ that includes $\mathscr{H}$. In this relaxed setting, often referred to as *improper* or *representation independent* PAC learning, the aforementioned class of k-term DNF formulas is efficiently learnable using k-CNF formulas, simply because any disjunction of k monomials can be encoded into a CNF expression, involving at most k literals per clause. Based on a similar encoding, the class of decision trees with at most s leaves is efficiently learnable using $\log_2 s$-decision lists (Blum 1992). In this representation independent setting, various *sub-exponential* time algorithms have been found for learning expressive concepts, such as DNF formulas or intersections of halfspaces, using polynomial threshold representations (Klivans et al. 2004; Klivans and Servedio 2004). Yet, *polynomial* time learning algorithms seem to be unachievable, under the standard assumption that NP $\neq$ RP. Notably, several negative results indicate that DNF formulas are not efficiently learnable in the representation independent setting (Alekhnovich et al. 2008; Daniely and Shalev-Shwartz 2016). Analogous results have been obtained for intersections of halfspaces (Klivans and Sherstov 2009).

5.3 Agnostic Concept Learning

In the agnostic PAC learning setting, which does not make any assumption about the labels of incoming instances, the growth function of a hypothesis class, and hence its VC-dimension, may be used for assessing the sample complexity of binary classification under the zero-one loss. The proof of the next result, related to the sample complexity of uniform convergence, can be found in several textbooks (Mohri et al. 2012; Shalev-Shwartz and Ben-David 2014).

Theorem 6 *Let $\mathscr{H}$ be a hypothesis class from a domain $\mathscr{X}$ to $\{0, 1\}$, and let ℓ be the zero-one loss function. Then, for every distribution $\mathscr{D}$ over $\mathscr{X} \times \{0, 1\}$, every $\delta \in (0, 1)$ and every $h \in \mathscr{H}$, with probability $1 - \delta$ over the choice of $S \sim \mathscr{D}^m$,*

$$|L_D(h) - L_S(h)| \leq \sqrt{\frac{2 \ln \prod_{\mathscr{H}}(m)}{m}} + \sqrt{\frac{\ln {}^1\!/\!_\delta}{2m}}$$

Thus, by combining the above result with Theorem 2, it follows that if $\mathscr{H}$ has a finite VC-dimension, then $\mathscr{H}$ is agnostic PAC learnable with sample complexity

$$O\left(\frac{\mathrm{VCdim}(\mathscr{H}) + \ln {}^1\!/\!_\delta}{\varepsilon^2}\right)$$

As shown in Anthony and Barlett (1999), this asymptotic bound is tight: the O function can be replaced with the Θ function. Thus, the increase of sample complexity is mainly related to the accuracy parameter: the dependence on ${}^1\!/\!_\varepsilon$ is nearly linear in the realizable case, while it is quadratic in the agnostic case.

From a computational viewpoint, a sufficient condition for achieving efficient agnostic PAC learnablity is a polynomial time empirical risk minimizer. Indeed, as established in Theorem 2, the ERM learning rule is statistically consistent whenever $\mathscr{H}$ is endowed with the uniform convergence property. To this point, recall that realizable concept learning is a feasibility problem: find $h \in \mathscr{H}$ such that $L_S(h) = 0$. By contrast, agnostic concept learning is an optimization problem: minimize $L_S(h)$ subject to $h \in \mathscr{H}$. This crucial difference has drastic consequences on the computational learnability of concept classes. Notably, for simple classes such as monotone monomials and linear threshold functions, the problem of finding a concept that minimizes is empirical error on a training sample is NP-hard (Johnson and Preparata

1978; Angluin and Laird 1987; Höffgen and Simon 1992; Kearns and Li 1993; Kearns et al. 1994b). Consequently, monotone monomials and linear threshold functions are *not* efficiently agnostic PAC learnable, unless NP = RP.

In order to alleviate this computational barrier, a natural approach is to consider approximation schemes: for a given approximation parameter $\alpha \geq 1$, an α-approximation algorithm for $\mathscr{H}$ is a polynomial-time algorithm that takes as input an arbitrary sample S, and returns as output a hypothesis $h \in \mathscr{H}$, such that $L_S(h) \leq \alpha \inf_{h' \in \mathscr{H}} L_S(h')$. In other words, the learner must find a concept for which the empirical error is at most α times the empirical error of the ERM rule. Unfortunately, even under this relaxed setting, the problem of approximately minimizing the empirical error of monotone monomials and linear threshold functions remain NP-hard (Arora et al. 1997; Ben-David et al. 2003; Feldman et al. 2009).

Another approach, already suggested for realizable concept learning, is to allow the learner to return hypotheses in some class $\mathscr{H}'$ that covers $\mathscr{H}$. Yet, even in this representation-independent setting, simple concept classes are hard to learn (under the usual assumption that NP $\neq$ RP). For instance, monomials are not efficiently agnostic PAC learnable using arbitrary disjunctions of conjunctions (Kearns et al. 1994b), or halfspaces (Feldman et al. 2012).

In a nutshell, concept learning is an area of stark contrast from the viewpoint of runtime complexity. On the one hand, realizable concept learning is computationally easy for relatively simple classes, but remains difficult for more expressive hypothesis classes. On the other hand, the more "realistic" problem of agnostic concept learning proves to be very hard, even for simple hypothesis classes.

5.4 *Bagging and Boosting*

As expressive models are difficult to learn, what about learning simple models and combining them together, in order to produce more accurate predictors? Bagging and boosting are two techniques which grew out of this pragmatic question and became very practical tools for solving complex learning problems. The basic idea underlying these techniques is to amplify the accuracy of *weak learners*. One can think of a weak learner as an algorithm that uses a simple heuristic or "rule of thumb" in order to produce a hypothesis whose performance is just slightly better than a pure random guess. If such a weak learner can be implemented efficiently, then bagging and boosting may be used to iteratively combine weak hypotheses in order to produce a gradually better predictor. In what follows, we assume that $\mathscr{H}$ is closed under linear combinations, in order to produce model ensembles.

Introduced by Breiman (1996), the boostrap aggregating technique, abbreviated as *bagging*, aims at creating diverse weak hypotheses on different random samples of the training set S. As explained in Algorithm 1, These samples are taken uniformly with replacement, and a simple averaging of weak hypotheses is used to produce the final predictor. Bagging is particularly useful for learning combinations of decision trees, trained with weak learners such as ID3 (Quinlan 1986) or C4.5 (Quinlan

1993, 1996). When applied to tree models, bagging is often coupled with another idea, referred to as *subspace sampling*: at each iteration $t \in [T]$, select uniformly at random $n' \leq n$ features from $\mathscr{X}$ and train the weak learner A (without pruning) on the sample S'_t formed by the projection of S_t onto $[n']$. This encourages the diversity in the ensemble of weak hypotheses, and contributes to reduce the runtime of learning. The resulting method, called *random forests* (Breiman 2001), is easily parallelizable, and its performance in binary classification is comparable to that of Support Vector Machines (Caruana et al. 2008).

The algorithmic paradigm of *boosting*, studied by Schapire (1990), consists in gradually training diverse weak hypotheses by increasing the weight of previously misclassified examples. This paradigm gave rise to a practically useful algorithm, called AdaBoost (Freund and Schapire 1997), which is described in Algorithm 2. For convenience, the set of labels is here given by $\mathscr{Y} = \{-1, +1\}$. The AdaBoost algorithm maintains a probability distribution $\boldsymbol{p}_t$ over the training instances in S. Namely, on each round t, AdaBoost starts by training the weak learner A on the weighted dataset $(S_t, \boldsymbol{p}_t) = \{(\boldsymbol{x}_1, y_1, p_{t,1}), \ldots, (\boldsymbol{x}_m, y_m, p_{t,m})\}$. Next, the ensemble learner chooses a weight w_t for the weak hypothesis h_t, and then, updates the distribution $\boldsymbol{p}_t$ in a multiplicative way, where Z_t is the partition constant. A common choice for w_t is

$$w_t = \frac{1}{2} \ln \left(\frac{1}{\varepsilon_t} - 1 \right) \quad \text{where} \quad \varepsilon_t = \sum_{i \in [m], h_t(\boldsymbol{x}_i) = y_i} p_{i,t}$$

Algorithm 1: Bagging

```
input: data set $S \in \mathscr{X}^m$, number of rounds $T$, weak learner $A : \mathscr{X}^m \to \mathscr{H}$
for $t = 1$ to $T$ do
    build a sample $S_t$ from $S$ by drawing $m$ instances with replacement
    run $A$ on $S_t$ to find a concept $h_t$ in $\mathscr{H}$
end
output: $h(\boldsymbol{x}) = \frac{1}{T} \sum_{t=1}^{T} h_t(\boldsymbol{x})$
```

Algorithm 2: Boosting (AdaBoost)

```
input: data set $S \in \mathscr{X}^m$, number of rounds $T$, weak learner $A : \mathscr{X}^m \to \mathscr{H}$
initialize: set $p_{t,i} = \frac{1}{m}$ for each $i \in [m]$
for $t = 1$ to $T$ do
    run $A$ on $(S_t, \boldsymbol{p}_t)$ to find a concept $h_t$ in $\mathscr{H}$
    choose $w_t$
    set $p_{t+1,i} = \frac{1}{Z_t} p_{t,i} \exp(-w_t h_t(\boldsymbol{x}_i))$
end
output: $h(\boldsymbol{x}) = \text{sign} \sum_{t=1}^{T} w_t h_t(\boldsymbol{x})$
```

The AdaBoost algorithm benefits from a solid theoretical analysis, surveyed in Schapire and Freund (2012). As a well-known result, let $\gamma \in (0, 1)$, and suppose that at each iteration of AdaBoost, the weak learner returns a hypothesis for which $\varepsilon_t \leq 1/2 - \gamma$. Then, the training error of the final hypothesis h returned by AdaBoost after T iterations is at most:

$$L_S(h) \leq \exp(-2\gamma^2 T)$$

From a practical viewpoint, the AdaBoost algorithm has been successfully applied to face recognition tasks, using axis-aligned rectangles for weak hypotheses (Viola and Jones 2001). Moreover, the boosting technique is particularly suited for learning linear combinations of decision rules (Cohen and Singer 1999; Schapire and Singer 1999), and alternating decision trees (Freund and Mason 1999).

Finally, it is important to emphasize that bagging and boosting are not limited to binary classification tasks. Notably, bagging and random forests have been applied to regression, density estimation, and manifold learning; a detailed survey can be found in Criminisi et al. (2012). The boosting technique has been extended to multi-class learning and ranking; see again (Schapire and Freund 2012) for a comprehensive survey about this paradigm.

6 Convex Learning

Convex learning problems cover a wide variety of learning tasks, where the hypothesis class is a convex set and the loss function is convex. Many, if not most, statistical learning problems which are easy to solve fall into this category. In this section, we first introduce some mathematical background about convex learning problems, next we examine several well-known algorithms for solving these problems, and then, we briefly survey the topic of Support Vector Machines which heavily relies on convex learning techniques.

6.1 Convex Learning Problems

Let $\mathscr{W}$ be a subset of an Euclidean space or, more generally, a Hilbert space. The, $\mathscr{W}$ is convex if for any two points $\boldsymbol{u}, \boldsymbol{w} \in \mathscr{W}$, and any scalar $\lambda \in (0, 1)$, the point formed by the convex combination $\lambda \boldsymbol{u} + (1 - \lambda)\boldsymbol{w}$ belongs to $\mathscr{W}$. By extension, a function $f : \mathscr{W} \rightarrow \mathbb{R}$ is convex if its epigraph $\{(\boldsymbol{w}, v) \mid v \geq f(\boldsymbol{w})\}$ is a convex set. For the sake of simplicity, we shall consider in this section that every convex function is differentiable, but most results can be extended to non-differentiable functions, using the notion of sub-differential (Hiriart-Urrut and Lemaréchal 2004; Rockafellar 1970). A real-valued, differentiable function f on $\mathscr{W}$ is convex if and only if, for any $\boldsymbol{u}, \boldsymbol{w} \in \mathscr{W}$,

$$f(\boldsymbol{u}) - f(\boldsymbol{w}) \geq \langle \nabla f(\boldsymbol{w}), \boldsymbol{u} - \boldsymbol{w} \rangle$$

Families of convex learning problems are typically characterized in terms of three basic properties about convex objectives. Namely, given a convex set $\mathscr{W}$ and three positive scalars ρ, α, and β, a convex function $f : \mathscr{W} \to \mathbb{R}$ is

- ρ-*Lipschitz* if for any $\boldsymbol{u}, \boldsymbol{w} \in \mathscr{W}$,

$$|f(\boldsymbol{u}) - f(\boldsymbol{w})| \leq \rho \, \|\boldsymbol{u} - \boldsymbol{w}\|$$

- α-*strongly convex* if for any $\boldsymbol{u}, \boldsymbol{w} \in \mathscr{W}$,

$$|f(\boldsymbol{u}) - f(\boldsymbol{w})| \geq \langle \nabla f(\boldsymbol{w}), \boldsymbol{u} - \boldsymbol{w} \rangle + \frac{\alpha}{2} \, \|\boldsymbol{u} - \boldsymbol{w}\|^2$$

- β-*smooth* if for any $\boldsymbol{u}, \boldsymbol{w} \in \mathscr{W}$,

$$|f(\boldsymbol{u}) - f(\boldsymbol{w})| \leq \langle \nabla f(\boldsymbol{w}), \boldsymbol{u} - \boldsymbol{w} \rangle + \frac{\beta}{2} \, \|\boldsymbol{u} - \boldsymbol{w}\|^2$$

Furthermore, given a positive scalar $B > 0$, we say that a convex set $\mathscr{W}$ is B-*bounded* if $\|\boldsymbol{w}\| \leq B$ for all $\boldsymbol{w} \in \mathscr{W}$.

Informally, the Lipschitzness property indicates that f cannot change too fast. A sufficient condition for this condition is that $\|\nabla f(\boldsymbol{w})\| \leq \rho$ for every $\boldsymbol{w} \in \mathscr{W}$. The properties of smoothness and strong convexity are related to the curvature of f. Notably, if f is twice-differentiable, then f is β-smooth and α-strongly convex if and only if, for every $\boldsymbol{w} \in \mathscr{W}$, we have:

$$\alpha \boldsymbol{I} \preceq \nabla^2 f(\boldsymbol{w}) \preceq \beta \boldsymbol{I}$$

where $\boldsymbol{A} \preceq \boldsymbol{B}$ denotes the fact that $\boldsymbol{A} - \boldsymbol{B}$ is positive semi-definite. In other words, the scalars α and β can be viewed as bounds on the eigenvalues of f. The ratio α/β is often referred to as the *condition number* of f.

Definition 7 (*Convex Learning*) Let $\mathscr{Z}$ be an instance space, $\mathscr{H}$ be a hypothesis class over $\mathscr{Z}$, and $\ell : \mathscr{H} \times \mathscr{Z} \to \mathbb{R}$ be a loss function. Then, $(\mathscr{Z}, \mathscr{H}, \ell)$ is a *convex learning problem* if $\mathscr{H}$ is representable by a convex set $\mathscr{W}$, and for every $z \in \mathscr{Z}$, the function $f : \mathscr{W} \to \mathbb{R}$ given by $f(\boldsymbol{w}) = \ell(h_{\boldsymbol{w}}, z)$ is convex.

For convex learning problems we shall replace the hypothesis class $\mathscr{H}$ by its convex representation class $\mathscr{W}$, and rewrite the loss function ℓ as a mapping from $\mathscr{W} \times \mathscr{Z}$. Based on the aforementioned properties about convex objectives, convex learning problems may be declined into several categories, depending on whether the loss function is Lipschitz, smooth, or strongly convex on its first argument. For example, consider the binary classification task defined over an instance space $\mathscr{Z} = \mathscr{X} \times \{-1, +1\}$, a convex representation class $\mathscr{W}$, and the hinge loss function:

$$\ell(\boldsymbol{w}, (\boldsymbol{x}, y)) = \max(0, 1 - y \langle \boldsymbol{w}, \boldsymbol{x} \rangle)$$

If the domain set is the ball $\mathscr{X} = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| \leq \rho\}$, then ℓ is both convex and ρ-Lipschitz. Now, if we use the same domain set, but replace the above loss function with the regularized hinge loss function:

$$\ell(\boldsymbol{w}, (\boldsymbol{x}, y)) = \max(0, 1 - y \langle \boldsymbol{w}, \boldsymbol{x} \rangle) + \frac{\alpha}{2} \|\boldsymbol{w}\|^2$$

it follows that ℓ is both α-strongly convex and ρ-Lipschitz. As another example, consider the regression task defined over an instance space $\mathscr{Z} = \mathscr{X} \times \mathbb{R}$, a convex representation class $\mathscr{W}$, and the square loss function

$$\ell(\boldsymbol{w}, (\boldsymbol{x}, y)) = (y - \langle \boldsymbol{w}, \boldsymbol{x} \rangle)^2$$

If the domain set is the ball $\mathscr{X} = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| \leq \beta/2\}$, then ℓ is both convex and β-smooth.

In general, a convex learning problem can be formulated as a *stochastic convex optimization* task of the form:

$$\begin{aligned} &\text{minimize} \quad L_D(\boldsymbol{w}) = \mathbb{E}_{z \sim \mathscr{D}}[\ell(\boldsymbol{w}, z)] \\ &\text{subject to} \quad \boldsymbol{w} \in \mathscr{W} \end{aligned} \tag{14}$$

where $\mathscr{W}$ is a convex set, and ℓ is convex on its first argument. We may attempt to solve this problem in a direct way, using a stochastic convex optimization algorithm that calls the example oracle EX($\mathscr{D}$) for approximating the unknown objective L_D. Alternatively, we may rely on an indirect approach, by using learning rules defined over the empirical risk L_S, and described in Sect. 4. In the convex setting, Regularized Risk Minimization RRM is the paradigm of choice. Recall here that the RRM rule finds a minimizer of

$$\frac{1}{m} \sum_{i=1}^{m} \ell(\boldsymbol{w}, z_i) + \text{reg}(\boldsymbol{w})$$

subject to $\boldsymbol{w} \in \mathscr{W}$, where reg : $\mathscr{W} \to \mathbb{R}_+$ is a regularization function. Namely, the next result established in Shalev-Shwartz et al. (2010), Shalev-Shwartz and Ben-David (2014) indicates that the RRM rule is stable for various families of convex learning problems.

Theorem 7 (Stability of RRM) *Let $(\mathscr{Z}, \mathscr{W}, \ell)$ be a convex learning problem. Then,*

- *the* RRM *rule with the Tikhonov regularizer* $\text{reg}(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|^2$ *is stable with rate $O(1/m)$, whenever ℓ is ρ-Lipschitz or β-smooth;*
- *the* ERM *rule (i.e.* RRM *with no regularizer) is stable with rate $O(1/m)$, whenever ℓ is ρ-Lipschitz and α-strongly convex.*

6.2 Convex Learning Algorithms

In the rich literature of convex optimization, a wide variety of algorithms have been devised for solving convex learning problems in a computationally efficient way. We invite the reader in browsing excellent textbooks about this active research topic (Bertsekas 2015; Boyd and Vandenberghe 2004; Bubeck 2015; Kushner and Yin 2010; Nesterov 2004; Nemirovski 1995; Sra et al. 2012). Here, we shall focus on three, well-studied convex learning algorithms: *Stochastic Gradient Descent* (SGD), *Stochastic Coodinate Descent* (SCD), and *Conditional Gradient* (CG).

6.2.1 Stochastic Gradient Descent

Arguably, the *Gradient Descent* algorithm is one of the oldest strategy for solving convex optimization problems (Cauchy 1847). The overall idea of this iterative optimization algorithm is to improve the solution at each iteration, by taking a step along the negative of the gradient of the function to be minimized at the current point. The stochastic version of this algorithm, which dates back to Robbins and Monro (1951), aims at minimizing a stochastic convex objective function of the form $L_D(\boldsymbol{w})$. To this end, SGD takes at each iteration a step along a random direction, for which the expectation is the negative of the gradient. As most convex learning problems are defined over a restricted subset $\mathscr{W}$ of an Euclidean or Hilbert space, the adaptation of SGD to statistical learning typically involves an additional *projection step*, which maintains the current point in the set of feasible solutions $\mathscr{W}$.

Algorithm 3: Stochastic Gradient Descent

```
input: scalar $\eta$, integer $m$
initialize: $\boldsymbol{v}_1 = \mathbf{0}$
for $t = 1$ to $m$ do
    $\boldsymbol{w}_t = argmin_{\boldsymbol{w} \in \mathscr{W}} \; \|\boldsymbol{w} - \boldsymbol{v}_t\|^2$
    $z_t = \text{EX}(\mathscr{D})$
    $\boldsymbol{v}_{t+1} = \boldsymbol{w}_t - \eta \nabla \ell(\boldsymbol{w}_t, z_t)$
end
output: $\boldsymbol{w} = \frac{1}{m} \sum_{t=1}^{m} \boldsymbol{w}_t$
```

The resulting projected SGD method is described in Algorithm 3. At each iteration t, the algorithm first projects the current point $\boldsymbol{v}_t$ onto the representation class $\mathscr{W}$, next calls the example oracle for an instance $z_t \in \mathscr{Z}$, and then performs a descent step using the gradient of the loss $\ell(\boldsymbol{w}_t, z_t)$. The convergence of SGD has been analyzed for various families of objective functions (Kushner and Yin 2010; Rakhlin et al. 2012; Shalev-Shwartz et al. 2009; Shalev-Shwartz and Ben-David 2014). The next theorem summarizes convergence results obtained for the three aforementioned families.

Theorem 8 (Convergence of SGD) *Let $(\mathscr{Z}, \mathscr{W}, \ell)$ be a convex learning problem. Then, the SGD algorithm is*

- *universally consistent with rate $O(1/\sqrt{m})$ if $\mathscr{W}$ is B-bounded, and ℓ is ρ-Lipschitz or β-smooth.*
- *universally consistent with rate $\tilde{O}(1/m)$ if $\mathscr{W}$ is B-bounded, and ℓ is both ρ-Lipschitz and α-strongly convex.*

In other words, stochastic convex optimization problems of the form (14) can be solved directly, using the SGD algorithm, under reasonable assumptions about the representation class $\mathscr{W}$ and the loss function ℓ. The choice of the learning parameter η is governed by the input parameters defining the family of convex learning problems. For example, if $\mathscr{W}$ is B-bounded and ℓ is ρ-Lipschitz then, using $\eta = B/\rho\sqrt{m}$, the convergence rate is bounded by $B\rho/\sqrt{m}$. Thus, given a desired accuracy ε, it suffices to run SGD $m \geq (B\rho/\varepsilon)^2$ iterations in order to achieve, in expectation, a risk that is ε-close to the smallest risk.

The Gradient Descent method and its stochastic variant belong to the larger family of *Mirror Descent* algorithms (Nemirovski and Yudin 1983; Beck and Teboulle 2003), used to solve regularized risk minimization tasks for various regularization functions. The overall idea is to first map the current point $\boldsymbol{w}_t \in \mathscr{W}$ into the dual space $\mathscr{W}^*$, next perform the gradient descent in the dual space, and then mapping back the resulting point into the primal space. Various instances of Mirror Descent schemes include the *Exponentiated Gradient* algorithm (Kivinen and Warmuth 1997), and the *p-norm* algorithms (Gentile 2003). One of key geometric properties of Mirror Descent schemes is that an objective function f over the primal space $\mathscr{W}$ is α-strongly convex with respect to a norm $\|\cdot\|$ if and only if its conjugate f^* on the dual space $\mathscr{W}^*$ is $1/\alpha$-strongly smooth with respect to the dual norm $\|\cdot\|^*$ (Hiriart-Urrut and Lemaréchal 2004; Kakade et al. 2012). This, together with standard properties of Bregman divergences, typically yield convergence rates in $O(1/\sqrt{m})$ or $\tilde{O}(1/m)$ which depend only logarithmically in the dimension n of the data instances.

Algorithm 4: Stochastic Coordinate Descent

```
input: convex objective $L_S(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \ell(\boldsymbol{w}, z_i)$
initialize: $\boldsymbol{w}_1 = \boldsymbol{0}$
for $t = 1$ to $T$ do
    Choose index $j$ uniformly at random in $[n]$
    Choose stepsize $\eta_t$
    $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t |\frac{\partial L_S(\boldsymbol{w}_t)}{\partial j}| \boldsymbol{e}_j$
end
output: $\boldsymbol{w} = \boldsymbol{w}_T$
```

From a computational viewpoint, the main bottleneck of the SGD algorithm, and more generally Mirror Descent algorithms, lies in the projection step, which is a constrained convex optimization task performed *at each iteration*. For simple representation classes $\mathcal{W}$, such as balls, hypercubes, simplices, and permutahedra, fast projection methods have been proposed (Duchi et al. 2008; Krichene et al. 2015; Lim and Wright 2016). However, for more complex representation classes, such as polyhedra described by linear inequalities, the projection step has to rely on general, time-consuming, convex optimization techniques. Circumventing this bottleneck by limiting the number of projection steps in gradient descent algorithms is a subject of ongoing research (Mahdavi et al. 2012; Zhang et al. 2013).

6.2.2 Stochastic Coordinate Descent

When the hypothesis class is a simple convex object, characterized by separable or block-separable constraints, the empirical risk can be minimized using the family of *Coordinate Descent* algorithms (Censor and Zenios 1997; Tseng and Yun 2009; Nesterov 2012; Wright 2015). Such methods, inspired from the Gauss-Seidel method for systems of linear equations, solve convex optimization tasks by iteratively performing approximate minimization along coordinate directions.

Algorithm 4 describes a stochastic version of Coordinate Descent for minimizing the empirical risk in the unconstrained setting (i.e. $\mathcal{W} = \mathbb{R}^n$). During each iteration t, the SCD algorithm first selects a coordinate j uniformly at random, and independently of past rounds, and then performs a descent according to the derivative of the empirical risk $L_S(\boldsymbol{w}_t)$ of the current point $\boldsymbol{w}_t$ at coordinate j. As detailed for instance in Wright (2015), the SCD algorithm may be easily upgraded to constrained versions of this task, using block-separable constraints.

Algorithm 5: Conditional Gradient

```
input: convex objective $L_S(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \ell(\boldsymbol{w}, \boldsymbol{z}_i)$
initialize: $\boldsymbol{w}_1$ is an arbitrary point in $\mathcal{W}$
for $t = 1$ to $T$ do
    $\boldsymbol{v}_t = argmin_{\boldsymbol{v}\in\mathcal{W}} \langle \nabla L_S(\boldsymbol{w}_t), \boldsymbol{v}\rangle$
    Choose stepsize $\eta_t \in (0, 1)$
    $\boldsymbol{w}_{t+1} = (1 - \eta_t)\boldsymbol{w}_t + \eta_t \boldsymbol{v}_t$
end
output: $\boldsymbol{w} = \boldsymbol{w}_T$
```

Although SCD is a fast, easy-to-implement algorithm, its convergence analysis requires more sophisticated conditions on the feasible set and the objective function (Nesterov 2012; Lu and Xiao 2015; Wright 2015). Notably, if L_S satisfies the property of coordinate-wise Lipschitz continuity with constants $\{\beta_j\}_{j=1}^{n}$, and the diameter of $\mathcal{W}$ with respect to the norm

$$\|\boldsymbol{w}\|_\beta = \sqrt{\sum_{j=1}^{n} \beta_j w_j^2}$$

is bounded by a constant R, then SCD converges to the empirical risk minimizer with rate in $O(1/T)$. Better convergence bounds may be achieved for strongly convex loss functions, or using accelerated versions of SCD.

6.2.3 Conditional Gradient

For hypothesis classes characterized by complex geometric objects, such as cones or polyhedra, convex projection tasks may be computationally demanding. Yet, *linear optimization* tasks on those objects are typically much easier. *Projection-free* algorithms constitute a family of convex optimization methods which replace the convex projection step with a cheaper linear optimization step (Clarkson 2010; Hazan and Kale 2012; Jaggi 2013; Lacoste-Julien and Jaggi 2015; Freund and Grigas 2016; Garber and Hazan 2016; Garber and Meshi 2016). The prototypical algorithm in this family is the *Conditional Gradient* method, due to Franck and Wolfe (1956).

Algorithm 5 describes a simple version of CG. During each iteration t, the algorithm starts by performing a linear optimization step using the gradient of the empirical risk of the current point $\boldsymbol{w}_t$, and then updates its solution according to a convex combination of $\boldsymbol{w}_t$ and the linear minimizer $\boldsymbol{v}_t$. Different strategies for choosing the stepsize η_t at each iteration are reported in Jaggi (2013), Freund and Grigas (2016). Apart from the choice of η_t, CG algorithms essentially differ in the linear optimization step. For example, a *local* linear optimization step is suggested in Garber and Hazan (2016), while *step-away* strategies are advocated in Lacoste-Julien and Jaggi (2015), Garber and Meshi (2016).

Overall, the performance of CG is relatively similar to the performance of SGD (for minimizing the empirical risk), using only linear optimization steps. Specifically, the convergence rate of CG is in

- $O(1/\sqrt{T})$ if $\mathscr{W}$ is B-bounded, and L_S is ρ-Lipschitz,
- $\tilde{O}(1/T)$ if $\mathscr{W}$ is B-bounded, and L_S is both ρ-Lipschitz and α-strongly convex.

We mention in passing that the SGD, SCD, and SG algorithms enjoy convergence rates in $O(\exp(-T))$ when the objective function is both smooth and strongly convex (Bubeck 2015).

6.3 Support Vector Machines

As mentioned in the introduction of this section, convex learning problems constitute the most important family of statistical learning problems where efficient learnability results can be obtained. It is therefore not surprising that convex learning algorithms have been successfully applied to a wide range of statistical learning tasks. In particular, the key tools for handling high-dimensional learning tasks are *Support Vector Machines* (SVMs). Introduced in Boser et al. (1992), SVMs have been a subject of extensive research, both from a theoretical and practical perspective, summarized in various textbooks (Vapnik 1998; Cristianini and Shawe-Taylor 2000; Schölkopf and Smola 2002; Steinwart and Christmann 2008).

Support Vector Machines are defined through two main notions: *margins* and *kernels*. Intuitively, the notion of margin is related to the sample complexity of learning: SVMs handle high-dimensional hypothesis classes by searching for large margin separators. A linear classifier separates a training set with a large margin if it does not only classify examples in a correct way, but also pushes those examples away from the separating hyperplane. Thus, a large margin classifier may require a small sample complexity, even if the dimensionality of the feature space is high, or even infinite. The notion of kernel is related to the runtime complexity of learning. Basically, a kernel is a similarity measure between instances, which can be characterized as an inner product in some Hilbert space. For classifiers involving a feature expansion mapping, the "kernel trick" enables a computationally efficient implementation of learning, without explicitly handling the high dimensional feature expansion vector. Of course, the notions of margins and kernels are not limited to binary classification: they have been extended to various learning task including, for example, multi-class prediction and structured prediction.

There are two main categories of SVMs, depending on whether the training set supplied to the learner is assumed to be separable, or not. For the sake of simplicity, we focus here on zero-threshold linear functions, but the SVM rules defined below can easily be extended to non-homogeneous linear functions, using data points in the extended domain set $\mathscr{X} \times \{1\}$. A training set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ is linearly separable if there exists a vector $\boldsymbol{w}$ such that $y_i = \text{sign}\,\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$ for all $i \in [m]$. In this separable case, the margin of the hyperplane $\boldsymbol{w}$ with respect to the training set S is the minimal distance between an example in S and the hyperplane. In particular, if $\|\boldsymbol{w}\| = 1$, then the distance between $\boldsymbol{w}$ and any example $(\boldsymbol{x}_i, y_i)$ is simply given by $y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$. Therefore, the *Hard*-SVM rule is to find a separating hyperplane $\boldsymbol{w}$ with $\|\boldsymbol{w}\| = 1$ that maximizes the distance $\min_{i \in [m]} y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$. The Hard-SVM rule may be formulated in an equivalent way by the (constrained) convex optimization task:

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{w}\|^2 \\ \text{subject to} \quad & y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1 \;\; \forall i \in [m] \end{aligned} \tag{15}$$

If the training set S is linearly separable, then this optimization task is feasible. In this case, the solution $\boldsymbol{w}$ is normalized by $\|\boldsymbol{w}\|$ to yield the final predictor.

In the more general case where S is not linearly separable, the formulation (15) can be relaxed by allowing separability constraints to be violated by some examples. As usual, this may be formulated by adding slack variables $\xi_1, \ldots, \xi_m$, where each ξ_i captures by how much the the constraint $y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1$ is violated. The resulting *Soft*-SVM rule jointly minimizes the margin and the violations of separability constraints, using the following optimization task:

$$\begin{aligned}&\text{minimize} \;\; \lambda \left\|\boldsymbol{w}\right\|^2 + \frac{1}{m}\sum_{i=1}^{m}\xi_i \\ &\text{subject to} \;\; y_i \left\langle \boldsymbol{w}, \boldsymbol{x}_i \right\rangle \geq 1 - \xi_i \;\; \forall i \in [m]\end{aligned} \tag{16}$$

To this point, recall that the hinge loss between a linear model $\boldsymbol{w}$ and an example $(\boldsymbol{x}, y)$ is given by $\ell(\boldsymbol{w}, (\boldsymbol{x}, y)) = \max\{0, 1 - y\left\langle \boldsymbol{w}, \boldsymbol{x}\right\rangle\}$. With this formulation in hand, the Soft-SVM rule (16) can be expressed as a standard RRM task, given by

$$\min_{\boldsymbol{w},b}\left(\sum_{i=1}^{m}\ell((\boldsymbol{w}, b), (\boldsymbol{x}_i, y_i)) + \lambda\left\|\boldsymbol{w}\right\|^2\right) \tag{17}$$

This RRM objective is referred to as the *primal* formulation of the Soft-SVM rule. Since we are dealing with a convex optimization task, the Soft-SVM rule admits an equivalent *dual* formulation, where the optimal solution is characterized by a linear combination of examples in S, using Lagrangian variables $\alpha_1, \ldots, \alpha_m$:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m, \boldsymbol{\alpha}\geq\boldsymbol{0}}\left(\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \left\langle \boldsymbol{x}_i, \boldsymbol{x}_j\right\rangle\right) \tag{18}$$

and the correspondence between (17) and (18) is given by $\boldsymbol{w} = \sum_{i=1}^{m}\alpha_i\boldsymbol{x}_i$. If $\boldsymbol{w}$ is an optimal solution of (17) then the data points $\boldsymbol{x}_i$ for which α_i is positive are called the *support vectors* of $\boldsymbol{w}$.

Based on the above formulations, various convex optimization algorithms can be exploited for implementing linear Soft-SVMs. For example, (Shalev-Shwartz et al. 2007) solve the primal problem (17) using Stochastic Gradient Descent, and (Hsieh et al. 2008) solve the dual problem (18) using (dual) Coordinate Descent. The Conditional Gradient algorithm was also advocated for solving structured prediction tasks with SVMs (Lacoste-Julien et al. 2013).

Since the expressive power of linear functions is limited, a natural approach for extending SVMs to non-linear functions is to use a feature expander, that is, an embedding $\boldsymbol{\phi}$ of the domain set $\mathscr{X}$ onto some (possibly infinite dimensional) Hilbert space $\mathscr{F}$. Based on this feature expander, the hypothesis class $\mathscr{H}$ is represented by the set of vectors $\boldsymbol{w}$ such that $h_{\boldsymbol{w}}(\boldsymbol{x}) = \text{sign}(\langle\boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x})\rangle)$. Given an embedding $\boldsymbol{\phi}$, the corresponding *Kernel operator* is defined as

$$K(\boldsymbol{x}, \boldsymbol{x}') = \left\langle\phi(\boldsymbol{x}), \phi(\boldsymbol{x}')\right\rangle$$

and the dual formulation (18) of Soft-SVM can be rewritten using the "kernel trick":

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m, \boldsymbol{\alpha}\geq\boldsymbol{0}}\left(\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)\right) \tag{19}$$

By the Kernel Representer Theorem (Schölkopf et al. 2001), the optimal solution $\boldsymbol{w}$ can be expressed as a linear combination of expanded points, that is, $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{\phi}(x_i)$. Since the dimension of $\boldsymbol{w}$ can be large or infinite, the kernel trick allows us to efficiently encode $\boldsymbol{w}$ as a set of support vectors x_i, each associated with its coefficient α_i. Furthermore, since the kernel operator K associated with a feature expander $\boldsymbol{\phi}$ defines a positive semidefinite matrix, the kernelized SVM rule (19) is a concave optimization problem. Again, convex optimization algorithms can be advocated for efficiently solving this task, provided that the kernel operator K can be computed in polynomial time. Various kernels satisfying this condition have been proposed in the literature, and we refer the reader to Herbrich (2002), Schölkopf and Smola (2002), Shawe-Taylor and Cristianini (2004), Bottou (2007), Kung (2014), for detailed surveys about kernel methods.

Finally, as kernels provide a way to express prior knowledge about the learning task at hand, an important subject of ongoing research in SVMs is to *learn kernels*, using a kernel family (Lanckriet et al. 2004; Bach 2008; Cortes et al. 2009, 2010).

7 Conclusion

In this chapter, we have drawn a conceptual map of statistical computational learning by providing answers to several questions: what is a statistical learning problem? How to measure the performance of a learning algorithm? Which are the main optimization principles in statistical learning? And, under which conditions a hypothesis class is learnable? Based on these foundations, we have surveyed two important problems in Machine Learning: concept learning and convex learning. In this concluding section, we highlight several topics of research at the intersection of statistical computational learning and AI. Due to space reasons, the list is by no means exhaustive, and we apologize for omitting other topics of interest.

Learning Sparse Models The concept of sparsity is ubiquitous in many scientific and engineering applications, for identifying parsimonious solutions to high-dimensional problems. Informally, a sparse solution can be viewed as a high-dimensional vector or matrix satisfying some *sparsity constraint*, which limits the degrees of freedom of the model. Various sparsity constraints have been proposed in the literature of machine learning and signal processing, ranging from the standard cardinality constraint that restrains the number of nonzero coordinates (Shalev-Shwartz et al. 2010), to more sophisticated sparsity constraints which impose a low-dimensional structure on the set of nonzero features (Hegde et al. 2015; Jain et al. 2016). For example, in the "group-structured" sparsity constraint, the relevant features are partitioned into a small number of contiguous blocks, and in the "tree-structured" sparsity constraint, such features are arranged into a connected acyclic graph. As convex optimization under sparsity constraints is NP-hard (Natarajan 1995), two main approaches have been advocated for learning sparse models: convex relaxation (Shalev-Shwartz and Tewari 2011; Bach et al. 2012), and approximation algorithms (Bahmani et al. 2013; Jain et al. 2014). A recent survey on sparse modelling and learning can be found in Rish and Grabarnik (2014).

Learning Probabilistic Models In statistical learning, probabilistic models aim at estimating the hidden distribution that generates data instances. Of particular interest in AI are probabilistic graphical models which are able to represent high-dimensional probability distributions (Koller and Friedman 2009; Murphy 2012). As explained in Sect. 2, a probabilistic graphical model is a pair $(G, \boldsymbol{\theta})$, where G is a graphical structure and $\boldsymbol{\theta}$ is a vectorized set of parameters. *Parameter learning* is the task of estimating from data the parameters of a probabilistic model, when the structure is fixed. Correspondingly, *structure learning* is the problem of extracting the graphical structure of a probabilistic model, given a class of candidate structures, such as directed acyclic graphs for Bayesian networks, or hypertrees for bounded-treewidth Markov networks. Various algorithms have been proposed for estimating parameters under the (possibly regularized) log-likelihood loss function. In particular, *Expectation Minimization* (EM) (Dempster et al. 1977) is the prototypical algorithm for estimating parameters in presence of missing values (Lauritzen 1995). A comprehensive treatment of the subject is given in Little and Rubin (2014).

Structure learning is arguably more challenging, since the corresponding regularized risk minimization task involves combinatorial constraints capturing admissible graphical structures. Although structure learning is tractable for tree-directed models (Chow and Liu 1968) and their mixtures (Meila and Jaakkola 2006), the problem is NP-hard for more expressive models, such as Bayesian networks (Chickering 1996), Bayesian polytrees (Dasgupta 1999), and bounded-treewidth Markov networks (Srebro 2003). For this reason, structure learning is an active research topic relying on both statistical and combinatorial methods. Notably, (Cussens 2011; Kumar and Bach 2013; Nie et al. 2014; Bartlett and Cussens 2017) use Integer Linear Programming techniques for learning the structure of Bayesian networks or Markov networks with bounded-treewidth. SAT and CSP techniques have also been proposed for solving these structure learning problems (Cussens 2008; Berg et al. 2014; van Beek and Hoffmann 2015).

Learning Preference Models The spectrum of applications that resort on the ability to learn preferences is extremely wide, ranging from configuration softwares and recommender systems to information retrieval and group decision-making (see e.g. chapter "Compact Representation of Preferences" of Volume 1). It is therefore not surprising that topic of preference learning has gained a considerable interest in statistical and computational learning. As explained in Sect. 2, preference learning problems can be divided into several categories, depending on the type of reference set, the type of preference relation, the examples provided to the learner and, of course, the class of preference models.

In *label ranking* (Vembu and Gärtner 2010), the problem is to associate instances with a total order of predefined labels. With each training instance, we receive supervision given as a binary relation on the labels. More formally, the instance space is given $\mathscr{Z} = \mathscr{X} \times \mathscr{Y}$, where $\mathscr{X}$ is the domain set, and $\mathscr{Y}$ is the space of all directed acyclic graphs over the set of labels $[k]$. The goal is to learn from a training set S a hypothesis $h : \mathscr{X} \to \mathscr{Y}^\dagger$ in the available hypothesis class $\mathscr{H}$ that minimizes some loss function $\ell : \mathscr{H} \times \mathscr{Z} \to \mathbb{R}$. For total ranking tasks, $\mathscr{Y}^\dagger$ is the group of permutations over $[k]$, and for more general ranking tasks, $\mathscr{Y}^\dagger$ is a subset of $\mathscr{Y}$. Several families of label ranking problems can be solved by reduction to binary classification (Hüllermeier et al. 2008), boosting (Dekel et al. 2003), multi-label classification (Crammer and Singer 2003), ordinal regression (Herbrich et al. 2000), or regularized least-square minimization (Gärtner and Vembu 2009).

In *object ranking* (Kamishima et al. 2010), $\mathscr{X}$ is a set of objects, and $\mathscr{Y}$ is a space of total rankings (permutations) or partial rankings (DAGs) over $\mathscr{X}$. Each training instance is formed by a pair, or more generally a set, of objects in $\mathscr{X}$, and the supervision is given by a preference ordering on these objects. The goal is to learn a hypothesis $h : \mathscr{X} \to \mathscr{Y}$, chosen from a class $\mathscr{H}$ that minimizes some loss function ℓ. Again, various statistical learning techniques have been successfully applied for solving tractable object ranking problems. They include, among others, boosting methods (Freund et al. 2003; Xu and Li 2007), and SVMs (Joachims 2002; Kazawa et al. 2005; Cao et al. 2006).

Ranking tasks are intrinsically related to preference aggregation problems. Notably, the problem of finding a total ranking of objects minimizing a pairwise loss function is generally NP-hard (Cohen et al. 1999; Alon 2006). The difficulty is even more accute when the hypothesis class is a Mallows model or an exponential family (Vembu et al. 2009; Lu and Boutilier 2014).

Learning Neural Models As mentioned in Sect. 2, neural models and Machine Learning have a long shared history, dating back to Rosenblatt's invention of the Perceptron algorithm (Rosenblatt 1958). Neural networks were extensively studied in the 1980s, but with mixed empirical results. During this past decade, a combination of algorithmic advances in Machine Learning, together with increasing computational power and data size, has led to a breakthrough in the effectiveness of deep neural networks (Goodfellow et al. 2016). In particular, the families of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown impressive performance on a variety of application domains, including computer vision (LeCun et al. 2010; Krizhevsky et al. 2012; Pinheiro and Collobert 2014), speech recognition (Hinton et al. 2012; Graves et al. 2013), and natural language processing (Collobert and Weston 2008; Cho et al. 2014; Kalchbrenner et al. 2014). In the present book, deep neural networks are discussed in chapter "Reinforcement Learning" of Volume 1, and chapter "Designing Algorithms for Machine Learning and Data Mining" of Volume 2.

Despite the undoubled practical success of deep learning, there are many open theoretical questions related to this fascinating subject of research. As discussed in Sect. 5, intersections of separating hyperplanes over $\{0, 1\}^n$ are not efficiently PAC learnable for the zero-one loss (Klivans and Sherstov 2009). This implies that no efficient algorithm can be found for training neural networks, even if we allow additional layers or effective activation functions. For other loss functions advocated in deep learning, the corresponding optimization task remains highly non-convex, and hence, generally intractable. So, there is a fundamental gap between the theory of statistical computational learning and the practical efficiency of deep learning, achieved by gradient-based methods with backpropagation (Rumelhart et al. 1986). Recent investigations in the theoretical analysis of deep models have attempted to bridge this gap (Kawaguchi 2016; Bach 2017; Kawaguchi et al. 2017; Shalev-Shwartz et al. 2017; Song et al. 2017; Zhang et al. 2017), but much remains to be done before having a comprehensive analysis of practical results.

References

Aggarwal C, Reddy C (2013) Data clustering: algorithms and applications. Taylor and Francis
Alekhnovich M, Braverman M, Feldman V, Klivans A, Pitassi T (2008) The complexity of properly learning simple concept classes. J Comput Syst Sci 74(1):16–34
Alon N (2006) Ranking tournaments. SIAM J Discret Math 20(1):137–142
Alon N, Ben-David S, Cesa-Bianchi N, Haussler D (1997) Scale-sensitive dimensions, uniform convergence, and learnability. J ACM (JACM) 44(4):615–631
Alpaydin E (2009) Introduction to machine learning. MIT, USA
Angluin D, Laird PD (1987) Learning from noisy examples. Mach Learn 2(4):343–370
Anthony M (2001) Discrete mathematics of neural networks: selected topics. SIAM monographs on. discrete mathematics and applications
Anthony M (2010) Probabilistic learning and boolean functions. In: Crama Y, Hammer P (eds) Boolean models and methods in mathematics, computer science, and engineering, encyclopedia of mathematics and its applications. Cambridge University, Cambridge, pp 197–220
Anthony M, Barlett P (1999) Neural network learning: theoretical foundations. Cambridge University, Cambridge
Anthony M, Biggs N (1997) Computational learning theory. Cambridge University, Cambridge
Anthony M, Shawe-Taylor J (1993) Using the perceptron algorithm to find consistent hypotheses. Comb, Probab Comput 2:385–387
Arora S, Babai L, Stern J, Sweedyk Z (1997) The hardness of approximate optima in lattices, codes, and systems of linear equations. J Comput Syst Sci 54(2):317–331
Bach FR (2008) Exploring large feature spaces with hierarchical multiple kernel learning. In: Advances in neural information processing systems 21 (NIPS 2008), pp 105–112
Bach FR (2017) Breaking the curse of dimensionality with convex neural networks. J Mach Learn Res 18:19:1–19:53
Bach FR, Jenatton R, Mairal J, Obozinski G (2012) Optimization with sparsity-inducing penalties. Found Trends Mach Learn 4(1):1–106
Bahmani S, Raj B, Boufounos P (2013) Greedy sparsity-constrained optimization. J Mach Learn Res 14:807–841
Bartlett M, Cussens J (2017) Integer linear programming for the bayesian network structure learning problem. Artif Intell 244:258–271
Beck A, Teboulle M (2003) Mirror descent and nonlinear projected subgradient methods for convex optimization. Oper Res Lett 31(3):167–175
Ben-David S, Eiron N, Long PM (2003) On the difficulty of approximately maximizing agreements. J Comput Syst Sci 66(3):496–514
Berg J, Järvisalo M, Malone B (2014) Learning optimal bounded treewidth bayesian networks via maximum satisfiability. In: Proceedings of the 17th international conference on artificial intelligence and statistics (AISTATS 2014), pp 86–95
Bertsekas D (2015) Convex optimization algorithms. MIT, USA
Birge J, Louveaux F (2011) Introduction to stochastic programming. Springer, Berlin
Bishop C (2006) Pattern recognition and machine learning. Springer, Berlin

Blum A (1992) Rank-r decision trees are a subclass of r-decision lists. Inf Process Lett 42(4): 183–185

Blum A, Rivest RL (1992) Training a 3-node neural network is NP-complete. Neural Netw 5(1): 117–127

Blumer A, Ehrenfeucht A, Haussler D, Warmuth M (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM (JACM) 36(4):929–965

Boser BE, Guyon L, Vapnik V (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the 5th annual ACM conference on computational learning theory (COLT 1992), pp 144–152

Bottou L (2007) Large-scale kernel machines, Neural information processing series. MIT, USA

Bousquet O, Elisseeff A (2002) Stability and generalization. J Mach Learn Res 2(Mar):499–526

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University, Cambridge

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Bubeck S (2015) Convex optimization: algorithms and complexity. Found Trends Mach Learn 8(3–4):231–358

Cao Y, Xu J, Liu T, Li H, Huang Y, Hon H (2006) Adapting ranking SVM to document retrieval. In: Proceedings of the 29th annual international ACM conference on research and development in information retrieval (SIGIR 2006), pp 186–193

Caruana R, Karampatziakis N, Yessenalina A (2008) An empirical evaluation of supervised learning in high dimensions. In: Proceedings of the 25th international conference on machine learning (ICML 2008), pp 96–103

Cauchy A (1847) Méthode générale pour la résolution des systèmes d'équations simultanées. C. R. Acad. Sci. Paris 25:536–538

Censor Y, Zenios SA (1997) Parallel optimization. Oxford University, Oxford

Chickering DM (1996) Learning Bayesian networks is NP-complete. In: Learning from data: artificial intelligence and statistics V, Springer, Berlin, pp 121–130

Cho K, van Merrienboer B, Gülçehre Ç, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP 2014), pp 1724–1734

Chow CK, Liu CN (1968) Approximating discrete probability distributions with dependence trees. IEEE Trans Inf Theory 14(3):462–467

Clarkson KL (2010) Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. ACM Trans Algorithms 6(4):63:1–63:30

Clémençon S, Vayatis N (2007) Ranking the best instances. J Mach Learn Res 8:2671–2699

Cohen W, Schapire R, Singer Y (1999) Learning to order things. J Artif Intell Res (JAIR) 10:243–270

Cohen WW, Singer Y (1999). A simple, fast, and effictive rule learner. In: Proceedings of the 16th national conference on artificial intelligence (AAAI 1999), pp 335–342

Collobert R, Weston J (2008) A unified architecture for natural language processing: deep neural networks with multitask learning. In: Proceedings of the twenty-fifth international conference in machine learning (ICML 2008), pp 160–167

Cortes C, Mohri M, Rostamizadeh A (2009) Learning non-linear combinations of kernels. In: Advances in neural information processing systems 22 (NIPS 2009), pp 396–404

Cortes C, Mohri M, Rostamizadeh A (2010) Generalization bounds for learning kernels. In: Proceedings of the 27th international conference on machine learning (ICML 2010), pp 247–254

Crammer K, Singer Y (2003) A family of additive online algorithms for category ranking. J Mach Learn Res 3:1025–1058

Criminisi A, Shotton J, Konukoglu E (2012) Decision forests: a unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. Found Trends Comput Graph Vis 7(2–3):81–227

Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other kernel-based learning methods. Cambridge University, Cambridge

Cussens J (2008) Bayesian network learning by compiling to weighted MAX-SAT. In: Proceedings of the 24th conference in uncertainty in artificial intelligence (UAI 2008), pp 105–112

Cussens J (2011) Bayesian network learning with cutting planes. In: Proceedings of the 27th conference on uncertainty in artificial intelligence (UAI 2011), pp 153–160

Daniely A, Shalev-Shwartz S (2016) Complexity theoretic limitations on learning dnf's. In: Proceedings of the 29th conference on learning theory (COLT 2016), pp 815–830

Darwiche A (2009) Modeling and reasoning with bayesian networks. Cambridge University, Cambridge

DasGupta A (2011) Probability for statistics and machine learning: fundamentals and. advanced topics. Springer, Berlin

Dasgupta S (1999) Learning polytrees. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence (UAI 1999), pp 134–141

De Raedt L (2008) Logical and relational learning. Springer, Berlin

Dekel O, Manning CD, Singer Y (2003) Log-linear models for label ranking. In: Advances in neural information processing systems 16 (NIPS 2003), pp 497–504

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J R Stat Society Ser B (Methodological) 39(1):1–38

Devroye L, Györfi L, Lugosi G (2013) A probabilistic theory of pattern recognition. Springer, Berlin

Du K-L, Swamy MNS (2013) Neural networks and statistical learning. Springer, Berlin

Duchi JC, Shalev-Shwartz S, Singer Y, Chandra T (2008) Efficient projections onto the l_1-ball for learning in high dimensions. In: Machine learning, proceedings of the twenty-fifth international conference (ICML 2008), pp 272–279

Engel A, Broeck C (2001) Statistical mechanics of learning. Cambridge University, Cambridge

Feldman V, Gopalan P, Khot S, Ponnuswami AK (2009) On agnostic learning of parities, monomials, and halfspaces. SIAM J Comput 39(2):606–645

Feldman V, Guruswami V, Raghavendra P, Wu Y (2012) Agnostic learning of monomials by halfspaces is hard. SIAM J Comput 41(6):1558–1590

Flach P (2012) Machine learning: the art and science of algorithms that make sense of data. Cambridge University, Cambridge

Fligner MA, Verducci JS (1986) Distance based ranking models. J R Stat Soc 48(3):359–369

Franck M, Wolfe P (1956) An algorithm for quadratic programming. Naval Res Logis Q 3:95–110

Freund RM, Grigas P (2016) New analysis and results for the Frank-Wolfe method. Math Program 155(1–2):199–230

Freund Y, Iyer RD, Schapire RE, Singer Y (2003) An efficient boosting algorithm for combining preferences. J Mach Learn Res 4:933–969

Freund Y, Mason L (1999) The alternating decision tree learning algorithm. In: Proceedings of the 16th international conference on machine learning (ICML 1999), pp 124–133

Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139

Fürnkranz J, Hüllermeier E (2010) Preference learning. Springer, Berlin

Garber D, Hazan E (2016) A linearly convergent variant of the conditional gradient algorithm under strong convexity, with applications to online and stochastic optimization. SIAM J Optim 26(3):1493–1528

Garber D, Meshi O (2016) Linear-memory and decomposition-invariant linearly convergent conditional gradient algorithm for structured polytopes. In: Advances in neural information processing systems 29 (NIPS 2016), pp 1001–1009

Gärtner T, Vembu S (2009) On structured output training: hard cases and an efficient alternative. Mach Learn 76(2–3):227–242

Gentile C (2003) The robustness of the p-norm algorithms. Mach Learn 53(3):265–299

Getoor L, Taskar B (2007) Introduction to statistical relational learning. MIT, Cambridge

Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT, USA

Graves A, Mohamed A, Hinton GE (2013) Speech recognition with deep recurrent neural networks. In: IEEE international conference on acoustics, speech and signal processing (ICASSP 2013), pp 6645–6649

Grünwald P (2007) The minimum description length principle. MIT, USA

Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference, and prediction. Springer, Berlin

Haussler D (1988) Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. Artif Intell 36(2):177–221

Haussler D (1992) Decision theoretic generalizations of the PAC model for neural net and other learning applications. Inf Comput 100(1):78–150

Hazan E, Kale S (2012) Projection-free online learning. In: Proceedings of the 29th international conference on machine learning (ICML 2012)

Hegde C, Indyk P, Schmidt L (2015) A nearly-linear time framework for graph-structured sparsity. In: Proceedings of the 32nd international conference on machine learning (ICML 2015), pp 928–937

Helmbold DP, Sloan RH, Warmuth MK (1992) Learning integer lattices. SIAM J Comput 21(2): 240–266

Herbrich R (2002) Learning kernel classifiers: theory and algorithms. MIT, USA

Herbrich R, Graepel T, Obermayer K (2000) Large margin rank boundaries for ordinal regression. In: Advances in large margin classifiers. MIT Press, USA, pp 115–132

Hinton G, Deng L, Yu D, Dahl GE, r. Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN, Kingsbury B (2012) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process Mag 29(6):82–97

Hiriart-Urrut JB, Lemaréchal C (2004) Fundamentals of convex analysis. Springer, Berlin

Höffgen K, Simon HU (1992) Robust trainability of single neurons. In: Proceedings of the fifth annual acm conference on computational learning theory (COLT 1992), pp 428–439

Hsieh C, Chang K, Lin C, Keerthi SS, Sundararajan S (2008) A dual coordinate descent method for large-scale linear SVM. In: Proceedings of the 25th international conference on machine learning, pp 408–415

Hüllermeier E, Fürnkranz J, Cheng W, Brinker K (2008) Label ranking by learning pairwise preferences. Artif Intell 172(16–17):1897–1916

Jaggi M (2013) Revisiting frank-wolfe: projection-free sparse convex optimization. In: Proceedings of the 30th international conference on machine learning (ICML 2013), pp 427–435

Jain P, Rao N, Dhillon I (2016) Structured sparse regression via greedy hard thresholding. In: Advances in neural information processing systems 29 (NIPS 2016), pp 1516–1524

Jain P, Tewari A, Kar P (2014) On iterative hard thresholding methods for high-dimensional M-estimation. In: Advances in neural information processing systems 27 (NIPS 2014), pp 685–693

James G, Witten D, Hastie T, Tibshirani R (2013) An introduction to statistical learning: with applications in R. Springer texts in statistics, Springer, New York

Joachims T (2002) Optimizing search engines using clickthrough data. In: Proceedings of the 8th ACM international conference on knowledge discovery and data mining (SIGKDD 2002), pp 133–142

Johnson DS, Preparata FP (1978) The densest hemisphere problem. Theorertical Comput Sci 6:93–107

Kakade SM, Shalev-Shwartz S, Tewari A (2012) Regularization techniques for learning with matrices. J Mach Learn Res 13:1865–1890

Kalchbrenner N, Grefenstette E, Blunsom P (2014) A convolutional neural network for modelling sentences. In: Proceedings of the 52nd annual meeting of the association for computational linguistics (ACL 2014), pp 655–665

Kamishima T, Kazawa H, Akaho S (2010) A survey and empirical comparison of object ranking methods. Preference learning. Springer, Berlin, pp 181–201

Kawaguchi K (2016) Deep learning without poor local minima. In: Advances in neural information processing systems 29 (NIPS 2016), pp 586–594

Kawaguchi K, Kaelbling LP, Bengio Y (2017) Generalization in deep learning. CoRR. arXiv:1710.05468

Kazawa H, Hirao T, Maeda E (2005) Order SVM: a kernel method for order learning based on generalized order statistics. Syst Comput Jpn 36(1):35–43

Kearns M, Li M (1993) Learning in the presence of malicious errors. SIAM J Comput 22(4):807–837

Kearns M, Li M, Pitt L, Valiant L (1987) Recent results on boolean concept learning. In: Proceedings of the fourth international workshop on machine learning (ICML 1987), pp 337–352

Kearns M, Li M, Valiant LG (1994a) Learning boolean formulas. J. ACM 41(6):1298–1328

Kearns M, Schapire R, Sellie L (1994b) Toward efficient agnostic learning. Mach Learn 17(2): 115–141

Kearns M, Vazirani U (1994) An introduction to computational learning theory. MIT, USA

Kivinen J, Warmuth MK (1997) Exponentiated gradient versus gradient descent for linear predictors. Inf Comput 132(1):1–63

Klivans AR, O'Donnell R, Servedio RA (2004) Learning intersections and thresholds of halfspaces. J Comput Syst Sci 68(4):808–840

Klivans AR, Servedio RA (2004) Learning DNF in time $2^{\tilde{o}(n^{1/3})}$. J Comput Syst Sci 68(2):303–318

Klivans AR, Sherstov AA (2009) Cryptographic hardness for learning intersections of halfspaces. J Comput Syst Sci 75(1):2–12

Koller D, Friedman N (2009) Probabilistic graphical models. MIT, USA

Krichene W, Krichene S, Bayen AM (2015) Efficient bregman projections onto the simplex. In: Proceedings of the 54th IEEE conference on decision and control, (CDC 2015), pp 3291–3298

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems 25 (NIPS 2012), pp 1106–1114

Kulkarni S, Harman G (2011) An elementary introduction to statistical learning theory. Wiley series in probability and statistics, Wiley, New York

Kumar KSS, Bach FR (2013) Convex relaxations for learning bounded-treewidth decomposable graphs. In: Proceedings of the 30th international conference on machine learning (ICML 2013), pp 525–533

Kung S (2014) Kernel methods and machine learning. Cambridge University, Cambridge

Kushner HJ, Yin GG (2010) Stochastic approximation and recursive algorithms and applications. Springer, Berlin

Lacoste-Julien S, Jaggi M (2015) On the global linear convergence of frank-wolfe optimization variants. In: Advances in neural information processing systems 28 (NIPS 2015), pp 496–504

Lacoste-Julien S, Jaggi M, Schmidt MW, Pletscher P (2013) Block-coordinate frank-wolfe optimization for structural SVMs. In: Proceedings of the 30th international conference on machine learning (ICML 2013), pp 53–61

Lanckriet GRG, Cristianini N, Bartlett PL, Ghaoui LE, Jordan MI (2004) Learning the kernel matrix with semidefinite programming. J Mach Learn Res 5:27–72

Lauritzen SL (1995) The em algorithm for graphical association models with missing data. Comput Stat Data Anal 19(2):191–201

Lebanon G, Lafferty J (2002) Conditional models on the ranking poset. In: Advances in neural information processing systems 15 (NIPS 2002), pp 415–422

LeCun Y, Kavukcuoglu K, Farabet C (2010) Convolutional networks and applications in vision. In: Proceedings of the international symposium on circuits and systems (ISCAS 2010), pp 253–256

Lim CH, Wright SJ (2016) Efficient bregman projections onto the permutahedron and related polytopes. In: Proceedings of the 19th international conference on artificial intelligence and statistics (AISTATS 2016), pp 1205–1213

Little R, Rubin D (2014) Statistical analysis with missing data. Wiley, New York

Liu T, Lugosi G, Neu G, Tao D (2017) Algorithmic stability and hypothesis complexity. In: Proceedings of the 34th international conference on machine learning (ICML 2017), pp 2159–2167

Lu T, Boutilier C (2014) Effective sampling and learning for Mallows models with pairwise-preference data. J Mach Learn Res 15(1):3783–3829

Lu Z, Xiao L (2015) On the complexity analysis of randomized block-coordinate descent methods. Math Program 152(1–2):615–642
Ma Y, Fu Y (2011) Manifold learning theory and applications. CRC
Mahdavi M, Yang T, Jin R, Zhu S, Yi J (2012) Stochastic gradient descent with only one projection. In: Advances in neural information processing systems 25 (NIPS 2012), pp 503–511
Mallows CL (1957) Non-null ranking models. Biometrika 44(1–2):114–130
Megiddo N (1988) On the complexity of polyhedral separability. Discret Comput Geom 3(4): 325–337
Meila M, Chen H (2010) Dirichlet process mixtures of generalized mallows models. In: Proceedings of the twenty-sixth conference on uncertainty in artificial intelligence (UAI 2010), pp 358–367
Meila M, Jaakkola TS (2006) Tractable bayesian learning of tree belief networks. Stat Comput 16(1):77–92
Mitchell T (1982) Generalization as search. Artif Intell 18(2):203–226
Mitchell T (1997) Machine learning. McGraw-Hill Education
Mohammadi L, van de Geer S (2005) Asymptotics in empirical risk minimization. J Mach Learn Res 6:2027–2047
Mohri M, Rostamizadeh A, Talwalkar A (2012) Foundations of machine learning. MIT, USA
Mukherjee S, Niyogi P, Poggio T, Rifkin R (2006) Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization. Adv Comput Math 25(1):161–193
Murphy K (2012) Machine learning: a probabilistic perspective. MIT, USA
Natarajan B (1991) Machine learning: a theoretical approach. M. Kaufmann Publishers
Natarajan B (1995) Sparse approximate solutions to linear systems. SIAM J Comput 24(2):227–234
Nemirovski A (1995) Efficient methods in convex programming. http://www2.isye.gatech.edu/~nemirovs/Lec_EMCO.pdf
Nemirovski AS, Yudin DB (1983) Problem complexity and method efficiency in optimization. J. Wiley, New York
Nesterov Y (2004) Introductory lectures on convex optimization: a basic course. Kluwer Academic Publishers
Nesterov Y (2012) Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM J Optim 22(2):341–362
Nie S, Mauá DD, de Campos CP, Ji Q (2014) Advances in learning bayesian networks of bounded treewidth. In: Advances in neural information processing systems 27 (NIPS 2014), pp 2285–2293
Parberry I (1994) Circuit complexity and neural networks. MIT, USA
Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Mateo
Pinheiro PHO, Collobert R (2014) Recurrent convolutional neural networks for scene labeling. In: Proceedings of the 31th international conference on machine learning (ICML 2014), pp 82–90
Pitt L, Valiant L (1988) Computational limitations on learning from examples. J ACM 35(4):965–984
Plackett RL (1975) The analysis of permutations. J R Stat Soc 24(10):193–202
Poffio T, Rifkin R, Kukherjee S, Niyogi P (2004) General conditions for predictivity in learning theory. Nature 428(6981):419
Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106
Quinlan JR (1993) C4. 5: Programs for machine learning. Morgan Kaufmann
Quinlan JR (1996) Bagging, boosting, and C4.5. In: Proceedings of the 30th national conference on artificial intelligence (AAAI 1996), pp 725–730
Rakhlin A, Shamir O, Sridharan K (2012) Making gradient descent optimal for strongly convex stochastic optimization. In: Proceedings of the 29th international conference on machine learning (ICML 2012)
Rish I, Grabarnik G (2014) Sparse modeling: theory, algorithms, and applications. CRC
Rissanen J (1983) A universal prior for integers and estimation by minimum description length. Ann stat 416–431

Rissanen J (1985) Minimum description length principle. Wiley, New York
Rivest RL (1987) Learning decision lists. Mach Learn 2(3):229–246
Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat 22(3):400–407
Rockafellar T (1970) Convex analysis. Princeton University, Princeton
Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65:386–408
Rumelhart DE, Hinton GE, Williams RJ (1986) Learning internal representations by error propagation. In: Parallel distributed processing: explorations in the microstructure of cognition, vol 1. MIT, USA, pp 318–362
Sauer N (1972) On the density of families of sets. J Comb Theory 13:145–147
Schapire RE (1990) The strength of weak learnability. Mach Learn 5:197–227
Schapire RE, Freund Y (2012) Boosting. MIT, USA
Schapire RE, Singer Y (1999) Improved boosting algorithms using confidence-rated predictions. Mach Learn 37(3):297–336
Schölkopf B, Herbrich R, Smola AJ (2001) A generalized representer theorem. In: Proceedings of the 14th annual conference on computational on computational learning theory (COLT 2001), pp 416–426
Schölkopf B, Smola A (2002) Learning with Kernels: support vector machines, regularization, optimization, and beyond. Adaptive computation and machine learning, MIT, USA
Shalev-Shwartz S, Ben-David S (2014) Understanding machine learning: from theory to algorithms. Cambridge University, Cambridge
Shalev-Shwartz S, Shamir O, Shammah S (2017) Failures of gradient-based deep learning. In: Proceedings of the 34th international conference on machine learning (ICML 2017), pp 3067–3075
Shalev-Shwartz S, Shamir O, Srebro N, Sridharan K (2009) Stochastic convex optimization. In: Proceedings of the 22nd conference on learning theory (COLT 2009), pp 177–186
Shalev-Shwartz S, Shamir O, Srebro N, Sridharan K (2010) Learnability, stability and uniform convergence. J Mach Learn Res 11:2635–2670
Shalev-Shwartz S, Singer Y, Srebro N (2007) Pegasos: primal estimated sub-gradient solver for SVM. In: Proceedings of the 24th international conference on machine learning (ICML 2007), pp 807–814
Shalev-Shwartz S, Srebro N, Zhang T (2010) Trading accuracy for sparsity in optimization problems with sparsity constraints. SIAM J Optim 20(6):2807–2832
Shalev-Shwartz S, Tewari A (2011) Stochastic methods for l_1-regularized loss minimization. J Mach Learn Res 12:1865–1892
Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Kernel methods for pattern analysis, Cambridge University, Cambridge
Song L, Vempala S, Wilmes J, Xie B (2017) On the complexity of learning neural networks. CoRR. arXiv:1707.04615
Sra S, Nowozin S, Wright S (2012) Optimization for machine learning. Neural information processing series, MIT, USA
Srebro N (2003) Maximum likelihood bounded tree-width Markov networks. Artif Intell 143(1):123–138
Sridharan K (2012) Learning from an optimization viewpoint. Ph.D. thesis, Technicological Institute of Chicago, Toyota
Steinwart I, Christmann A (2008) Support vector machines. Information science and statistics, Springer, Berlin
Sugiyama M (2015) Introduction to statistical machine learning. Elsevier Science
Theodoridis S (2015) Machine learning: a bayesian and optimization perspective. Elsevier Science
Tikhonov A (1943) On the stability of inverse problems. Doklady Akademii Nauk SSSR 39(5):195–198
Tseng P, Yun S (2009) A coordinate gradient descent method for nonsmooth separable minimization. Math Program 117(1–2):387–423

Turing A (1950) Computing machinery and intelligence. Mind 59:433–460

Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

van Beek P, Hoffmann H (2015) Machine learning of bayesian networks using constraint programming. In: Proceedings of the 21st confernce on principles and practice of constraint programming (CP 2015), pp 429–445

Vapnik V (1998) Statistical learning theory. Wiley, New York

Vapnik V (2013) The nature of statistical learning theory, 3rd edn. Springer, Berlin

Vapnik V, Chervonenkis A (1974) Theory of pattern recognition. Nauka, Moskow (in Russian)

Vembu S, Gärtner T (2010) Label ranking algorithms: a survey. In: Preference learning. Springer, Berlin, pp 45–64

Vembu S, Gärtner T, Boley M (2009) Probabilistic structured predictors. In: Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence (UAI 2009), pp 557–564

Viola PA, Jones MJ (2001) Robust real-time face detection. In: Proceedings of the 8th international conference on computer vision ICCV 2001, p 747

Wainwright M, Jordan M (2008) Graphical models, exponential families, and variational inference. Found Trends Mach Learn 1(1–2):1–305

Watanabe S (2009) Algebraic Geometry and Statistical Learning Theory. Cambridge University, Cambridge

Webb A, Copsey K (2011) Statistical pattern recognition. Wiley, New York

Wibisono A, Rosasco L, Poggio T (2009) Sufficient conditions for uniform stability of regularization algorithms. Technical Report MIT-CSAIL-TR-2009-060. MIT, Computer Science and artificial intelligence laboratory

Wright SJ (2015) Coordinate descent algorithms. Math Program 151(1):3–34

Xu J, Li H (2007) AdaRank: a boosting algorithm for information retrieval. In: Proceedings of the 30th annual international ACM conference on research and development in information retrieval (SIGIR 2007), pp 391–398

Zhang L, Yang T, Jin R, He X (2013) $O(\log T)$ projections for stochastic optimization of smooth and strongly convex functions. In: Proceedings of the 30th international conference on machine learning (ICML 2013), pp 1121–1129

Zhang X (2010) Empirical risk minimization. In: Sammut C, Webb G, (eds) Encyclopedia of machine learning, Springer, Berlin, p 312

Zhang Y, Liang P, Wainwright M (2017) Convexified convolutional neural networks. In: Proceedings of the 34th international conference on machine learning (ICML 2017), pp 4044–4053

Zhao Z, Piech P, Xia L (2016) Learning mixtures of plackett-luce models. In: Proceedings of the 33nd international conference on machine learning (ICML 2016), pp 2906–2914