# 3

# *Supervised learning*

Previously, we talked about the fundamentals of prediction and statistical modeling of populations. Our goal was, broadly speaking, to use available information described by a random variable $X$ to conjecture about an unknown outcome $Y$.

In the important special case of a binary outcome $Y$, we saw that we can write an optimal predictor $\widehat{Y}$ as a threshold of some function $f$:

$$\widehat{Y}(x) = \mathbb{1}\{f(x) > t\}$$

We saw that in many cases the optimal function is a ratio of two likelihood functions.

This optimal predictor has a serious limitation in practice, however. To be able to compute the prediction for a given input, we need to know a probability density function for the positive instances in our problem and also one for the negative instances. But we are often unable to construct or unwilling to assume a particular density function.

As a thought experiment, attempt to imagine what a probability density function over images labeled *cat* might look like. Coming up with such a density function appears to be a formidable task, one that's not intuitively any easier than merely classifying whether an image contains a cat or not.

In this chapter, we transition from a purely mathematical characterization of optimal predictors to an algorithmic framework. This framework has two components. One is the idea of working with finite samples from a population. The other is the theory of supervised learning and it tells us how to use finite samples to build predictors algorithmically.

## *Sample versus population*

Let's take a step back to reflect on the interpretation of the pair of random variables $(X, Y)$ that we've worked with so far. We think of the random variables $(X, Y)$ as modeling a population of instances in our prediction problem. From this pair of random variables, we can derive other random variables such as a predictor $\widehat{Y} = \mathbb{1}\{f(X) > t\}$. All of these are random variables in the same probability space. When we talk about, say, the true positive rate of the predictor $\widehat{Y}$, we therefore make a statement about the joint distribution of $(X, Y)$.

In almost all prediction problems, however, we do not have access to the entire population of instances that we will encounter. Neither do we have a probability model for the joint distribution of the random variables $(X, Y)$. The joint distribution is a theoretical construct that we can reason about, but it doesn't readily tell us what to do when we don't have precise knowledge of the joint distribution.

What knowledge then do we typically have about the underlying population and how can we use it algorithmically to find good predictors? In this chapter we will begin to answer both questions.

First we assume that from past experience we have observed $n$ labeled instances $(x_1, y_1), ..., (x_n, y_n)$. We assume that each data point $(x_i, y_i)$ is a draw from the same underlying distribution $(X, Y)$. Moreover, we will often assume that the data points are drawn independently. This pair of assumptions is often called the "i.i.d. assumption", a shorthand for *independent and identically distributed*.

To give an example, consider a population consisting of all currently eligible voters in the United States and some of their features, such as, age, income, state of residence etc. An *i.i.d. sample* from this population would correspond to a repeated sampling process that selects a uniformly random voter from the entire reference population.

Sampling is a difficult problem with numerous pitfalls that can strongly affect the performance of statistical estimators and the validity of what we learn from data. In the voting example, individuals might be unreachable or decline to respond. Even defining a good population for the problem we're trying to solve is often tricky. Populations can change over time. They may depend on a particular social context, geography, or may not be neatly characterized by formal criteria. Task yourself with the idea of taking a random sample of spoken sentences in the English language, for example, and you will quickly run into these issues.

In this chapter, as is common in learning theory, we largely ignore these important issues. We instead focus on the significant challenges that remain even if we have a well-defined population and an unbiased sample from it.

## *Supervised learning*

*Supervised learning* is the prevalent method for constructing predictors from data. The essential idea is very simple. We assume we have labeled data, in this context also called *training examples*, of the form $(x_1, y_1), ..., (x_n, y_n)$, where each *example* is a pair $(x_i, y_i)$ of an *instance* $x_i$ and a corresponding *label* $y_i$. The notion of *supervision* refers to the availability of these labels.

Given such a collection of labeled data points, supervised learning turns the task of finding a good predictor into an optimization problem involving these data points. This optimization problem is called *empirical risk minimization*.

Recall, in the last chapter we assumed full knowledge of the joint distribution of $(X, Y)$ and analytically found predictors that minimize risk. The risk is equal to the expected value of a *loss function* that quantifies the cost of each possible prediction for a given true outcome. For binary prediction problems, there are four possible pairs of labels corresponding to true positives, false positives, true negatives, and false negatives. In this case, the loss function boils down to specifying a cost to each of the four possibilities.

More generally, a loss function is a function $\mathit{loss}\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, where $\mathcal{Y}$ is the set of values that $Y$ can assume. Whereas previously we focused on the predictor $\widehat{Y}$ as a random variable, in this chapter our focus shifts to the functional form that the predictor has. By convention, we write $\widehat{Y} = f(X)$, where $f\colon \mathcal{X} \to \mathcal{Y}$ is a function that maps from the sample space $\mathcal{X}$ into the label space $\mathcal{Y}$.

Although the random variable $\widehat{Y}$ and the function $f$ are mathematically not the same objects, we will call both a predictor and extend our risk definition to apply the function as well:

$$R[f] = \mathbb{E}\left[\mathit{loss}(f(X), Y)\right] .$$

The main new definition in this chapter is a finite sample analog of the risk, called empirical risk.

**Definition 1.** *Given a set of labeled data points $S = ((x_1, y_1), ..., (x_n, y_n))$, the* empirical risk *of a predictor $f\colon \mathcal{X} \to \mathcal{Y}$ with respect to the sample $S$ is defined as*

$$R_S[f] = \frac{1}{n} \sum_{i=1}^{n} \mathit{loss}(f(x_i), y_i) .$$

Rather than taking expectation over the population, the empirical risk averages the loss function over a finite sample. Conceptually, we think of the finite sample as something that is in our possession, e.g., stored on our hard disk.

Empirical risk serves as a proxy for the risk. Whereas the risk $R[f]$ is a population quantity—that is, a property of the joint distribution $(X, Y)$ and our predictor $f$—the empirical risk is a *sample quantity*.

We can think of the empirical risk as the sample average estimator of the risk. When samples are drawn i.i.d., the empirical risk is a random variable that equals the sum of $n$ independent random variables. If losses are bounded, the central limit theorem therefore suggests that the empirical risk approximates the risk for a fixed predictor $f$.

Regardless of the distribution of $S$, however, note that we can always compute the empirical risk $R_S[f]$ entirely from the sample $S$ and the predictor $f$. Since empirical risk a quantity we can compute from samples alone, it makes sense to turn it into an objective function that we can try to minimize numerically.

*Empirical risk minimization* is the optimization problem of finding a predictor in a given function family that minimizes the empirical risk.

**Definition 2.** *Given a function class* $\mathcal{F} \subseteq \mathcal{X} \to \mathcal{Y}$, empirical risk minimization *on a set of labeled data points S corresponds to the objective:*

$$\min_{f \in \mathcal{F}} R_S[f]$$

*A solution to the optimization problem is called* empirical risk minimizer.

There is a tautology relating risk and empirical risk that is good to keep in mind:

$$R[f] = R_S[f] + (R[f] - R_S[f])$$

Although mathematically trivial, the tautology reveals an important insight. To minimize risk, we can first attempt to minimize empirical risk. If we successfully find a predictor $f$ that achieves small empirical risk $R_S[f]$, we're left worrying about the term $R[f] - R_S[f]$. This term quantifies how much the empirical risk of $f$ underestimates its risk. We call this difference *generalization gap* and it is of fundamental importance to machine learning. Intuitively speaking, it tells us how well the performance of our predictor transfers from seen examples (the training examples) to unseen examples (a fresh example from the population) drawn from the same distribution. This process is called *generalization*.

Generalization is not the only goal of supervised learning. A constant predictor that always outputs 0 generalizes perfectly well, but is almost always entirely useless. What we also need is that the predictor achieves small empirical risk $R_S[f]$. Making the empirical risk small is fundamentally about *optimization*. As a consequence, a large part of supervised learning deals with optimization. For us to be able to talk about optimization, we need to commit to a *representation* of the function class $\mathcal{F}$ that appears in the empirical risk minimization problem. The representation of the function class, as well as the choice of a suitable loss function, determines whether or not we can efficiently find an empirical risk minimizer.

To summarize, introducing empirical risk minimization directly leads to three important questions that we will work through in turn.

- **Representation:** What is the class of functions $\mathcal{F}$ we should choose?
- **Optimization:** How can we efficiently solve the resulting optimization problem?
- **Generalization:** Will the performance of predictor transfer gracefully from seen training examples to unseen instances of our problem?

These three questions are intertwined. Machine learning is not so much about studying these questions in isolation as it is about the often delicate interplay between them. Our choice of representation influences both the difficulty of optimization and our generalization performance. Improvements in optimization may not help, or could even hurt, generalization. Moreover, there are aspects of the problem that don't neatly fall into only one of these categories. The choice of the loss function, for example, affects all of the three questions above.

There are important differences between the three questions. Results in optimization, for example, tend to be independent of the statistical assumptions about the data generating process. We will see a number of different optimization methods that under certain circumstances find either a global or local minimum of the empirical risk objective. In contrast, to reason about generalization, we need some assumptions about the data generating process. The most common one is the i.i.d.-assumption we discussed earlier. We will also see several mathematical frameworks for reasoning about the gap between risk and empirical risk.

Let's start with a foundational example that illustrates these core concepts and their interplay.

## *A first learning algorithm: The perceptron*

As we discussed in the introduction, in 1958 the New York Times reported the Office of Naval Research claiming the perceptron algorithm[1] would "be able to walk, talk, see, write, reproduce itself and be conscious of its existence." Let's now dive into this algorithm that seemed to have such unbounded potential.

Toward introducing this algorithm, let's assume we're in a binary prediction problem with labels in $\{-1, 1\}$ for notational convenience. The perceptron algorithm aims to find a *linear separator* of the data, that is, a hyperplane specified by coefficients $w \in \mathbb{R}^d$ that so that all positive examples lie on one side of the hyperplane and all negative ones on the other.

Formally, we can express this as $y_i\langle w, x_i\rangle > 0$. In other words, the linear function $f(x) = \langle w, x\rangle$ agrees in sign with the labels on all training instances $(x_i, y_i)$. In fact, the perceptron algorithm will give us a bit more. Specifically, we require that the sign agreement has some *margin* $y_i\langle w, x_i\rangle \geq$

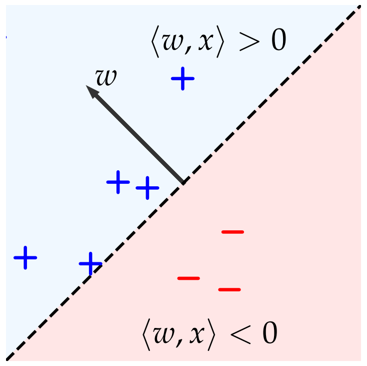

Figure 1: Illustration of a linear separator

1. That is, when $y = 1$, the linear function must take on a value of at least 1 and when $y = -1$, the linear function must be at most $-1$. Once we find such a linear function, our prediction $\widehat{Y}(x)$ on a data point $x$ is $\widehat{Y}(x) = 1$ if $\langle w, x \rangle \geq 0$ and $\widehat{Y}(x) = -1$ otherwise.

The algorithm goes about finding a linear separator $w$ incrementally in a sequence of update steps.

**Perceptron**

- Start from the initial solution $w_0 = 0$
- At each step $t = 0, 1, 2, ...$:
    - Select a random index $i \in \{1, ..., n\}$
    - Case 1: If $y_i \langle w_t, x_i \rangle < 1$, put

$$w_{t+1} = w_t + y_i x_i$$

    - Case 2: Otherwise put $w_{t+1} = w_t$.

Case 1 corresponds to what's called a *margin mistake*. The sign of the linear function may not disagree with the label, but it doesn't have the required margin that we asked for.

When an update occurs, we have

$$\langle w_{t+1}, x_i \rangle = \langle w_t, x_i \rangle + \|x_i\|^2 .$$

In this sense, the algorithm is nudging the hyperplane to be less wrong on example $x_i$. However, in doing so it could introduce errors on other examples. It is not yet clear that the algorithm converges to a linear separator when this is possible.

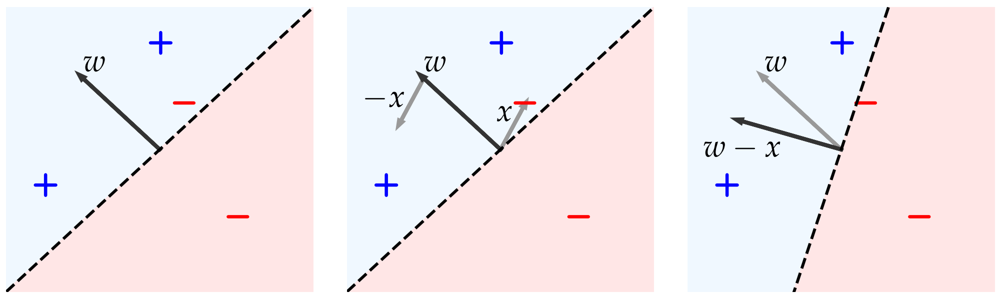

Figure 2: Illustration of the perceptron update. Left: One misclassified example $x$. Right: After update.

## *Connection to empirical risk minimization*

Before we turn to the formal guarantees of the perceptron, it is instructive to see how to relate it to empirical risk minimization. In order to do so, it's helpful to introduce two *hyperparameters* to the algorithm by considering the alternative update rule:

$$w_{t+1} = \gamma w_t + \eta y_i x_i$$

Here $\eta$ is a positive scalar called a *learning rate* and $\gamma \in [0, 1]$ is called the *forgetting rate*.

First, it's clear from the description that we're looking for a linear separator. Hence, our function class is the set of linear functions $f_w(x) = \langle w, x \rangle$, where $w \in \mathbb{R}^d$. We will sometimes call the vector $w$ the *weight vector* or vector of *model parameters*.

An optimization method that picks a random example at each step and makes a local improvement to the model parameters is the *stochastic gradient method*. This method will figure prominently in our chapter on optimization as it is the workhorse of many machine learning applications today. The local improvement the method picks at each step is given by a local linear approximation of the loss function around the current model parameters. This linear approximation can be written neatly in terms of the vector of first derivatives, called *gradient*, of the loss function with respect to the current model parameters.

The formal update rule reads

$$w_{t+1} = w_t - \eta \nabla_{w_t} loss(f_{w_t}(x_i), y_i)$$

Here, the example $(x_i, y_i)$ is randomly chosen and the expression $\nabla_{w_t} loss(f_{w_t}(x_i), y_i)$ is the gradient of the loss function with respect to the model parameters $w_t$

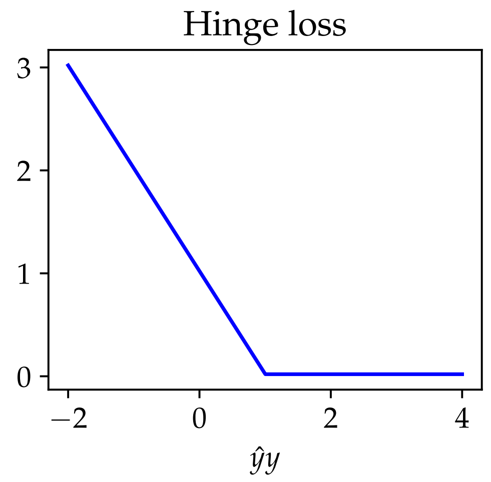

Figure 3: Hinge loss

on the example $(x_i, y_i)$. We will typically drop the vector $w_t$ from the subscript of the gradient when it's clear from the context. The scalar $\eta > 0$ is a step size parameter that we will discuss more carefully later. For now, think of it as a small constant.

It turns out that we can connect this update rule with the perceptron algorithm by choosing a suitable loss function. Consider the loss function

$$\mathit{loss}(\langle w, x\rangle, y) = \max\left\{1 - y\langle w, x\rangle,\, 0\right\}.$$

This loss function is called *hinge loss*. Note that its gradient is $-yx$ when $y\langle w, x\rangle < 1$ and 0 when $y\langle w, x\rangle > 1$.

The gradient of the hinge loss is not defined when $y\langle w, x\rangle = 1$. In other words, the loss function is not differentiable everywhere. This is why technically speaking the stochastic gradient method operates with what is called a *subgradient*. The mathematical theory of subgradient optimization rigorously justifies calling the gradient 0 when $y\langle w, x\rangle = 1$. We will ignore this technicality throughout the book.

We can see that the hinge loss gives us part of the update rule in the perceptron algorithm. The other part comes from adding a weight penalty $\frac{\lambda}{2}\|w\|^2$ to the loss function that discourages the weights from growing out of bounds. This weight penalty is called *$\ell_2$-regularization*, *weight decay*, or *Tikhonov regularization* depending on which field you work in. The purpose of regularization is to promote generalization. We will therefore return to regularization in detail when we discuss generalization in more depth. For now, note that the margin constraint we introduced is inconsequential unless we penalize large vectors. Without the weight penalty we could simply scale up any linear separator until it separates the points with the desired margin.

Putting the two loss functions together, we get the $\ell_2$-regularized empirical risk minimization problem for the hinge loss:

$$\frac{1}{n}\sum_{i=1}^{n} \max\left\{1 - y_i\langle w, x_i\rangle,\, 0\right\} + \frac{\lambda}{2}\|w\|_2^2$$

The perceptron algorithm corresponds to solving this empirical risk objective with the stochastic gradient method. The constant $\eta$, which we dubbed the learning rate, is the step size of the stochastic gradient methods. The forgetting rate constant $\gamma$ is equal to $(1 - \eta\lambda)$. The optimization problem is also known as *support vector machine* and we will return to it later on.

## *A word about surrogate losses*

When the goal was to maximize the accuracy of a predictor, we mathematically solved the risk minimization problem with respect to the *zero-one loss*

$$\mathit{loss}(\widehat{y}, y) = \mathbb{1}\{\widehat{y} \neq y\}$$

that gives us penalty 1 if our label is incorrect, and penalty 0 if our predicted label $\widehat{y}$ matches the true label $y$. We saw that the optimal predictor in this case was a *maximum a posteriori* rule, where we selected the label with higher posterior probability.

Why don't we directly solve empirical risk minimization with respect to the zero-one loss? The reason is that the empirical risk with the zero-one loss is computationally difficult to optimize directly. In fact, this optimization problem is NP-hard even for linear prediction rules.[2] To get a better sense of the difficulty, convince yourself that the stochastic gradient method, for example, fails entirely on the zero-one loss objective. Of course, the stochastic gradient method is not the only learning algorithm.

The hinge loss therefore serves as a *surrogate loss* for the zero-one loss. We hope that by optimizing the hinge loss, we end up optimizing the zero-one loss as well. The hinge loss is not the only reasonable choice. There are numerous loss functions that approximate the zero-one loss in different ways.

- The *hinge loss* is $\max\{1 - y\widehat{y}, 0\}$ and *support vector machine* refers to empirical risk minimization with the hinge loss and $\ell_2$-regularization. This is what the perceptron is optimizing.
- The *squared loss* is given by $\frac{1}{2}(y - \widehat{y})^2$. Linear least squares regression corresponds to empirical risk minimization with the squared loss.

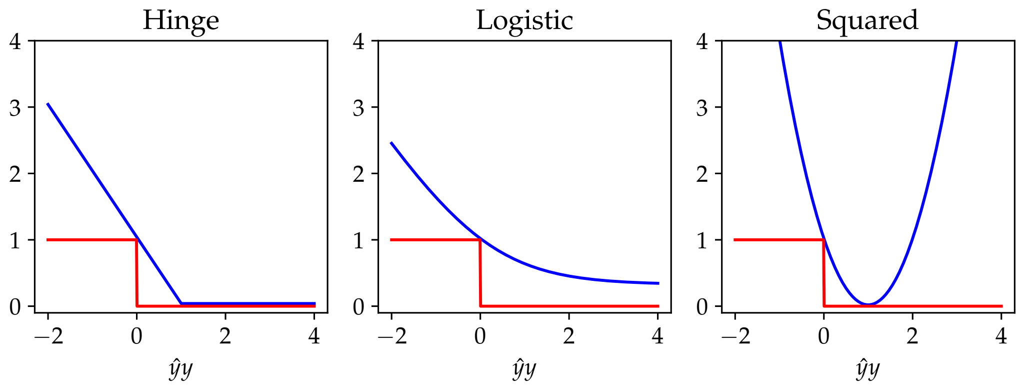

Figure 4: Hinge, squared, logistic loss compared with the zero-one loss.

- The *logistic loss* is $-\log(\sigma(\widehat{y}))$ when $y = 1$ and $-\log(1 - \sigma(\widehat{y}))$ when $y = -1$, where $\sigma(z) = 1/(1 + \exp(-z))$ is the logistic function. *Logistic regression* corresponds to empirical risk minimization with the logistic loss and linear functions.

Sometimes we can theoretically relate empirical risk minimization under a surrogate loss to the zero-one loss. In general, however, these loss functions are used heuristically and practitioners evaluate performance by trial-and-error.

## *Formal guarantees for the perceptron*

We saw that the perceptron corresponds to finding a linear predictor using the stochastic gradient method. What we haven't seen yet is a proof that the perceptron method works and under what conditions. Recall that there are two questions we need to address. The first is why the perceptron method successfully fits the training data, a question about *optimization*. The second is why the solution should also correctly classify unseen examples drawn from the same distribution, a question about *generalization*. We will address each in turn with results from the 1960s. Even though the analysis here is over 50 years old, it has all of the essential parts of more recent theoretical arguments in machine learning.

### *Mistake bound*

To see why we perform well on the training data, we use a *mistake bound* due to Novikoff.[3] The bound shows that if there exists a linear separator of the training data, then the perceptron will find it quickly provided that the *margin* of the separating hyperplane isn't too small.

Margin is a simple way to evaluate how well a predictor separates data. Any vector $w \in \mathbb{R}^d$ defines a hyperplane $\mathcal{H}_w = \{x \ : \ w^T x = 0\}$. Suppose that the hyperplane $\mathcal{H}_w$ corresponding to the vector $w$ perfectly separates the data in $S$. Then we define the margin of such a vector $w$ as the smallest distance of our data points to this hyperplane:

$$\gamma(S, w) = \min_{1 \le i \le n} \operatorname{dist}(x_i, \mathcal{H}_w)\,.$$

Here,

$$\operatorname{dist}(x, \mathcal{H}_w) = \min\{\|x - x'\| \colon x' \in \mathcal{H}_w\} = \frac{|\langle x, w\rangle|}{\|w\|}.$$

Overloading terminology, we define the margin of a *dataset* to be the maximum margin achievable by any predictor $w$:

$$\gamma(S) = \max_{\|w\|=1} \gamma(S, w)\,.$$

We will now show that when a dataset has a large margin, the perceptron algorithm will find a separating hyperplane quickly.

Let's consider the simplest form of the perceptron algorithm. We initialize the algorithm with $w_0 = 0$. The algorithm proceeds by selecting a random index $i_t$ at step $t$ checking whether $y_{i_t} w_t^T x_{i_t} < 1$. We call this condition a margin mistake, i.e., the prediction $w_t^T x_{i_t}$ is either wrong or too close to the hyperplane. If a margin mistake occurs, the perceptron performs the update

$$w_{t+1} = w_t + y_{i_t} x_{i_t}\,.$$

That is, we rejigger the hyperplane to be more aligned with the signed direction of the mistake. If no margin mistake occurs, then $w_{t+1} = w_t$.

To analyze the perceptron we need one additional definition. Define the diameter of a data $S$ to be

$$D(S) = \max_{(x,y) \in S} \|x\|\,.$$

We can now summarize a worst case analysis of the perceptron algorithm with the following theorem.

**Theorem 1.** *The perceptron algorithm makes at most $(2 + D(S)^2)\gamma(S)^{-2}$ margin mistakes on any sequence of examples $S$ that can be perfectly classified by a linear separator.*

*Proof.* The main idea behind the proof of this theorem is that since $w$ only changes when you make a mistake, we can upper bound and lower bound $w$ at each time a mistake is made, and then, by comparing these two bounds, compute an inequality on the total number of mistakes.

To find an upper bound, suppose that at step $t$ the algorithm makes a margin mistake. We then have the inequality:

$$\begin{aligned}\|w_{t+1}\|^2 &= \|w_t + y_{i_t} x_{i_t}\|^2 \\ &= \|w_t\|^2 + 2y_{i_t}\langle w_t, x_{i_t}\rangle + \|x_{i_t}\|^2 \\ &\leq \|w_t\|^2 + 2 + D(S)^2\,.\end{aligned}$$

The final inequality uses the fact that $y_{i_t}\langle w_t, x_{i_t}\rangle < 1$. Now, let $m_t$ denote the total number of mistakes made by the perceptron in the first $t$ iterations. Summing up the above inequality over all the mistakes we make and using the fact that $\|w_0\| = 0$, we get our upper bound on the norm of $w_t$:

$$\|w_t\| \leq \sqrt{m_t(2 + D(S)^2)}\,.$$

Working toward a lower bound on the norm of $w_t$, we will use the following argument. Let $w$ be any unit vector that correctly classifies all points in $S$. If we make a mistake at iteration $t$, we have

$$\langle w, w_{t+1} - w_t\rangle = \langle w, y_{i_t} x_{i_t}\rangle = \frac{|\langle w, x_{i_t}\rangle|}{\|w\|} \geq \gamma(S, w)\,.$$

Note that the second equality here holds because $w$ correctly classifies the point $(x_{i_t}, y_{i_t})$. This is where we use that the data are linearly separable. The inequality follows from the definition of margin.

Now, let $w_\star$ denote the hyperplane that achieves the maximum margin $\gamma(S)$. Instantiating the previous argument with $w_\star$, we find that

$$\|w_t\| \geq \langle w_\star, w_t\rangle = \sum_{k=1}^{t} w_\star^T(w_k - w_{k-1}) \geq m_t\gamma(S)\,,$$

where the equality follows from a telescoping sum argument.

This yields the desired lower bound on the norm of $w_t$. Combined with the upper bound we already derived, it follows that the total number of mistakes has the bound

$$m_t \leq \frac{2 + D(S)^2}{\gamma(S)^2}\,. \qquad \square$$

The proof we saw has some ingredients we'll encounter again. Telescoping sums, for example, are a powerful trick used throughout the analysis of optimization algorithms. A telescoping sum lets us understand the behavior of the final iterate by decomposing it into the incremental updates of the individual iterations.

The mistake bound does not depend on the dimension of the data. This is appealing since the requirement of linear separability and high margin, intuitively speaking, become less taxing the larger the dimension is.

An interesting consequence of this theorem is that if we run the perceptron repeatedly over the same dataset, we will eventually end up with a separating hyperplane. To see this, imagine repeatedly running over the dataset until no mistake occurred on a full pass over the data. The mistake bound gives a bound on the number of passes required before the algorithm terminates.

## *From mistake bounds to generalization*

The previous analysis shows that the perceptron finds a good predictor on the training data. What can we say about new data that we have not yet seen?

To talk about generalization, we need to make a statistical assumption about the data generating process. Specifically we assume that the data points in the training set $S = \{(x_1, y_1) \dots, (x_n, y_n)\}$ where each drawn i.i.d. from a fixed underlying distribution $\mathcal{D}$ with the labels taking values $\{-1, 1\}$ and each $x_i \in \mathbb{R}^d$.

We know that the perceptron finds a good linear predictor for the training data (if it exists). What we now show is that this predictor also works on new data drawn from the same distribution.

To analyze what happens on new data, we will employ a powerful *stability* argument. Put simply, an algorithm is stable if the effect of removing or replacing a single data point is small. We will do a deep dive on stability in our chapter on generalization, but we will have a first encounter with the idea here.

The perceptron is stable because it makes a bounded number of mistakes. If we remove a data point where no mistake is made, the model doesn't change at all. In fact, it's as if we had *never seen* the data point. This lets us relate the performance on seen examples to the performance on examples in the training data on which the algorithm never updated.

Vapnik and Chervonenkis presented the following stability argument in their classic text from 1974, though the original argument is likely a decade older.[4] Their main idea was to leverage our assumption that the data are i.i.d., so we can swap the roles of training and test examples in the analysis.

**Theorem 2.** *Let $S_n$ denote a training set of $n$ i.i.d. samples from a distribution $\mathcal{D}$ that we assume has a perfect linear separator. Let $w(S)$ be the output of the* *perceptron on a dataset $S$ after running until the hyperplane makes no more margin mistakes on $S$. Let $Z = (X, Y)$ be an additional independent sample from $\mathcal{D}$. Then, the probability of making a margin mistake on $(X, Y)$ satisfies the upper bound*

$$\mathbb{P}[Yw(S_n)^T X < 1] \le \frac{1}{n+1}\mathbb{E}_{S_{n+1}}\left[\frac{2 + D(S_{n+1})^2}{\gamma(S_{n+1})^2}\right].$$

*Proof.* First note that

$$\mathbb{P}[Yw^T X < 1] = \mathbb{E}[\mathbb{1}\{Yw^T X < 1\}].$$

Let $S_n = (Z_1, ..., Z_n)$ and with $Z_k = (X_k, Y_k)$ and put $Z_{n+1} = Z = (X, Y)$. Note that these $n+1$ random variables are i.i.d. draws from $\mathcal{D}$. As a purely analytical device, consider the "leave-one-out set"

$$S^{-k} = \{Z_1, \dots, Z_{k-1}, Z_{k+1}, ..., Z_{n+1}\}.$$

Since the data are drawn i.i.d., running the algorithm on $S^{-k}$ and evaluating it on $Z_k = (X_k, Y_k)$ is equivalent to running the algorithm on $S_n$ and evaluating it on $Z_{n+1}$. These all correspond to the same random experiment and differ only in naming. In particular, we have

$$\mathbb{P}[Yw(S_n)^T X < 1] = \frac{1}{n+1}\sum_{k=1}^{n+1}\mathbb{E}[\mathbb{1}\{Y_k w(S^{-k})^T X_k < 1\}].$$

Indeed, we're averaging quantities that are each identical to the left hand side. But recall from our previous result that the perceptron makes at most

$$m = \frac{2 + D((Z_1, \dots, Z_{n+1}))^2}{\gamma((Z_1, \dots, Z_{n+1}))^2}$$

margin mistakes when run on the entire sequence $(Z_1, \dots, Z_{n+1})$. Let $i_1, \dots, i_m$ denote the indices on which the algorithm makes a mistake in any of its cycles over the data. If $k \notin \{i_1, \dots, i_m\}$, the output of the algorithm remains the same after we remove the $k$-th sample from the sequence. It follows that such $k$ satisfy $Y_k w(S^{-k}) X_k \ge 1$ and therefore $k$ does not contribute to the summation above. The other terms can at most contribute 1 to the summation. Hence,

$$\sum_{k=1}^{n+1}\mathbb{1}\{Y_k w(S^{-k})^T X_k < 1\} \le m,$$

and by linearity of expectation, as we hoped to show,

$$\mathbb{P}[Yw(S_n)^T X < 1] \le \frac{\mathbb{E}[m]}{n+1}. \qquad \square$$

We can turn our mistake bounds into bounds on the empirical risk and risk achieved by the perceptron algorithm by choosing the loss function $loss(\langle w, x\rangle, y) = \mathbb{1}\{\langle w, x\rangle y < 1\}$. These bounds also imply bounds on the (empirical) risk with respect to the zero-one loss, since the prediction error is bounded by the number of margin mistakes.

## *Chapter notes*

Rosenblatt developed the perceptron in 1957 and continued to publish on the topic in the years that followed.[5,6] The perceptron project was funded by the US Office of Naval Research (ONR), who jointly announced the project with Rosenblatt in a press conference in 1958, that lead to the New York Times article we quoted earlier. This development sparked significant interest in perceptrons research throughout the 1960s.

The simple proof the mistake bound we saw is due to Novikoff.[3] Block is credited with a more complicated contemporaneous proof.[7] Minsky and Papert attribute a simple analysis of the convergence guarantees for the perceptron to a 1961 paper by Papert.[8]

Following these developments Vapnik and Chervonenkis proved the generalization bound for the perceptron method that we saw earlier, relying on the kind of stability argument that we will return to in our chapter on generalization. The proof of Theorem 2 is available in their 1974 book.[4] Interestingly, by the 1970s, Vapnik and Chervonenkis must have abandoned the stability argument in favor of the VC-dimension.

In 1969, Minksy and Papert published their influential book "Perceptrons: An introduction to computational geometry".[9] Among other results, it showed that perceptrons fundamentally could not learn certain concepts, like, an XOR of its input bits. In modern language, linear predictors cannot learn parity functions. The results remain relevant in the statistical learning community and have been extended in numerous ways. On the other hand, pragmatic researchers realized one could just add the XOR to the feature vector and continue to use linear methods. We will discuss such feature engineering in the next chapter.

The dominant narrative in the field has it that Minsky and Papert's book curbed enthusiasm for perceptron research and their multilayer extensions, now better known as deep neural networks. In an updated edition of their book from 1988, Minsky and Papert argue that work on perceptrons had already slowed significantly by the time their book was published for a lack of new results:

> One popular version is that the publication of our book so discouraged research on learning in network machines that a

> promising line of research was interrupted. Our version is that progress had already come to a virtual halt because of the lack of adequate basic theories, [. . . ].

On the other hand, the pattern recognition community had realized that perceptrons were just one way to implement linear predictors. Highleyman was arguably the first to propose empirical risk minimization and applied this technique to optical character recognition.[10] Active research in the 1960s showed how to find linear rules using linear programming techniques.[11] Work by Aizerman, Braverman and Rozonoer developed iterative methods to fit nonlinear rules to data.[12] All of this work was covered in depth in the first edition of Duda and Hart, which appeared five years after *Perceptrons*.

It was at this point that the artificial intelligence community first split from the pattern recognition community. While the artificial intelligence community turned towards more *symbolic* techniques in 1970s, work on statistical learning continued in Soviet and IEEE journals. The modern view of empirical risk minimization, of which we began this chapter, came out of this work and was codified by Vapnik and Chervonenkis in the 1970s.

It wasn't until the 1980s that work on pattern recognition, and with it the tools of the 1960s and earlier, took a stronger foothold in the machine learning community again.[13] We will continue this discussion in our chapter on datasets and machine learning benchmarks, which were pivotal in the return of pattern recognition to the forefront of machine learning research.

# *Bibliography*

[1] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[2] Michael J. Kearns, Robert E. Schapire, and Linda M. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3):115–141, 1994.

[3] Albert B. J. Novikoff. On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata*, pages 615–622, 1962.

[4] Vladimir Vapnik and Alexey Chervonenkis. *Theory of Pattern Recognition: Statistical Learning Problems*. Nauka, 1974. In Russian.

[5] Frank Rosenblatt. *Two Theorems of Statistical Separability in the Perceptron*. United States Department of Commerce, 1958.

[6] Frank Rosenblatt. *Principles of Neurodynamics: Perceptions and the Theory of Brain Mechanisms*. Spartan, 1962.

[7] Hans-Dieter Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34(1):123, 1962.

[8] Seymour A. Papert. Some mathematical models of learning. In *London Symposium on Information Theory*. Academic Press, New York, 1961.

[9] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 2017.

[10] Wilbur H. Highleyman. Linear decision functions, with application to pattern recognition. *Proceedings of the IRE*, 50(6):1501–1514, 1962.

[11] Olvi L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13(3):444–452, 1965.

[12] M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer. The Robbins-Monro process and the method of potential functions. *Automation and Remote Control*, 26:1882–1885, 1965.

[13] Pat Langley. The changing science of machine learning, 2011.