Chapter 1

Background on machine learning and learning theory

In this chapter we give a brief background on machine learning and statistical learning theory, which will give context for the development of deep learning and its goals.

1.1 Prediction versus estimation

Inspired by [5, Section I.A].

1.1.1 Introduction

Prediction versus estimation; correlation versus causation. When you hear these phrases in the context of machine learning, what do you think of? Maybe one thinks of the difference between classifying new data points and generating new data points. Or perhaps one considers that correlation is a symmetric assessment (e.g., if A is correlated with B, then B is correlated with A), but causation is directional (e.g., if A causes B, B does not necessarily cause A)[1].

These concepts are in some sense the difference between machine learning and statistics. In machine learning and prediction based tasks, we are often interested in developing algorithms that are capable of learning patterns from given data in an automated fashion, and then using these learned patterns to make predictions or assessments of newly given data. In many cases, our primary concern is the quality of the predictions or assessments, and we are less concerned about the underlying patterns that were learned in order make these predictions. Neural networks are, in some sense, the epitome of this point of view. In various contexts they are incredibly good at making predictions, but

[1]Thanks to Cullen Haselby and Bashir Sadeghi for these comments in class.

they are often referred to as "black box" methods due to the difficulty in understanding the model by which they make such predictions. For example, the most powerful convolutional neural networks are incredibly good at classifying natural images (sometimes even better than humans), but it is very difficult to understand the mechanisms by which they make such predictions.

In (classical) statistics and estimation, one is more concerned with the underlying model that makes the prediction. In other words, are the parameters of the model that makes the prediction statistically significant? Or could several other models (i.e., different parameter choices) have made the same prediction? This is the correlation versus causation issue. It comes up, for example and perhaps most notably, in medical trials and studies, in which one must not only find correlations and patterns in the data, but one must find the causal factors of a disease, so that one may develop and prescribe treatment.

1.1.2 Making things more precise with probability

Let us try to make this difference a bit more precise. To do so we will use the language of probability. Consider a given data set

$$T = \{(x_1, y_1), \ldots, (x_N, y_N)\},$$

consisting of data points $\{x_1, \ldots, x_N\} \subset \mathcal{X}$ and associated scalar valued labels $\{y_1, \ldots, y_N\} \subset \mathcal{Y}$. Let us assume that each data point x_i was sampled from $\mathcal{X}$ according to a probability distribution $\mathbb{P}_X$. This means, more precisely, we have a *probability space* $(\mathcal{X}, \Sigma, \mathbb{P}_X)$, which consists of:

- $\mathcal{X}$: The set of all possible outcomes, i.e., data points.
- Σ: The space of all possible events, i.e., collections of data. This is a set of sets, which has additional structure (see below).
- $\mathbb{P}_X$: The probability measure. For each event (set / collection of data points) $A \in \Sigma$, $\mathbb{P}_X(A)$ is the probability of the event A occuring.

The set Σ is a σ-algebra, meaning it has the following properties:

1. $\mathcal{X} \in \Sigma$.
2. If $A \in \Sigma$, then the complement of A, $A^c = \mathcal{X} \setminus A$, is also in Σ. Note this means $\emptyset \in \Sigma$.
3. If $A_1, A_2, \ldots$ are all in Σ, then their union is also in Σ, i.e.,

$$\bigcup_{i=1}^{\infty} A_i \in \Sigma.$$

Note that these properties also imply that if $A_1, A_2, \ldots$ are in Σ, then

$$\bigcap_{i=1}^{\infty} A_i \in \Sigma.$$

The probability measure $\mathbb{P}_X$ satisfies the following properties:

1. $\mathbb{P}_X(\mathcal{X}) = 1$.
2. Whenever $A_1, A_2, \ldots$ is a sequence of disjoint sets in Σ, then

$$\mathbb{P}_X\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mathbb{P}_X(A_i)\,.$$

From these properties we can also conclude that $\mathbb{P}_X(\emptyset) = 0$ and $\mathbb{P}_X(A^c) = 1 - \mathbb{P}_X(A)$.

Example 1.1. A simple example, that is not too relevant to our future discussions but which illustrates the idea, is the following. Consider flipping a coin twice, with the probability of heads being p and the probability of tails being q. There are four possible outcomes:

$$\mathcal{X} = \{HH\,, HT\,, TH\,, TT\}\,.$$

We also know the probabilities of each of these outcomes are p^2, pq, pq, and q^2, respectively. We thus set

$$\mathbb{P}_X(HH) = p^2\,, \mathbb{P}_X(HT) = \mathbb{P}_X(TH) = pq\,, \mathbb{P}_X(TT) = q^2\,. \tag{1.1}$$

This information is enough to define a probability space $(\mathcal{X}, \Sigma, \mathbb{P}_X)$, but it does not specify all the sets in Σ or their probabilities. Indeed, the event $A = \{HH\,, HT\}$ = "the first coin toss is a head," should be in Σ since it is the union of HH and HT. We assign A the probability $\mathbb{P}_X(A) = p^2 + pq = p$. In fact, the easiest way to ensure Σ is a σ-algebra is to include every subset of $\mathcal{X}$, including $\emptyset$ and $\mathcal{X}$ itself, and to assign probabilities using the rules of (1.1)[2].

Example 1.2. Another example, more relevant to our studies in this course, is inspired by the MNIST database of handwritten digits; see Figure 1.1. In this case $\mathcal{X}$ is the infinite set of all possible handwritten digits, and the σ-algebra Σ and the probability measure $\mathbb{P}_X$ are unknown to us, but we assume the training and testing set in the database are sampled according to $\mathbb{P}_X$, whatever it may be[3].

Since the data points x_i are randomly sampled, and the label y_i depends on x_i, the labels y_i are sampled from $\mathcal{Y}$ according to a probability distribution that encodes the dependence of y_i on x_i. We model this as a conditional probability distribution $\mathbb{P}_{Y|X}(B \mid X = x)$, which measures the probability of an event $B \subset \mathcal{Y}$ (i.e., a set of labels) given an outcome $x \in \mathcal{X}$. Together these two distributions induce a joint distribution $\mathbb{P}_{X,Y}$ on $\mathcal{X} \times \mathcal{Y}$, from which we draw the training samples.

[2] As pointed out by ???, in this case Σ is the power set of the set of outcomes.

[3] As pointed out by Cullen Haselby, if we assume that each image is of a fixed resolution and has a finite grayscale gradient, then the number of possible images is very large, but finite.

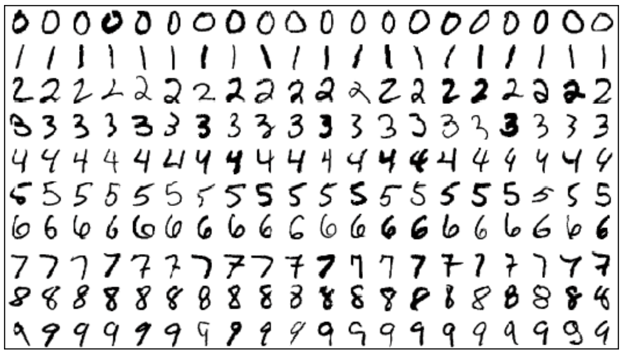


Figure 1.1: *Examples from the MNIST database of handwritten digits.*

In particular, suppose we draw the x_i's independently from $\mathbb{P}_X$. This means we "run the experiment" of drawing a point N times, each time independent from the others, and we obtain a random data point x_i. An analogy is flipping the coin, i.e. Example 1.1. Suppose we carry out the experiment of flipping the coin twice N times, each time independent from the other times. Then each time we will get a "data point," which corresponds to one of the four outcomes HH, HT, TH, TT, with the probabilities calculated earlier. Drawing a training sample (that is, a data point and a label) first entails drawing a point $x \in \mathcal{X}$ according to the distribution $\mathbb{P}_X$, and then drawing a point $y \in \mathcal{Y}$ according to $\mathbb{P}_{Y|X}(\cdot \mid X = x)$. When we want to think of the data point as being determined (say by an experiment, or a draw of a training point) we will write (x, y). On the other hand, when we want to think of the training point as a pair of random variables, one X taking values in $\mathcal{X}$ and the other Y taking values in $\mathcal{Y}$, we will write (X, Y).

Example 1.3. An example to keep in mind, that we will come back to later, is the following. Suppose $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$, and that a label $y_i \in \mathcal{Y}$ is generated from an underlying deterministic function $F : \mathbb{R}^d \to \mathbb{R}$ plus random noise:

$$y_i = F(x_i) + \varepsilon_i\,, \quad 1 \leq i \leq N\,.$$

Often we will assume that the ε_i are independently and identically distributed (i.i.d.) according to the normal distribution with mean zero and variance σ^2, i.e. $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$. In this case, if X is the random variable that takes values in $\mathbb{R}^d$ according to the probability distribution $\mathbb{P}_X$, and Y is the random variable that takes values in $\mathbb{R}$ according to $\mathbb{P}_{Y|X}(\cdot \mid X = x)$, then we have that

$$Y \sim \mathcal{N}(F(x), \sigma^2)\,,$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2. In other words, given that $X = x$, the label Y is a normal random variable with mean $F(x)$ and variance σ^2.

1.1.3 Maximum likelihood estimator

In order to make our analysis more concrete and precise, let us put the general probabilistic framework of the previous section in the more specific language of discrete and continuous random variables.

Let us begin with the discrete setting. In this case $\mathcal{X}$ is a finite, but possible very large set (e.g., see the footnote for the MNIST data set in Example 1.2), and $\mathcal{Y}$ is also a finite set, for example because we are doing classification and there are a finite number of classes (again, think of MNIST, there are 10 classes). In this case, there is a probability of drawing each individual point $x \in \mathcal{X}$ into our training set $T = \{(x_i, y_i)\}_{i=1}^N$. Let us call that probability $p_X(x)$, where we use X to denote the random variable that takes values in $\mathcal{X}$ according to $p_X(x)$. Note then, the probability of drawing a point from a subset $A \subseteq \mathcal{X}$ is (equivalently, the probability that X takes a value in A):

$$\mathbb{P}_X(X \in A) = \mathbb{P}_X(A) = \sum_{x \in A} p_X(x) \, .$$

Also note, if the x_i's in the training set are sampled independently from $\mathcal{X}$ according to $p_X(x)$, then the probability of getting that particular set $\{x_i\}_{i=1}^N$ is:

$$p_X\left(\{x_i\}_{i=1}^N\right) \propto \prod_{i=1}^N p_X(x_i) = p_X(x_1) \cdot p_X(x_2) \cdots p_X(x_N) \, ,$$

where the notation $\propto$ mean "proportional to." In particular, since we just care about the *set* $\{x_i\}_{i=1}^N$ and not the order in which it was drawn, the probability of drawing the set, $p_X(\{x_i\}_{i=1}^N)$, is higher than the right hand side. For example, if all the x_i's are unique (almost always the case in machine learning tasks), the correct factor is $N! = N \cdot (N-1) \cdots 2 \cdot 1$.

Recall the labels $\{y_i\}_{i=1}^N$ depend on their respective data points $\{x_i\}_{i=1}^N$. In this case, for each label $y \in \mathcal{Y}$, there is a probability of drawing y conditional on the data point being x. Let us call that probability $p_{Y|X}(y \mid x)$, where we use Y to denote the random variable that takes values in $\mathcal{Y}$ according to the conditional probabilities $p_{Y|X}(y \mid x)$. Note that if given x there is no ambiguity in the label y according to the underlying data generation process (not our model for the data!) (e.g., suppose $\mathcal{Y} = \{0, 1\}$ and for a specific x the label is always $y = 0$), then $p_{Y|X}(y \mid x)$ will be one when y is the correct label for x and zero otherwise. The probability of drawing a label from a subset $B \subseteq \mathcal{Y}$, given that our data point is $X = x$, is:

$$\mathbb{P}_{Y|X}(B \mid X = x) = \sum_{y \in B} p_{Y|X}(y \mid x) \, .$$

The joint probability of drawing the pair (x, y) is then:

$$p_{X,Y}(x, y) = p_X(x) p_{Y|X}(y \mid x)$$

which basically says, take the probability of drawing x and multiply it by the probability of drawing y given that we drew x. It follows that the probability of drawing the training set is:

$$p_{X,Y}(T) \propto \prod_{i=1}^{N} p_{X,Y}(x_i, y_i)\,,$$

assuming again that the x_i's are drawn independently from $\mathcal{X}$ according to p_X. Note that we can extend these notions to continuous random variables as well, e.g., if $\mathcal{X} = \mathbb{R}^d$ or $\mathcal{Y} = \mathbb{R}$; see Remark 1.4 below.

Now let us describe how to model $p_{X,Y}(x, y)$ given that our only information is a single training set T. We assume $T = \{(x_i, y_i)\}_{i=1}^{N}$ is sampled according to the joint probability density $p_{X,Y}(x, y)$. In theory the joint probability density $p_{X,Y}(x, y)$ can be used to sample many different realizations of the data set T (e.g., in the coin tosses of Example 1.1 in which we know how the data is generated), but in practice one often does not have access to $p_{X,Y}(x, y)$ and one must make due with the given, single data set T (e.g., the MNIST data base of Example 1.2). Our goal is to find a good model for $p_{X,Y}(x, y)$ given that all we know is T. We thus consider a hypothesis space of parameterized probability distributions

$$\mathcal{P} = \{p_{X,Y}(x, y \mid \theta) : \theta \in \mathbb{R}^n\}\,,$$

where $p_{X,Y}(T \mid \theta)$ is a model that describes the probability of observing the data T given the parameters θ. If the x_i's are drawn independently from $\mathcal{X}$, then

$$p_{X,Y}(T \mid \theta) \propto \prod_{i=1}^{N} p_{X,Y}(x_i, y_i \mid \theta)\,.$$

The vector $\theta \in \mathbb{R}^n$ encodes the parameters that determine the probabilistic model $p(T \mid \theta)$. These could be the parameters of a neural network, or some other class of machine learning algorithms such as linear models or kernel methods. Since T is the only data we have, our goal is to find the model, i.e. the parameters θ, that maximizes the probability of observing T:

$$\widehat{\theta} = \arg\max_{\theta \in \mathbb{R}^n} p(T \mid \theta)\,. \tag{1.2}$$

Thus given T and the hypothesis space $\mathcal{P}$, our best guess for the underlying joint probability density $p_{X,Y}(x, y)$ is $p_{X,Y}(x, y \mid \widehat{\theta})$. The probability density $p_{X,Y}(x, y \mid \widehat{\theta})$ is the *maximum likelihood estimate* for the probability density $p_{X,Y}(x, y)$. We will come back to this later.

In machine learning tasks, one is often interested in the accuracy of $p_{X,Y}(x, y \mid \widehat{\theta})$ relative to $p_{X,Y}(x, y)$; that is, what is the predictive power of $p_{X,Y}(x, y \mid \widehat{\theta})$? In statistics and estimation tasks, one is concerned with the accuracy of $\widehat{\theta}$, the parameters of the model. In particular, is it significant that these particular parameters generate the optimal model from the hypothesis class $\mathcal{P}$? In either case, there are (in the current formulation) two considerations that influence the quality of the model. The first is one's ability to solve for $\widehat{\theta}$. This is not always possible, particularly in deep learning. Studying this problem amounts to studying how to optimize the parameters θ of a deep network. The second consideration is the hypothesis space $\mathcal{P}$. In particular, is it expressive enough to contain a model $p_{X,Y}(x, y \mid \theta)$ that is an accurate estimator for $p_{X,Y}(x, y)$, and furthermore can we find such a model with a finite amount of training data? In machine learning generally, this is the problem of model class selection, e.g., should we use a linear model, a quadratic model, kernel methods, or deep learning? Within deep learning, this may refer to a number of choices having to do with the architecture and design of the network, including the number of layers, the width of each layer, but also more nuanced choices such as how convolutional neural networks leverage additional structure in the data points x when x is an image.

This is a course on the mathematics of deep learning. Deep learning is, in the vast majority of cases, used for machine learning and prediction. However, in studying the mathematics of deep learning, we will attempt to understand which types of models and model classes yield good predictors $p_{X,Y}(x, y \mid \widehat{\theta})$. In certain cases this may help in understanding the role and significance of the specific $\widehat{\theta}$, although we will most likely not directly address this type of consideration; for more information in that direction, one should investigate causal inference in machine learning.

Remark 1.4. We can extend the framework at the beginning of this section to continuous random variables. In particular suppose that $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$. If X is a continuous random variable that takes values in $\mathcal{X} = \mathbb{R}^d$, then there exists a probability density function $p_X(x)$ such that

$$\mathbb{P}_X(A) = \int_A p_X(x)\, dx\,, \quad A \subseteq \mathcal{X} = \mathbb{R}^d\,.$$

Furthermore, suppose that Y conditioned on $X = x$ is a continuous random variable, which means there is a probability density function $p_{Y|X}(y \mid x)$ such that

$$\mathbb{P}_{Y|X}(B \mid X = x) = \int_B p_{Y|X}(y \mid x)\, dy\,, \quad B \subseteq \mathcal{Y} = \mathbb{R}\,.$$

The joint probability density function of X, Y is

$$p_{X,Y}(x, y) = p_X(x) p_{Y|X}(y \mid x)\,,$$

which means that the probability of A and B occurring is:

$$\mathbb{P}_{X,Y}(A, B) = \int_A \int_B p_{X,Y}(x, y)\, dy\, dx = \int_A \int_B p_X(x) p_{Y|X}(y \mid x)\, dy\, dx\,.$$

If Y is discrete, meaning without loss of generality that $\mathcal{Y} = \{1, \ldots, M\}$, we can amend the previous discussion as follows. In this case the part concerning X remains the same but for each of the M possible values of Y, we have a probability that Y takes the value conditioned on X:

$$\mathbb{P}_{Y|X}(Y = y \mid X = x) = p_{Y|X}(y \mid x)\,, \quad y \in \mathcal{Y} = \{1, \ldots, M\}\,,$$

In this case the joint probability density function is

$$p_{X,Y}(x,y) = p_X(x)p_{Y|X}(y \mid x)\,,$$

and the probability of $A \subseteq \mathcal{X} = \mathbb{R}^d$ and $B \subseteq \mathcal{Y} = \{1,\ldots,M\}$ occuring is

$$\mathbb{P}_{X,Y}(A,B) = \int_A \sum_{y\in B} p_{X,Y}(x,y)\,dx = \int_A \sum_{y\in B} p_X(x)p_{Y|X}(y \mid x)\,dx\,.$$

1.1.4 Supervised vs unsupervised learning

Let us clarify a bit the difference between supervised learning and unsupervised learning in this probabilistic framework that we have developed. In unsupervised learning we are only given data points $\{x_i\}_{i=1}^N \subseteq \mathcal{X}$ and we are interested in extracting patterns from the data or generating new data points; often this means our primary concern is estimating $p_X(x)$. In supervised learning, on the other hand, as we have already described we are given a training set $T = \{(x_i, y_i)\}_{i=1}^N$ consisting of data points $\{x_i\}_{i=1}^N \subset \mathcal{X}$ and labels $\{y_i\}_{i=1}^N \subset \mathcal{Y}$. Our primary concern is, given a new data point x, our ability to estimate its label $y(x)$.

Given the discussion in Sections 1.1.2 and 1.1.3, it is natural to model our candidate probability density functions $p_{X,Y}(x,y \mid \theta)$ by conditioning on x:

$$p_{X,Y}(x,y \mid \theta) = p_X(x \mid \theta)p_{Y|X}(y \mid x,\theta)\,.$$

Recall from (1.2) we want to maximize $p_{X,Y}(T \mid \theta)$ over all choices of $\theta \in \mathbb{R}^n$. We have:

$$\widehat{\theta} = \operatorname*{arg\,max}_{\theta\in\mathbb{R}^n} p(T \mid \theta) = \operatorname*{arg\,max}_{\theta\in\mathbb{R}^n} \prod_{i=1}^N p(x_i, y_i \mid \theta)\,,$$

since the right hand side is proportional to $p(T \mid \theta)$. Notice as well that we can take the logarithm, and still obtain the same $\widehat{\theta}$, that is:

$$\begin{aligned}\widehat{\theta} &= \operatorname*{arg\,max}_{\theta\in\mathbb{R}^n} \sum_{i=1}^N \log p(x_i, y_i \mid \theta) \\ &= \operatorname*{arg\,max}_{\theta\in\mathbb{R}^n} \left(\sum_{i=1}^N \log p_X(x_i \mid \theta) + \sum_{i=1}^N \log p_{Y|X}(y_i \mid x_i, \theta) \right)\,. \end{aligned} \tag{1.3}$$

Now in unsupervised learning, there are no labels, and so we only have the first summation in (1.3), and indeed maximizing that summation will give us the maximum likelihood estimator for $p_X(x)$. In generative modeling, we can then sample from $p_X(x \mid \widehat{\theta})$ to generate new data points. In supervised learning, remember we are primarily interested in the problem of given a new data point x, coming up with an accurate estimate for its label $y(x)$. This is encoded by the second summation, and so we often discard the first summation in this case. We will come back to this after we discuss the functional modeling perspective of supervised learning next in Section 1.2.

1.2 The supervised learning recipe

1.2.1 Functional models

We briefly describe the basic steps involved in supervised learning from a functional modeling perspective.

In supervised learning one is given a set of data that is partitioned into a training set $T = \{(x_i, y_i)\}_{i=1}^N$, consisting of data points $\{x_i\}_{i=1}^N \subset \mathcal{X}$ and associated labels $\{y_i\}_{i=1}^N \subset \mathcal{Y}$ that one is able to use to fit a model, and a test set T_{test} consisting of other data points and labels (x, y) not in the training set, and which are not to be used to fit the model but rather are to be used to evaluate the quality of the model fitted to the training data. Analogous to the hypothesis class $\mathcal{P}$, one defines a parameterized model class $\mathcal{F}$

$$\mathcal{F} = \{f(x;\theta) : \theta \in \mathbb{R}^n\}$$

consisting of candidate models $f(x;\theta)$ that are parameterized by $\theta \in \mathbb{R}^n$. For example, the class of linear models is given by:

$$\mathcal{F}_{\text{linear}} = \{f(x;\theta) = \langle x, \theta \rangle : \theta \in \mathbb{R}^d\},$$

where $\langle x, \theta \rangle$ is the standard dot product,

$$\langle x, \theta \rangle = \sum_{k=1}^{d} x(k)\theta(k).$$

One also defines a loss function $\ell(y, f(x))$ that measures the cost of a model $f(x)$ differing from a label $y = y(x)$; almost always we will take $\ell(y, f(x)) = |y - f(x)|^2$, the squared loss.

To select a model from the model class $\mathcal{F}$, we minimize the average loss over the training set:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, f(x_i;\theta))$$

$$\widehat{\theta} = \arg\min_{\theta \in \mathbb{R}^n} \mathcal{L}(\theta) = \arg\min_{\theta \in \mathbb{R}^n} \left[\frac{1}{N}\sum_{i=1}^{N} \ell(y_i, f(x_i;\theta)) \right].$$

We then evaluate the quality of the selected model, $f(x;\widehat{\theta})$, by evaluating it on the test set and computing the average loss over the test set:

$$\text{average test error} = \frac{1}{|T_{\text{test}}|} \sum_{(x,y) \in T_{\text{test}}} \ell(y, f(x;\widehat{\theta})).$$

Similarly to the discussion in Section 1.1, the quality of the arrived upon model depends on two points. The first is, can one solve for the parameters $\widehat{\theta}$ or some other parameters θ that are nearly as good? Second, the choice of model class $\mathcal{F}$ is incredibly important, as it determines the set of possible models from which we will select one. There needs to be a model in $\mathcal{F}$ that simultaneously fits the training data T while at the same time does not overfit to spurious patterns in the training set so that it generalizes well to the test set. This is a complicated problem because for each new data set or task, the underlying distribution $p_{Y|X}(y \mid x)$ will change and the relationship between data point x and label $y(x)$ will thus change. We will thus want algorithms that are able to adapt to a multitude of scenarios, but which also do not need too many training points to find the (near) optimal model.

1.2.2 Linking the probabilistic model with the functional model

Let us now link together the probabilistic models described in Section 1.1.3 and the functional models described in Section 1.2. To do so, recall Example 1.3 in which we assumed that given a day point x, a label y is generated according to

$$y = F(x) + \varepsilon \,, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2) \,. \tag{1.4}$$

In other words, there is a deterministic function $F : \mathcal{X} \to \mathbb{R}$ that is the "true" label, but it is corrupted by a noise ε and we observe the corrupted version. That is, given x, the label of x is sampled from $\mathcal{N}(F(x), \sigma^2)$, which is the normal distribution with mean $F(x)$ and variance σ^2. In this case, we have

$$p_{Y|X}(y \mid x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-|y - F(x)|^2 / 2\sigma^2} \,,$$

since the right hand size is the probability density function for $\mathcal{N}(F(x), \sigma^2)$.

Let us now consider a functional model class $\mathcal{F} = \{f(x \mid \theta) : \theta \in \mathbb{R}^n\}$ that contains our best guesses for F, which we do not know. Let us take our hypothesis space $\mathcal{P}_{Y|X}$ as:

$$\mathcal{P}_{Y|X} = \{p_{Y|X}(y \mid x, \theta) : \theta \in \mathbb{R}^n\} \,, \quad p_{Y|X}(y \mid x, \theta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-|y - f(x;\theta)|^2 / 2\sigma^2} \,.$$

Using (1.3) and the discussion thereafter, we have

$$\begin{aligned}
\widehat{\theta} &= \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} \sum_{i=1}^{N} \log p_{Y|X}(y_i \mid x_i, \theta) \\
&= \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} \left(-N \log \sqrt{2\pi}\sigma - \frac{1}{2\sigma^2} \sum_{i=1}^{N} |y_i - f(x_i; \theta)|^2 \right) \\
&= \operatorname*{arg\,min}_{\theta \in \mathbb{R}^n} \frac{1}{N} \sum_{i=1}^{N} |y_i - f(x; \theta_i)|^2 \\
&= \operatorname*{arg\,min}_{\theta \in \mathbb{R}^n} \mathcal{L}(\theta) \,.
\end{aligned}$$

In other words, the two formulations are equivalent, at least for the model (1.4). We will come back to this correspondence again when we discuss regularization.

Notice that even if the model of Example 1.3 does not hold, minimizing the squared error over the functional model class is still not a bad idea. Indeed, we see that doing so implicitly puts a normal probability distribution with mean $f(x;\widehat{\theta})$ over the possible labels for x. This will allow us to give probabilities of each label, which in turn can allow one to estimate uncertainty.

1.3 Two methods

We describe two basic methods for machine learning, linear regression and k-nearest neighbors, which will serve as a basis for further discussion. Linear regression is a simple class of models that are easy to fit with a few training points, but which will not fit labels $y(x)$ that depend non-linearly on the data x. On the other hand, k-nearest neighbors is a flexible class of models that can fit many different patterns, but which suffers from the curse of dimensionality and thus needs many training points as the dimension of x increases.

1.3.1 Linear regression

Let us now consider a more concrete scenario, linear regression. In the language of Section 1.2.1, linear regression models form the class of affine functions over x, i.e., in this section we consider:

$$\mathcal{F} = \left\{ f(x;\theta) = \theta(0) + \sum_{k=1}^{d} \theta(k)x(k) : \theta \in \mathbb{R}^{d+1} \right\} .$$

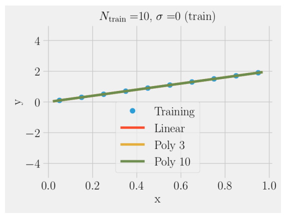
Linear regression models are affine over x but are linear over the augmented data point $(1, x) \in \mathbb{R}^{d+1}$. The extra parameter $\theta(0)$ is often referred to as the bias.

Given a training set $T = \{(x_i, y_i)\}_{i=1}^{N}$ we seek to find a linear model that best fits the data by minimizing the squared loss:

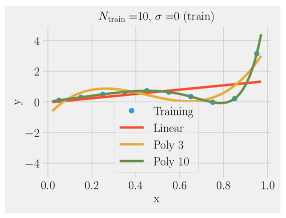
$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \langle (1, x_i), \theta \rangle)^2$$

We can solve for the minimum of $\mathcal{L}(\theta)$ analytically. To do so we use matrix notation. Define the $(d+1) \times N$ matrix $\mathbf{X}$ and the $N \times 1$ vector $\mathbf{y}$ as:

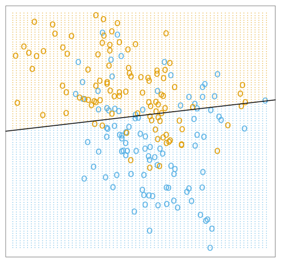
$$\mathbf{X} = \begin{pmatrix} 1 & \cdots & 1 \\ | & & | \\ x_1 & \cdots & x_N \\ | & & | \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} .$$



(a) *Data generated from a model in which $y(x)$ is an affine function of $x \in \mathbb{R}$. The linear regression model fits the data perfectly.*



(b) *Data generated from a model in which $y(x)$ is a 10th order polynomial of $x \in \mathbb{R}$. The linear regression model does not fit the data perfectly, but the 10th order polynomial regression model does.*



(c) *Two dimensional data with two classes, orange (+1) and blue (-1). The thick black line is the level line of $f(x;\widehat{\theta}) = \langle (1,x), \widehat{\theta} \rangle = 0$. Points above the level line are classified as orange; points below the level line are classified as blue.*

Figure 1.2: *Examples of linear regression and classification. Figures (a) and (b) taken from [5]; figure (c) taken from [6].*

We can the rewrite $\mathcal{L}(\theta)$ as

$$\mathcal{L}(\theta) = N^{-1}(\mathbf{y} - \mathbf{X}^T\theta)^T(\mathbf{y} - \mathbf{X}^T\theta)\,,$$

where we have considered $\theta \in \mathbb{R}^{d+1}$ as a $(d+1) \times 1$ vector. Differentiating $\mathcal{L}(\theta)$ and setting it equal to $\mathbf{0}$ (the vector of zeros), one obtains:

$$\nabla_\theta \mathcal{L}(\theta) = N^{-1}\mathbf{X}(\mathbf{y} - \mathbf{X}^T\theta) = \mathbf{0} \quad \Longrightarrow \quad \mathbf{X}(\mathbf{y} - \mathbf{X}^T\theta) = \mathbf{0}\,.$$

Solving for θ one obtains, assuming $\mathbf{X}\mathbf{X}^T$ is invertible,

$$\widehat{\theta} = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}\,. \tag{1.5}$$

The solution $\widehat{\theta}$ is the set of optimal weights that fits the training data, regardless of whether the relationship between x and $y(x)$ is affine. When indeed $y(x)$ is an affine function of x and $\mathbf{X}\mathbf{X}^T$ is invertible (necessarily then, $N \geq p$), the model $\widehat{\theta}$ will not only fit the data but will be the true underlying model. Figure 1.2 illustrates both scenarios.

Regularization

Model overfit occurs when we use too complex of a model to fit the training data, and thus fit spurious patterns from noise or other nuisance factors that reduce the ability of the model to generalize to new data (e.g. test points).

For example, in Figures 1.2a and 1.2b, training data $T = \{(x_i, y_i)\}_{i=1}^N$ is generated from a linear model and a 10th order polynomial model. In both cases a 10th order polynomial model is fit to the data, with success. When $\mathcal{X} \subseteq \mathbb{R}$ an $(n-1)^{\text{st}}$ order polynomial function class consists of models

$$f(x;\theta) = \sum_{k=0}^{n-1} \theta(k) x^k . \tag{1.6}$$

If one again uses the squared loss to evaluate the quality of the model, i.e.,

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \sum_{k=0}^{n-1} \theta(k) x_i^k \right)^2 , \tag{1.7}$$

then one can still use (1.5) to solve for $\widehat{\theta} = \arg\min_{\theta \in \mathbb{R}^n} \mathcal{L}(\theta)$ by redefining $\mathbf{X}$ as

$$\mathbf{X}^T = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^{n-1} \end{pmatrix} .$$

This is a particular form of a dictionary model, consisting of the new representation $\Phi(x) = (\phi_k(x))_{k=0}^{n-1} \in \mathbb{R}^n$ with $\phi_k(x) = x^k$. Linear regression over $\Phi(x)$ consists of models $f(x;\theta) = \langle \Phi(x), \theta \rangle$, which is exactly (1.6).

As n increases the complexity of the polynomial functional class increases, as it contains all polynomial models of order less than $n-1$ as well. However, fitting more complex models can become more delicate in the presence of noise or other confounding factors, as the added complexity can allow the models to fit the noise, which is not desirable. This is not shown in Figures 1.2a and 1.2b because there is no noise in the data. Figure 1.3 illustrates the point, as the data in this figure is generated from a linear model, but the labels y_i are corrupted by a small amount of additive white noise. In this case, fitting the data with a linear model and a third order polynomial model yields good interpolative and extrapolative models, but models fit with higher order polynomials such as 10th order overfit to the noise and cannot extrapolate and even do worse in interpolation.

This example may be a bit disheartening as it would seem to indicate that we need to know the appropriate complexity of the model class in advance, which is often not possible. However, we must remember that 10th order polynomials include 3rd order polynomials and 1st order polynomials. The loss function that we minimized, in this case (1.7), however selected an overly complex model because it minimized the mean squared error on the noisy training data. The question then becomes, how can we use a model class $\mathcal{F}$, such as 10th order polynomials, that includes complex models for when we need them (as in Figure 1.2b) but from which we can also draw a less complex model when the situation calls for it (as in Figure 1.3)? One answer to this question is *regularization*. In this case the loss function is amended to incorporate a regularization function that acts on the parameters θ, in many cases restricting the parameter

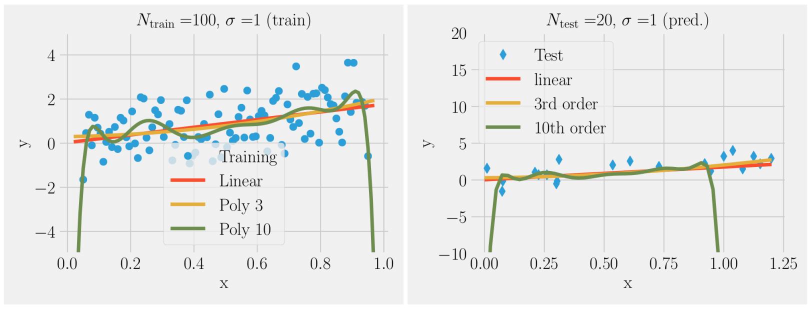


Figure 1.3: *Linear and polynomial models fit to noisy data generated from a linear model plus noise. Left panel: Models fit to training data with $x_i \in [0,1]$. Right panel: Models evaluated on test data with test points $x \in [0, 1.25]$. The linear and 3^{rd} order polynomials fit the data well and do not fit the noise, and thus are capable of extrapolating to test data outside the range of the training data. The 10^{th} order polynomial, however, fits spurious patterns in the noise and thus interpolates with a less regular model and cannot extrapolate. Figure taken from [5].*

set:

$$\mathcal{L}_{\mathcal{R}}(\theta, \lambda) = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, f(x_i; \theta)) + \lambda \mathcal{R}(\theta) \,.$$

The hyper-parameter λ balances the fit to the training data (the first term) with the strength of regularization on the parameters θ induced through the regularizer $\mathcal{R}(\theta)$. Setting $\lambda = 0$ leads to a high complexity model being selected from the model class $\mathcal{F}$, whereas larger values of λ increasingly restrict the viable parameters θ, thus reducing the complexity of the model $\widehat{\theta}$ that minimizes $\mathcal{L}_{\mathcal{R}}(\theta)$.

In linear and polynomial regression with a squared loss $\ell(y, f(x)) = |y - f(x)|^2$, there are two common choices for $\mathcal{R}(\theta)$. The first is ridge regression, which uses an ℓ^2 regularization:

$$\mathcal{R}_2(\theta) = \|\theta\|_2^2 = \sum_{k=1}^{n} |\theta(k)|^2 \,.$$

Ridge regression models, e.g. the following in the case of linear regression,

$$\widehat{\theta}_{\text{ridge}} = \arg\min_{\theta} \left[\frac{1}{N} \sum_{i=1}^{N} |y_i - \langle (1, x_i), \theta \rangle|^2 + \lambda \sum_{k=0}^{d} |\theta(k)|^2 \right] ,$$

penalize very large parameters $\theta(k)$ via the ℓ^2 regularization, and thus result in models that have a few large parameters $\theta(k)$, with the remaining parameters being small but rarely zero[4]. The larger λ, the fewer larger parameters and the more small parameters in θ. More precisely, one can show that

$$\widehat{\theta}_{\text{ridge}} = (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1}\mathbf{X}\mathbf{y}\,,$$

indicating that the size of the weights are depressed, approximately, by a factor of $(1+\lambda)$. In fact this is exactly the case if the rows of $\mathbf{X}$ are orthonormal, since in that case $\mathbf{X}\mathbf{X}^T = \mathbf{I}$.

The second common regularization is LASSO (least absolute shrinkage and selection operator), which uses an ℓ^1 regularization:

$$\mathcal{R}_1(\theta) = \|\theta\|_1 = \sum_{i=1}^{n} |\theta(i)|\,.$$

LASSO models, e.g. the following in the case of linear regression,

$$\widehat{\theta}_{\text{LASSO}} = \arg\min_{\theta} \left[\frac{1}{N} \sum_{i=1}^{N} |y_i - \langle (1, x_i), \theta \rangle|^2 + \lambda \sum_{i=0}^{d} |\theta(i)| \right],$$

penalize non-zero parameters $\theta(i)$ via the ℓ^1 regularization, and thus result in sparse models. With larger λ there are fewer non-zero parameters and the model increases in sparsity. In fact, when the columns of $\mathbf{X}$ are orthogonal, one obtains:

$$\widehat{\theta}_{\text{LASSO}}(k) = \operatorname{sign}(\widehat{\theta}_{\lambda=0}) \max(|\widehat{\theta}_{\lambda=0}(k)| - \lambda, 0)\,,$$

where $\widehat{\theta}_{\lambda=0}$ is the simple least squares optimal model with no regularization (i.e., $\lambda = 0$). The above formula shows that the ℓ^1 regularization of LASSO shrinks the weights by an additive factor of $-\lambda$, down to zero, thus resulting in sparse models.

One additional point that is important to remember is that both ridge and LASSO regularized regressions implicitly define new, restricted model subclasses of the linear regression class. In particular, for each $\lambda > 0$ there exists a $t = t(\lambda) < \infty$ such that

$$f(\cdot; \widehat{\theta}_{\text{ridge}}) \in \mathcal{F}_{2,t} = \{ f(x;\theta) = \langle (1,x), \theta \rangle : \|\theta\|_2 \leq t \}$$

and

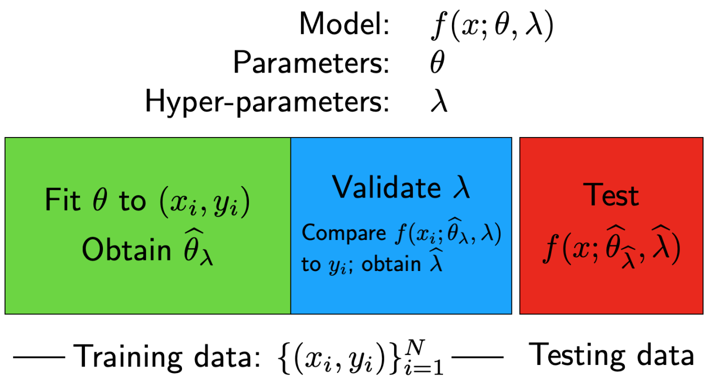
$$f(\cdot; \widehat{\theta}_{\text{LASSO}}) \in \mathcal{F}_{1,t} = \{ f(x;\theta) = \langle (1,x), \theta \rangle : \|\theta\|_1 \leq t \}\,.$$

Thus regularization implicitly defines a new model class $\mathcal{F}_{\lambda,\mathcal{R}} \subseteq \mathcal{F}$ that depends on the regularizer $\mathcal{R}$ and the strength of the regularization λ.

In practice, one must estimate the hyper-parameter λ using only the training data. To do so, one employs *cross validation*. Figure 1.4 describes the idea.

When the label function $y(x)$ is not an affine function of x, a nonlinear model is required. One option, also discussed in this section, are polynomial

[4]It makes a lot of sense to have the penalty start at $k = 1$, thus omitting the bias $\theta(0)$. However, this choice will complicate our analysis, so we will not pursue it further. Thanks to Cullen Haselby for pointing this out.



Model: $f(x;\theta,\lambda)$
Parameters: θ
Hyper-parameters: λ
Fit θ to (x_i, y_i)
Obtain $\widehat{\theta}_\lambda$
Validate λ
Compare $f(x_i;\widehat{\theta}_\lambda,\lambda)$ to y_i; obtain $\widehat{\lambda}$
Test
$f(x;\widehat{\theta}_{\widehat{\lambda}},\widehat{\lambda})$
Training data: $\{(x_i,y_i)\}_{i=1}^N$
Testing data

Figure 1.4: *Cross validation. A model $f(x;\theta,\lambda)$ consists of parameters θ and hyper-parameters λ. The training set $T = \{(x_i,y_i)\}_{i=1}^N$ is partitioned into two sets, green and blue. The green set is used to fit the parameters θ, for a fixed λ, to the data pairs (x_i,y_i) in this set. The blue set is used to validate the model, meaning one evaluates $f(x_i,\widehat{\theta}_{\widehat{\lambda}},\lambda)$ against the true label y_i for (x_i,y_i) in the green set. One does this for a finite number of λ and selects the optimal $\widehat{\lambda}$ based on which one has the smallest validation error. Finally, the model $f(x;\widehat{\theta}_{\widehat{\lambda}},\widehat{\lambda})$ is tested on different data (x,y) in the test set.*

models. The number of parameters in a polynomial model, though, scales poorly in the dimension. If $n-1$ is the order of the polynomial and d is the dimension, then $\#(\theta) = O(d^n)$. Kernel methods and in particular polynomial kernels remedy this. However, not every label function $y(x)$ is a global polynomial of the data points x. Sometimes the label function is based on locality or other nonlinear patterns that are not so easily modeled by compact mathematical formulas. Section 1.3.3 describes k-nearest neighbor models, which are based on local similarities and make very few assumptions about the way the data is generated. First though we return to the probabilistic models and interpret regularization from this perspective.

1.3.2 Bayes and maximum a posteriori

Recall from Section 1.1.3 we defined maximum likelihood estimation (MLE) as:

$$\widehat{\theta}_{\text{MLE}} = \arg\max_{\theta\in\mathbb{R}^n} p_{X,Y}(T \mid \theta) = \arg\max_{\theta\in\mathbb{R}^n} p(T \mid \theta)\,,$$

where we recall $T = \{(x_i,y_i)\}_{i=1}^N$ is our training set. In other words, we maximized the probability of observing the particular training set T that we have, given the model parameters θ. On the other hand, a more intuitive and perhaps useful object might be $p(\theta \mid T)$, the probability of the model θ being correct given the training set T. Indeed, this is how we generally think of machine learning. We are given a training set T and we pick a model θ that has the highest chance of describing the data. This is called the *maximum a posteriori* estimator:

$$\widehat{\theta}_{\mathrm{MAP}} = \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} p(\theta \mid T)\,. \tag{1.8}$$

The question then becomes, how do we compute $\widehat{\theta}_{\mathrm{MAP}}$? For this, we can use Bayes' Theorem:

$$p(\theta \mid T) \propto p(T \mid \theta) p(\theta)\,.$$

Notice that right hand side contains the term $p(T \mid \theta)$, which is what we used to compute the MLE model. But it also contains another, new term, $p(\theta)$. What is this term? It is referred to as the *prior distribution* on the parameters θ. It encodes any prior knowledge we have about our model, or behavior that we want to build into our model. For example, we can place a Gaussian/normal prior on θ, meaning that we assume each parameter $\theta(k)$ is distributed according to the normal distribution with mean zero and variance proportional to λ^{-1}:

$$p(\theta) = p(\theta \mid \lambda) = \prod_{k=1}^{n} \sqrt{\frac{\lambda}{2\pi}} e^{-\lambda |\theta(k)|^2} \quad \text{(normal prior)}\,.$$

We can also use a Laplace prior:

$$p(\theta) = p(\theta \mid \lambda) = \prod_{k=1}^{n} \frac{\lambda}{2} e^{-\lambda |\theta(k)|} \quad \text{(Laplace prior)}\,.$$

Let us now see how the $\widehat{\theta}_{\mathrm{MAP}}$ model is related to regularized functional models, and in particular ridge regression for the normal prior and LASSO for the Laplace prior. Using (1.8) we have:

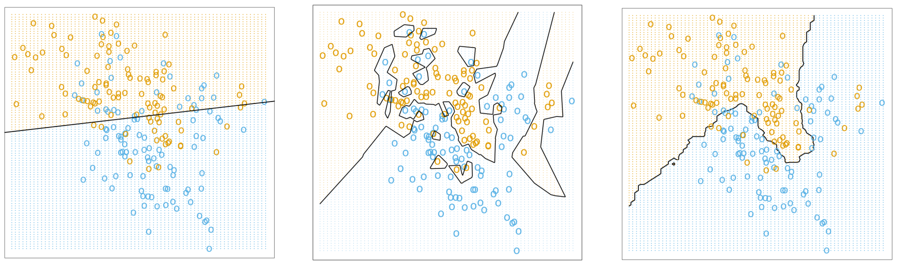
$$\begin{aligned}
\widehat{\theta}_{\mathrm{MAP}} &= \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} p(\theta \mid T) \\
&= \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} p(T \mid \theta) p(\theta) \\
&= \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} (\log p(T \mid \theta) + \log p(\theta))
\end{aligned}$$

Recall from (1.3) that if the $\{x_i\}_{i=1}^N$ are sampled iid (independently and identically distributed) from $\mathcal{X}$, then we can decompose $\log p(T \mid \theta)$ as:

$$\widehat{\theta}_{\mathrm{MAP}} = \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} \left(\sum_{i=1}^{N} \log p_X(x_i \mid \theta) + \sum_{i=1}^{N} \log p_{Y|X}(y_i \mid x_i, \theta) + \log p(\theta) \right)\,.$$

Furthermore, in the case of supervised learning we will often discard the first term involving p_X, and, as we saw in Section 1.2.2, taking

$$p_{Y|X}(y \mid x, \theta) = e^{-|y - f(x;\theta)|^2 / 2\sigma^2}\,,$$



(a) *Linear regression* **(b)** *k-nearest neighbors with $k = 1$* **(c)** *k-nearest neighbors with $k = 15$*

Figure 1.5: *Two dimensional data with two classes, orange (+1) and blue (-1). Three different classifiers and their respective decision boundaries. Figure taken from [6].*

is a reasonable choice. We then have:

$$\widehat{\theta}_{\text{MAP}} = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^n} \left(\frac{1}{2\sigma^2} \sum_{i=1}^{N} |y_i - f(x_i;\theta)|^2 - \log p(\theta) \right) .$$

We thus see that $\log p(\theta)$ serves as the regularization term on the parameters θ. In particular, if we use the Gaussian/normal prior, we obtain:

$$\widehat{\theta}_{\text{MAP}} = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^n} \left(\frac{1}{2\sigma^2} \sum_{i=1}^{N} |y_i - f(x_i;\theta)|^2 + \lambda \sum_{k=1}^{n} |\theta(k)|^2 \right) ,$$

which is precisely ridge regression if $f(x;\theta) = \langle (1,x), \theta \rangle$. Similarly, if we use the Laplace prior, we obtain LASSO.

1.3.3 *k*-Nearest neighbors and local methods

Figure 1.2c illustrates the need for a nonlinear classifier, capable of learning a nonlinear decision boundary as opposed to a straight line. A polynomial method could do better, but this still imposes a modeling assumption on the underlying data generation process. In order to avoid such assumptions, we instead turn to another type of method, *k-nearest neighbors*, which is a local method.

Let $N_k(x)$ denote k nearest of neighbors of x, in the Euclidean distance, from the set of training points $\{x_1, \ldots, x_N\} \subset \mathbb{R}^d$. The k-nearest neighbor model is:

$$f(x;k) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i .$$

Figure 1.5 illustrates the model for $k = 1$ and $k = 15$, and compares to the linear classifier from Figure 1.2c. In the figure we see that the k-nearest neighbors classifiers make significantly fewer mis-classifications on the training set than the linear regression classifier. However, classification rate on the training set is not the sole criterion for the quality of a model, as we saw in our discussion on regularization. Indeed models must generalize well to unseen points. The $k = 1$ nearest neighbor model in fact makes no mis-classifications on the training set, since it simply assigns to each training point its own label. But the decision boundary of the 1-nearest neighbor classifier is extremely irregular and likely to make mistakes on test data. The $k = 15$ nearest neighbor classifier, on the other hand, has a semi-regular boundary that is likely to generalize better than either the 1-nearest neighbor classifier (because it is too rough) or the linear regression classifier (because it is too smooth). This discussion is hinting at the bias-variance tradeoff in machine learning, which we will make more precise in Section 1.4. It also illustrates that the $k = 1$ model is, in some sense, more complex than the $k = 15$ model. In the extreme case, $k = N$, all points are classified the same, and so this is the simplest model. In fact, even though there is only one parameter, k, that we set, the effective number of parameters or complexity of the k-nearest neighbor model is N/k.

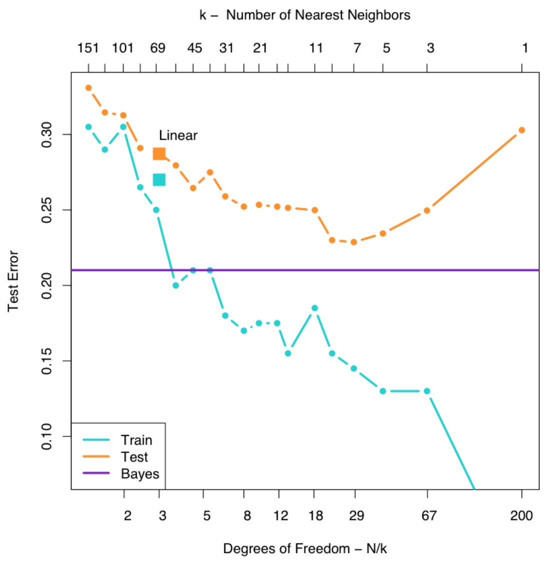


Figure 1.6: *k-nearest neighbors classifier for k decreasing from left to right, evaluated on the training set and the test (validation) set. The models with k* ≈ 10 *perform the best on the test set, even though the training error is essentially decreasing as k* $\to 1$. *Figure taken from [6].*

Note that because the 1-nearest neighbor classifier will always make zero errors on the training set, we require a different method by which to select k. We instead use cross-validation, which requires a validation set separate from the training set, but for which we still know the correct labels. In fact, many hyper-parameters, including k in k-nearest neighbors and λ in regularized linear models, are selected through cross-validation. The way it works for k-nearest neighbors is that we use the original training set T to define the neighborhoods of each point and the model $f(x;k)$, but we then evaluate this model on the validation set and compute the average error on the validation set. The k with the lowest average error on the validation set is the model we use on the test set. See Figure 1.6 for an illustration using k-nearest neighbors. The more general principle, which includes k-nearest neighbors and the regularized linear regularization, is based on the bias variance tradeoff, which will be discussed in the next section.

1.4 Basics of statistical learning theory

Figure 1.6 illustrates an important concept in machine learning, which is the bias-variance tradeoff. Later in Section 1.4.2 we will discuss this in more detail. First, though, in Section 1.4.1 we examine k-nearest neighbors and linear regression from a statistical point of view. We will also derive the naive Bayes classifier for classification, which will explain the purple line in Figure 1.6. Then in Section 1.4.3 we will discuss the curse of dimensionality.

1.4.1 Statistical view of models

We now consider k-nearest neighbors and linear regression from the perspective of statistical learning theory. Let us return to the assumptions of Section 1.1.3. In particular, recall that we assume we have a probability space $(\mathcal{X}, \Sigma, \mathbb{P}_X)$, $\mathcal{X} = \mathbb{R}^d$, with probability density function $p_X(x)$. We also have the label set $\mathcal{Y}$, which we will assume is $\mathcal{Y} = \mathbb{R}$. Labels are drawn from the conditional probability distribution $\mathbb{P}_{Y|X}$, which has probability density function $p_{Y|X}(y \mid x)$, and which together with $p_X(x)$ forms the joint probability density function $p_{X,Y}(x,y) = p_X(x)p_{Y|X}(y \mid x)$. We draw our training set $T = \{(x_i, y_i)\}_{i=1}^N$ from the joint probability density $p_{X,Y}(x,y)$.

Now let X be a random variable (more precisely, random vector) that takes values in $\mathcal{X}$ according to $p_X(x)$, and Y a random variable dependent upon X that takes values in $\mathbb{R}$ according to $p_{Y|X}(y \mid x)$, so that in particular the pair (X,Y) take values according to the joint probability distribution $p_{X,Y}(x,y)$. The variables X and Y, in other words, are just like the samples x_i and y_i except that we view them as random variables instead of as fixed samples.

If we had perfect knowledge of $\mathcal{X}$, $\mathcal{Y}$ and $p_{X,Y}(x,y)$; and we are using the squared loss as our measure of loss, i.e. $\ell(y, f(x)) = |y - f(x)|^2$; and we could pick any model f that we want, then we would minimize the following:

$$\begin{aligned} R_{\text{true}}(f) = \mathbb{E}_{X,Y}[(Y - f(X))^2] &= \int_{\mathcal{X}\times\mathcal{Y}} (y - f(x))^2 p_{X,Y}(x,y)\, dy\, dx \\ &= \int_{\mathcal{X}} \int_{\mathcal{Y}} (y - f(x))^2 p_{Y|X}(y \mid x)\, dy\, p_X(x)\, dx \\ &= \mathbb{E}_X \mathbb{E}_{Y|X}[(Y - f(X))^2 \mid X] \end{aligned} \tag{1.9}$$

The functional $R_{\text{true}}(f)$ is called the *true risk* of the model f. It measures the quality of the model f if we had an oracle that told us everything about the data generation process. In practice, of course, all we are given is the finite number of training points $T = \{(x_i, y_i)\}_{i=1}^N$. So instead of minimizing the true risk, which is almost always impossible because we simply do not have complete knowledge (in fact if we did, we would not be doing machine learning!), we instead minimize the *empirical risk*, which is the loss function we saw earlier:

$$R_{\text{emp}}(f) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 \,.$$

Note that the empirical risk is the finite sample average of $(Y - f(X))^2$. By the Law of Large Numbers, $R_{\text{emp}}(f) \to R_{\text{true}}(f)$ as $N \to \infty$ for a *fixed* f. This does not imply, though, that the minimizer of the empirical risk converges to the minimizer of the true risk as $N \to \infty$. Rather this is a topic in statistical learning theory and must be proved for each new model class.

We will study the ramifications of minimizing the empirical risk in Section 1.4.2. For now, let us return to the theoretical scenario now and minimize the true risk, $R_{\text{true}}(f)$. From (1.9) we see that we can solve for the optimal f pointwise, i.e., by finding $\widehat{f}(x)$ one x at a time. In particular, we obtain:

$$\widehat{f}(x) = \arg\min_{c} \mathbb{E}_{Y|X}[(Y - c)^2 \mid X = x] \,.$$

Taking the derivative with respect to c and setting it equal to zero, we obtain:

$$\widehat{f}(x) = \widehat{c} = \mathbb{E}_{Y|X}[Y \mid X = x] \,,$$

that is, the conditional expectation of the label Y given that $X = x$. Notice how the optimal model $\widehat{f}(x)$ depends only on the conditional expectation, and thus the conditional probability distribution $p_{Y|X}(y \mid x)$, giving more concrete justification to our earlier statement in Section 1.1.4 that in supervised learning we often ignore $p_X(x)$.

Note, in particular, if the label is a deterministic function of x, then seeing x once (i.e., drawing x as a training point x_i) is enough to determine $f(x)$. If there is noise in the labeling process, though, then one sample is not enough. However, in the training set there is typically only one observation $x_i = x$, and furthermore, in the test set we do not know the label and must estimate it. Therefore in k-nearest neighbors we replace $f(x) = \mathbb{E}[Y \mid X = x]$ with

$$f(x) = \text{Avg}[y_i \mid x_i \in N_k(x)] = \frac{1}{k}\sum_{x_i \in N_k(x)} y_i \,,$$

where we recall $N_k(x)$ consists of the k points from the training set T closest to x. The k-nearest neighbors algorithm is incorporating two approximations:

1. The expectation $\mathbb{E}_{Y|X}[Y \mid X = x]$ is approximated by a k-sample average over the training data.

2. The conditioning at the point $X = x$ is relaxed to conditioning on some region, the k-nearest neighbors in the training set, closest to the point x.

For large sample size N, the points in the k-nearest neighborhood are likely to be close to x, and as k gets larger the average will get more stable. In fact, under certain conditions on $p_{X,Y}(x,y)$, one can show that

$$\text{Avg}[y_i \mid x \in N_k(x_i)] \to \mathbb{E}_{Y|X}[Y \mid X = x] \text{ as } k, N \to \infty \text{ with } k/N \to 0\,.$$

However, we do not always have enough data points N to well approximate the above limits. In particular, as the dimension d increases, the number of sample points N needed to have k points close to each possible x increases rapidly, thus potentially leading to very bad approximations of $\mathbb{E}_{Y|X}[Y \mid X = x]$ (see Section 1.4.3 for more details).

The nearest neighbor model is good because it places very few assumptions on the underlying data generation process, encoded here by $p_{X,Y}(x,y)$. On the other hand, if we have prior knowledge about the data generation process, leveraging that knowledge and using a more structured class of models is preferred. The linear model described earlier is such a model. Recall the linear model is:

$$f(X;\theta) = \langle X, \theta \rangle = X^T\theta\,, \quad X, \theta : d \times 1\,,$$

where we have omitted the bias term but which can easily be incorporated or assumed to already be contained in X (as a constant dimension). Plugging this model into the true risk $R_{\text{true}}(f(\cdot;\theta))$ and minimizing the true risk with respect to θ (solved by differentiating with respect to θ and setting equal to zero) one obtains:

$$\widehat{\theta} = (\mathbb{E}_X[XX^T])^{-1}\mathbb{E}_{X,Y}[XY]\,.$$

Note the solution we obtained earlier for the empirical risk, given in (1.5), is the empirical version of the $\widehat{\theta}$ obtained here. Note, we do not condition on $X = x$. Rather we use the fact that we solving for an optimal linear model to average over the values of X and Y.

But what about classification? In the previous discussions, we assumed $\mathcal{Y} = \mathbb{R}$, but often we want to do classification and $\#(\mathcal{Y}) = M$, where M is the number of distinct classes. In this case the squared loss does not make sense. A standard choice is the 0-1 loss, which means that

$$\ell(y, f(x)) = \begin{cases} 0 & y = f(x) \\ 1 & y \neq f(x) \end{cases} \tag{1.10}$$

For now let us consider a general loss function, but we will come back to the 0-1 loss function shortly. Regardless of the choice of loss function, the true risk is:

$$R_{\text{true}}(f) = \mathbb{E}_{X,Y}[\ell(Y, f(X))]\,,$$

where we remind the reader that the expectation $\mathbb{E}_{X,Y}$ is taken over the joint probability density function of the data points X and the classes Y, $p_{X,Y}(x,y)$.

Similar to the calculation concluded in (1.9), we can condition on the data points X:

$$\begin{aligned}
R_{\text{true}}(f) = \mathbb{E}_{X,Y}[\ell(Y, f(X))] &= \int_{\mathcal{X}} \sum_{y \in \mathcal{Y}} \ell(y, f(x)) p_{X,Y}(x, y)\, dx \\
&= \int_{\mathcal{X}} \left(\sum_{y \in \mathcal{Y}} \ell(y, f(x)) p_{Y|X}(y \mid x) \right) p_X(x)\, dx \\
&= \mathbb{E}_X \left[\sum_{y \in \mathcal{Y}} \ell(y, f(X)) p_{Y|X}(y \mid X) \right].
\end{aligned}$$

Like before, from this calculation we again see it is sufficient to compute the optimal model, $\widehat{f}(x)$, point by point, meaning that:

$$\widehat{f}(x) = \arg\min_{y' \in \mathcal{Y}} \sum_{y \in \mathcal{Y}} \ell(y, y') p_{Y|X}(y \mid x)\,. \tag{1.11}$$

Now, if $\ell(y, f(x))$ is the 0-1 loss function (1.10), we can simplify (1.11) to

$$\widehat{f}(x) = \arg\max_{y' \in \mathcal{Y}} p_{Y|X}(y' \mid x)\,. \tag{1.12}$$

Indeed, suppose $\widehat{y} = \widehat{f}(x)$ in (1.11); we then have, since $\ell(y, y')$ is the 0-1 loss,

$$\min_{y' \in \mathcal{Y}} \sum_{y \in \mathcal{Y}} \ell(y, y') p_{Y|X}(y \mid x) = \sum_{y \neq \widehat{y}} p_{Y|X}(y \mid x)\,.$$

But then it must be that we removed the largest value from among $\{p_{Y|X}(y \mid x) : y \in \mathcal{Y}\}$ in the summation on the right hand side since the 0-1 loss weights all errors equally, i.e., it must be that $\widehat{y}$ is the most probable class; but that is (1.12)[5]. The model (1.12) is called the *naive Bayes classifier*. It says, given a data point x, assign the most probable class using the conditional probability $p_{Y|X}(y \mid x)$; in retrospect, this is sort of obvious. The catch is that we don't know $p_{Y|X}(y \mid x)$, and so we must estimate it. The k-nearest neighbors algorithm is one way of doing so, and it can be pretty good; see Figure 1.7.

Remark 1.5. The Bayes classifier is not perfect because of uncertainty in the data generation process. The source of that uncertainty is similar to the noisy label model (1.4), but it is generated differently since here the labels are binary. Rather, the uncertainty comes from the way in which the two classes are distributed over $\mathbb{R}^2$. Here is a simpler example to give you an idea of what is going on. Consider data in $\mathbb{R}^2$ generated from a Gaussian mixture model consisting of two Gaussians, i.e.,

$$p_X(x) = \frac{1}{2}\left[\frac{1}{2\pi\sigma_1^2} e^{-\|x-\mu_1\|_2^2/2\sigma_1^2} + \frac{1}{2\pi\sigma_2^2} e^{-\|x-\mu_2\|_2^2/2\sigma_2^2} \right].$$

[5] Thanks to Ali Zare for discussions related to this derivation and for helping to make the presentation clearer

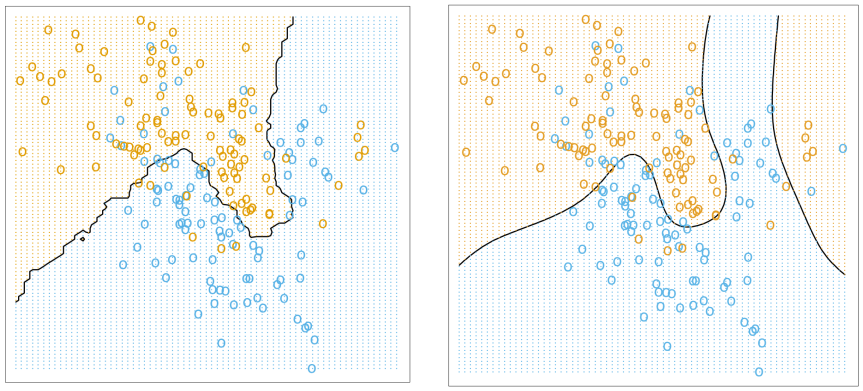

(a) *The k-nearest neighbors classifier for $k = 15$.* **(b)** *The naive Bayes classifier.*

Figure 1.7: *Comparison of the k-nearest neighbor classifier and the theoretical naive Bayes classifier. The k-nearest neighbor classifier does a good job of approximating the optimal Bayes decision boundary. Figures taken from [6].*

The way that data is sampled from $p_{X,Y}(x) = p_X(x)p_{Y|X}(y \mid x)$ is the following. First we flip a coin. If it lands heads we sample $x \sim \mathcal{N}(\mu_1, \sigma_1)$ and we assign x the label $y =$ `blue` (-1). If it lands tails, we sample $x \sim \mathcal{N}(\mu_2, \sigma_2)$ and we assign x the label $y =$ `orange` (+1). If the means μ_1, μ_2 are close enough and the standard deviations σ_1, σ_2 large enough, there will be significant overlap between the two distributions which creates uncertainty in the label of the point; see Figure 1.8. In our probabilistic framework, this means $p_{Y|X}(y \mid x)$ varies depending on the location of x, and in particular, $p_{Y|X}(\texttt{blue} \mid x) = p_{Y|X}(\texttt{orange} \mid x) = 1/2$ for x that are equidistant from μ_1 and μ_2 if $\sigma_1 = \sigma_2$.

1.4.2 Bias variance tradeoff

In the previous section we considered theoretical scenarios in which we had perfect knowledge of the data generating distribution $p_{X,Y}(x, y)$ and we were able to select any model $y = f(x)$ to fit the data. In practice, we only have a finite amount of data given by our training set $T = \{(x_i, y_i)\}_{i=1}^N$, and we can only select models from the model class $\mathcal{F}$ that we specify. This leads to two sources of additional error, beyond errors that may be impossible to avoid even with perfect knowledge (e.g., if the labels are noisy, as in the model given by (1.4)). These two additional sources of errors are:

1. *Bias:* Since we cannot pick any model that we wish, only models from the model class $\mathcal{F}$, there is the possibility that $\mathcal{F}$ does not contain a model that fully captures the relationship between the labels $y \in \mathcal{Y}$ and the data points $x \in \mathcal{X}$. In this case, even the best model from $\mathcal{F}$ will not optimally model this relationship, and the resulting error is called a bias error. For example, if we use linear regression to fit data in which there

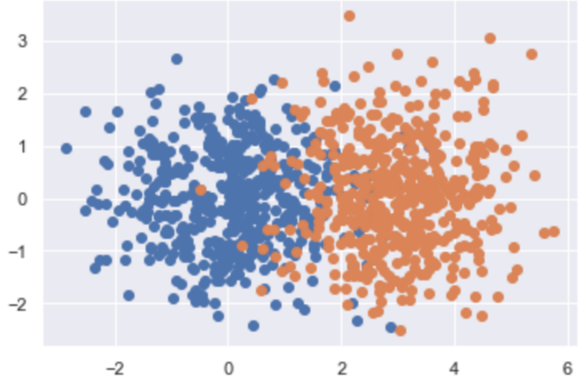


Figure 1.8: *Two Gaussian mixture model in which there is significant overlap between the two Gaussians. Determining the label of new points sampled in the overlap region is difficult and creates uncertainty, even for the naive Bayes classifier with complete knowledge of the data generation process.*

is a nonlinear relationship between y and x, then our machine learned model will be biased since it will not be able capture the nonlinearity in the data. Furthermore, it may be that $\mathcal{F}$ contains a good model, but we are incapable of selecting it with the training set T that we are given; this will also lead to bias error.

2. *Variance:* Since we do not have complete knowledge of $p_{X,Y}(x,y)$ and in fact only have a finite training set T sampled according to $p_{X,Y}(x,y)$, we must estimate the generalization error and select the best model using only T. However, different (theoretical) draws of the training set T will lead to different models being selected, some of which may generalize to new points better than others. This randomness imparted into the model selection process can either be relatively small or drastic. The variance error encodes this source of error.

Figure 1.9 illustrates the difference between model classes $\mathcal{F}$ with low bias and high variance, versus those with high bias and low variance. Indeed, these two sources of error are often in tension, i.e., reducing one increases the other. This phenomenon is referred to as the *bias-variance trade-off*. One way to think about this tradeoff is as follows. Simple models, such as linear regression, may not fit the training data perfectly (or even well), and hence have a high bias, but their predictions are robust to small perturbations in the test points, and thus have a low variance. On the other hand, complex models such as the 1-nearest neighbor classifier may be able to fit the training data extremely well (low bias), but their high complexity means they may fit spurious patterns in the data (such as noise) and their output may change drastically with small changes on the evaluated data point, so much so that their predictive power is

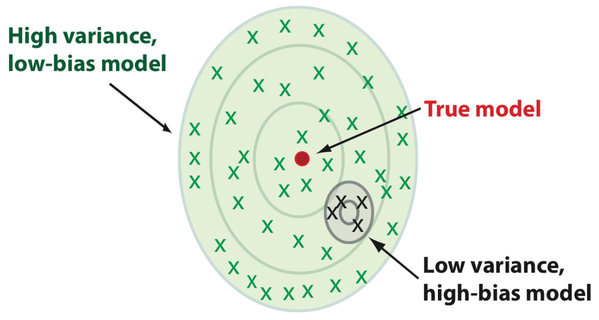


Figure 1.9: *Illustration of high variance, low bias model classes and low variance, high bias model classes. Each "X" denotes a model obtained through a particular draw of the training set T. In the high variance, low bias regime, indicated by the green X's, the average of all the models is close to the true model, marked with the red dot. However, there is a large variance between models as one varies the training set, meaning that any one model may be very far from the true model. In the low variance, high bias regime, indicated with the black X's, the models are not centered around the true model and thus the average model obtained from them will not be a good approximation of the true model. On the other hand, different draws of the training set lead to nearly the same model, as indicated by their tight arrangement. Figure taken from [5].*

limited (high variance). Models, such as the $k = 15$ nearest neighbor classifier (see Figure 1.6), that balance reducing bias with increasing variance are the goal in predictive machine learning.

Let us now derive the bias variance tradeoff in a general statistical setting using the squared loss. For this we will assume there exists a deterministic function $F : \mathbb{R}^d \to \mathbb{R}$ that determines the uncorrupted label of $x \in \mathbb{R}^d$, and that the labels we observe are corrupted by white noise:

$$y_i = F(x_i) + \varepsilon_i \, .$$

The variables ε_i independently and identically distributed normal random variables with mean zero and variance σ^2, i.e.,

$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2) \, ,$$

which implies, in particular, that $\mathbb{E}[\varepsilon_i] = 0$ and $\mathbb{E}[\varepsilon_i^2] = \sigma^2$.

Given a training set T drawn from $p_{X,Y}(x,y)$ and a parameterized model class $\mathcal{F} = \{f(\cdot;\theta) : \theta \in \mathbb{R}^n\}$, we obtain a model by minimizing the squared loss (empirical risk) over the training set:

$$\widehat{\theta}_T = \underset{\theta \in \mathbb{R}^n}{\arg\min}\, R_{\text{emp}}(f(\cdot;\theta)) = \underset{\theta \in \mathbb{R}^n}{\arg\min}\, \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i;\theta))^2 \, ,$$

where the subscript T emphasizes that $\widehat{\theta}_T$ depends upon the particular training set used to solve for the model. Given this model, we define the *conditional test* *error*, which is expected test error given the draw of the training set T:

$$\mathrm{err}(\mathcal{F}, T) = \mathbb{E}_{X,Y}[(Y - f(X; \widehat{\theta}_T))^2] .$$

Note that $\mathrm{err}(\mathcal{F}, T)$ is often a quantity we are very interested in, as it encodes the ability of our selected model $f(\cdot; \widehat{\theta}_T)$, obtained through our specific training set T, to generalize to new points. A related quantity is the *expected test error*, which is defined as:

$$\mathrm{Err}(\mathcal{F}) = \mathbb{E}_T[\mathrm{err}(\mathcal{F}, T)] .$$

The expectation $\mathbb{E}_T$ is an expectation over all possible training sets T with N elements, drawn according to $p_{X,Y}(x, y)$. Note that the expected test error, $\mathrm{Err}(\mathcal{F})$, measures the quality of the model class $\mathcal{F}$ and not any particular model $f(x; \widehat{\theta}_T)$ selected from it using a specific training set T.

Figure 1.10 shows that minimizing the empirical risk by increasing model complexity is not a good way to minimize the conditional test error or the expected test error. Indeed, the empirical risk decreases with model complexity assuming that increasing complexity allows us to obtain increasingly better approximations of F, or to even include F at some point. However, such complex models, particularly when the model is more complex than F, generalize poorly as they fit spurious patterns generated by the additive noise ε_i, which is reflected in the increase of the conditional and expected test errors at a certain point.

Let us now derive the bias variance tradeoff, which will give a quantitative interpretation for the empirical results we have observed. The result will decompose the expected test error at an arbitrary, but fixed point $x \in \mathcal{X}$. We thus define the *expected test error at x*[6] as:

$$\mathrm{Err}(\mathcal{F}, x) = \mathbb{E}_T \mathbb{E}_{Y|X}[(Y - f(X; \widehat{\theta}_T))^2 \mid X = x] .$$

Theorem 1.6 (Bias variance tradeoff)**.** *Let $T = \{(x_i, y_i)\} \subset \mathbb{R}^d \times \mathbb{R}$ be an arbitrary training set drawn from a joint distribution $p_{X,Y}(x, y)$ with $y_i = F(x_i) + \varepsilon_i$ for some deterministic function $F : \mathbb{R}^d \to \mathbb{R}$ and iid random variables $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$. Then the expected test error at $x \in \mathcal{X}$ can be decomposed as:*

$$\mathrm{Err}(\mathcal{F}, x) = \sigma^2 + (F(x) - \mathbb{E}_T[f(x; \widehat{\theta}_T)])^2 + \mathbb{E}_T[(f(x; \widehat{\theta}_T) - \mathbb{E}_T[f(x; \widehat{\theta}_T)])^2] .$$

Theorem 1.6 decomposes the test error at an arbitrary point $x \in \mathcal{X}$ into three components:

- The irreducible noise error: σ^2
- The bias induced by model class: $F(x) - \mathbb{E}_T[f(x; \widehat{\theta}_T)]$
- The variance of the selected model from the model class: $\mathbb{E}_T[(f(x; \widehat{\theta}_T) - \mathbb{E}_T[f(x; \widehat{\theta}_T)])^2]$

[6] Thanks to Ali Zare for pointing out some ambiguity in the original definition.

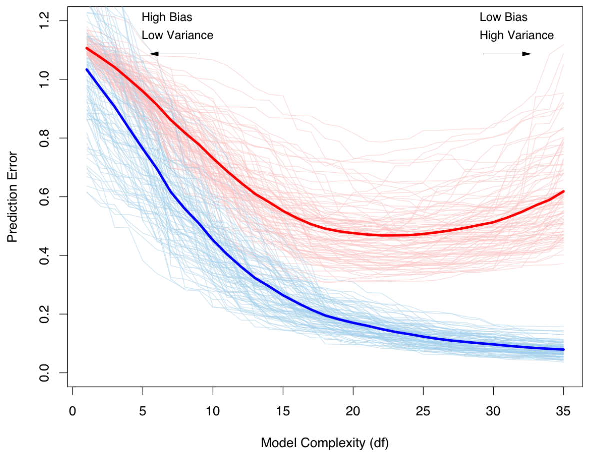


Figure 1.10: *Behavior of test error and training error as the model class complexity is increased. The plot depicts* 100 *different training sets drawn from the same distribution. The light blue curves show the empirical risk* $R_{emp}(f(\cdot;\widehat{\theta}_T))$ *for each of the* 100 *training sets,* T. *The light red curves show the corresponding conditional test error,* $\mathrm{err}_T(\mathcal{F}, f(\cdot;\widehat{\theta}_T))$ *for each of the training set. The dark blue curve is the expected empirical risk over all draws of the training set,* $\mathbb{E}_T[R_{emp}(f(\cdot;\widehat{\theta}_T))]$, *estimated by averaging the light blue curves. The dark red curve is the expected test error* $\mathrm{Err}(\mathcal{F})$, *estimated by averaging the light red curves. Figure taken from* [6].

The irreducible noise error is an error that is incurred regardless of the model class used to fit the training data; it is a fundamental limitation of any model learned from training data generated by $p_{X,Y}(x, y)$. The bias measures the capacity of the model class to contain a model that fits the underlying function $F(x)$ that generates the labels. The bias error will decrease (or at least stay flat) as the complexity of the model class increases. The variance measures the stability of the selected model, $f(x;\widehat{\theta}_T)$, over different draws of the training set T. If the variance is low, then the model class is stable and different draws of the training set (theoretical or not) will not influence the learned model too much. On the other hand, if the variance is high, then the model is unstable and a new training set may result in a very different model. Since the underlying, noiseless label $F(x)$ is deterministic, this may result in poor predictions because it is likely to get "unlucky" with a high variance model class. Figure 1.9 illustrates these ideas. Let us now prove the theorem.

Proof. We write $Y(x) = F(x) + \varepsilon$ to denote the random variable Y conditioned on $X = x$ for some $x \in \mathcal{X}$. Our first goal is to extract the irreducible noise error through the following calculation:

$$\begin{aligned}
&\mathbb{E}_{Y|X}\mathbb{E}_T[(Y - f(X;\widehat{\theta}_T))^2 \mid X = x] \\
&\qquad = \mathbb{E}_{Y|X}\mathbb{E}_T[(Y(x) - f(x;\widehat{\theta}_T))^2] \\
&\qquad = \mathbb{E}_{Y|X}\mathbb{E}_T[(Y(x) - F(x) + F(x) - f(x;\widehat{\theta}_T))^2] \\
&\qquad = \mathbb{E}_{Y|X}\mathbb{E}_T[(Y(x) - F(x))^2] \\
&\qquad\qquad + \mathbb{E}_{Y|X}\mathbb{E}_T[(F(x) - f(x;\widehat{\theta}_T))^2] \\
&\qquad\qquad + 2\mathbb{E}_{Y|X}\mathbb{E}_T[(Y(x) - F(x))(F(x) - f(x;\widehat{\theta}_T))] \\
&\qquad = \mathbb{E}_\varepsilon[\varepsilon^2] + \mathbb{E}_T[(F(x) - f(x;\widehat{\theta}_T))^2] + 2\mathbb{E}_\varepsilon\mathbb{E}_T[\varepsilon(F(x) - f(x;\widehat{\theta}_T))] \\
&\qquad = \sigma^2 + \mathbb{E}_T[(F(x) - f(x;\widehat{\theta}_T))^2] + 2\mathbb{E}_\varepsilon[\varepsilon]\mathbb{E}_T[F(x) - f(x;\widehat{\theta}_T)] \\
&\qquad = \sigma^2 + \mathbb{E}_T[(F(x) - f(x;\widehat{\theta}_T))^2]
\end{aligned}$$

where we used $\mathbb{E}[\varepsilon] = 0$, $\mathbb{E}[\varepsilon^2] = \sigma^2$, and the fact that the noise ε on the label of a test point x is independent of the training set T.

Now we further decompose the second term into the bias term and the variance term:

$$\begin{aligned}
&\mathbb{E}_T[(F(x) - f(x;\widehat{\theta}_T))^2] \\
&\qquad = \mathbb{E}_T[(F(x) - \mathbb{E}_T[f(x;\widehat{\theta}_T)] + \mathbb{E}_T[f(x;\widehat{\theta}_T)] - f(x;\widehat{\theta}_T))^2] \\
&\qquad = \mathbb{E}_T[(F(x) - \mathbb{E}_T[f(x;\widehat{\theta}_T))^2] + \mathbb{E}_T[(f(x;\widehat{\theta}_T) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])^2] \\
&\qquad\qquad + \mathbb{E}_T[(F(x) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])(f(x;\widehat{\theta}_T) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])] \\
&\qquad = (F(x) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])^2 + \mathbb{E}_T[(f(x;\widehat{\theta}_T) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])^2] \\
&\qquad\qquad + (F(x) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])\mathbb{E}_T[f(x;\widehat{\theta}_T) - \mathbb{E}_T[f(x;\widehat{\theta}_T)]] \\
&\qquad = (F(x) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])^2 + \mathbb{E}_T[(f(x;\widehat{\theta}_T) - \mathbb{E}_T[f(x;\widehat{\theta}_T)])^2]
\end{aligned}$$

The proof is thus completed. □

Example 1.7. [7] Let us now apply Theorem 1.6 to the k-nearest neighbors algorithm and the linear model. Starting with k-nearest neighbors, let us assume for simplicity that the training data points $\{x_i\}_{i=1}^N$ are fixed, but the labels $y_i = F(x_i) + \varepsilon_i$ are still random due to the random noise ε_i. In order to emphasize this assumption, we will replace $\mathbb{E}_T$ with $\mathbb{E}_\varepsilon$, to emphasize the expectation is just with respect to the noise and not the draws of $\{x_i\}_{i=1}^N$. In this case Theorem 1.6 can be written as:

$$\mathrm{Err}(k, x) = \sigma^2 + (F(x) - \mathbb{E}_\varepsilon[f(x;k)])^2 + \mathbb{E}_\varepsilon\left[(f(x;k) - \mathbb{E}_\varepsilon[f(x;k)])^2\right] . \quad (1.13)$$

[7] Thanks to Ali Zare and Xitong Zhang for helpful discussions on this example, which clarified its presentation.

Let us first compute $\mathbb{E}_\varepsilon[f(k,x)]$. We obtain

$$\begin{aligned}\mathbb{E}_\varepsilon[f(x;k)] = \mathbb{E}_\varepsilon\left[\frac{1}{k}\sum_{x_i\in N_k(x)} y_i\right] &= \mathbb{E}_\varepsilon\left[\frac{1}{k}\sum_{x_i\in N_k(x)} (F(x_i)+\varepsilon_i)\right]\\ &= \frac{1}{k}\sum_{x_i\in N_k(x)} (F(x_i)+\mathbb{E}_\varepsilon[\varepsilon_i])\\ &= \frac{1}{k}\sum_{x_i\in N_k(x)} F(x_i)\,,\end{aligned}$$

since $\mathbb{E}_\varepsilon[\varepsilon_i]=0$. Now going back to (1.13) let us use this calculation to further simplify the variance term. We have:

$$\begin{aligned}\mathbb{E}_\varepsilon\left[(f(x;k)-\mathbb{E}_\varepsilon[f(x;k)])^2\right] &= \mathbb{E}_\varepsilon\left[\left(\frac{1}{k}\sum_{x_i\in N_k(x)}(F(x_i)+\varepsilon_i)-\frac{1}{k}\sum_{x_i\in N_k(x)}F(x_i)\right)^2\right]\\ &= \mathbb{E}_\varepsilon\left[\left(\frac{1}{k}\sum_{x_i\in N_k(x)}\varepsilon_i\right)^2\right]\\ &= \mathbb{E}_\varepsilon\left[\frac{1}{k^2}\sum_{i,j=1}^{k}\varepsilon_i\varepsilon_j\right]\\ &= \frac{1}{k^2}\sum_{i,j=1}^{k}\mathbb{E}_\varepsilon[\varepsilon_i\varepsilon_j] = \frac{\sigma^2}{k}\,,\end{aligned}$$

where we used $\mathbb{E}_\varepsilon[\varepsilon_i\varepsilon_j]=\sigma^2\delta(i-j)$. Putting everything together in (1.13) we have

$$\mathrm{Err}(k,x) = \sigma^2 + \left(F(x)-\frac{1}{k}\sum_{x_i\in N_k(x)}F(x_i)\right)^2 + \frac{\sigma^2}{k}\,.$$

We see that the variance, σ^2/k, decreases as the number of neighbors k increases (recall that larger k means a less complex model). On the other hand, larger k means that $F(x)$ is estimated from a larger neighborhood around x, potentially increasing the bias, especially for irregular functions F.

Example 1.8. For linear models $f(x;\theta)=\langle x,\theta\rangle$, we obtain the following for the expected training error, again assuming the $\{x_i\}_{i=1}^N$ are fixed and it is only the labels $y_i=F(x_i)+\varepsilon_i$ that are random:

$$\frac{1}{N}\sum_{i=1}^{N}\mathrm{Err}(\mathcal{F}_{\text{linear}},x_i) = \sigma^2 + \frac{1}{N}\sum_{i=1}^{N}(F(x_i)-\mathbb{E}_\varepsilon[f(x_i;\widehat{\theta}_T)])^2 + \frac{d}{N}\sigma^2\,.$$

Here the model complexity increases with the dimension d, but the variance term only increases linearly in d. The bias term will depend on whether $F(x)$ is well approximated by a linear model. We will prove a similar result to this in the next section when we discuss the curse of dimensionality.

1.4.3 Curse of dimensionality

We have now examined two models closely: linear models and k-nearest neighbors. In the previous section, Section 1.4.2, we observed that linear models complexity scales with the dimension d of our data, but the variance only scales linearly in d. Thus linear models are stable. On the other hand, if the mapping $x \mapsto y$ is not linear, a machine learned linear model will not be able to capture the relationship between data point x and label y, and there will be a potentially large bias in the model.

For k-nearest neighbors the situation is a bit more subtle. Figures 1.6 and 1.7 would seem to indicate that it is superior to the linear model and in general a good choice. Indeed, without knowing the underlying data generation process, which was highly nonlinear, k-nearest neighbors resulted in a model that was nearly as good as the naive Bayes model, which is the best one can do under the circumstances of our framework. The main issue from the analysis of Section 1.4.2 is that in order to reduce the variance of the k-nearest neighbor model we must select a reasonably large k (certainly not $k = 1$, see Figure 1.5). However, increasing k may increase the bias, as we select more and more points that are further away from the point who's label we are trying to predict. It would seem, though, that if we have a large amount of data, this issue is removed as we can select a large k without having to incorporate points that are far away from the central point x into its neighborhood $N_k(x)$. This intuition works in low dimensions but breaks down in high dimensions. While there are many manifestations of this problem, they are all generally referred to as the *curse of dimensionality*. In what follows we give a few examples.

Example 1.9. Here is one manifestation. Consider a d-dimensional unit cube, $\mathcal{X} = Q \subset \mathbb{R}^d$,

$$Q = [0,1]^d = \underbrace{[0,1] \times \cdots \times [0,1]}_{d \text{ times}}$$

Suppose our test point x is the corner, $x = (0,\ldots,0)$, and that our training set is distributed uniformly in Q, meaning that $p_X(x) = 1$ for all $x \in Q$. A related method to k-nearest neighbors is to simply take all points within a geometric neighborhood of x and average their labels to obtain an estimate for the label of x. For example, we could take a sub-cube $Q_x \subseteq Q$, and average all the labels y_i of training points $x_i \in Q_x$ to obtain an estimate for the label of x:

$$f(x;\theta) = \operatorname{Avg}\{y_i : x_i \in Q_x\}\,. \tag{1.14}$$

The parameters θ of this model have to do with how we construct the cube Q_x. Let us suppose we want to capture a fraction r of the volume of Q, which is one (i.e., $\operatorname{vol}(Q) = 1$). How long must the edge of Q_x be? Let's call this edge length $e_d(r)$, since it depends on the fraction of the volume r we want to capture and

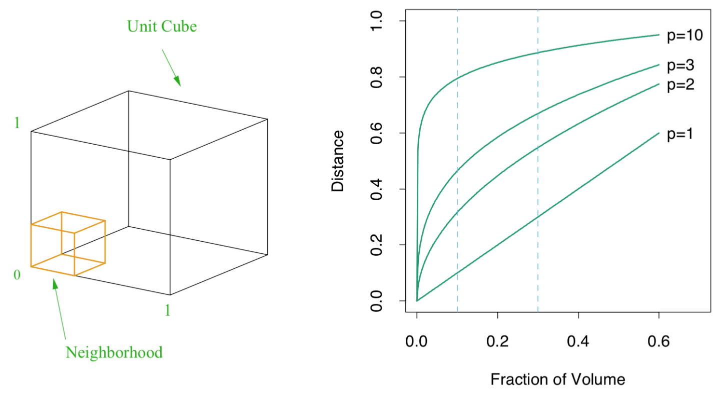


Figure 1.11: *One illustration of the curse of dimensionality. Left: A neighborhood cube Q_x (orange) as a subset of the unit cube Q. Right: The horizontal axis is the fraction of volume r that the neighborhood cube contains. The vertical axis is the required side length $e_d(r)$ of the neighborhood cube Q_x. Curves plotting $e_d(r)$ for $d = 1, 2, 3, 10$ are shown (note: in the figure, $p = d$). Figure taken from [6].*

the dimension d of the data space. In one dimension, it is clear that $e_1(r) = r$. However, in d-dimensions, we have

$$e_d(r) = r^{1/d}.$$

This is decidedly less favorable. Indeed, as an example, in 10 dimensions we have:

$$e_{10}(0.01) = 0.63 \quad \text{and} \quad e_{10}(0.1) = 0.80,$$

meaning that to capture just 1% of the volume in 10 dimensions we need an edge length of 0.63, and to capture 10% of the volume in 10 dimensions, we need an edge length of 0.80. Remember the side length of the cube Q is just 1.0! Therefore, what we thought was a local neighborhood due to our intuition about how things work in low dimensions, turns out, in fact, to be a very large neighborhood in high dimensions. Figure 1.11 illustrates this principle.

Example 1.10. Suppose instead we consider spherical neighborhoods, as is often the case. We consider a model similar to (1.14) but instead average the labels within a sphere of radius r centered at x:

$$f(x;r) = \mathrm{Avg}\{y_i : \|x - x_i\|_2 \leq r\}. \tag{1.15}$$

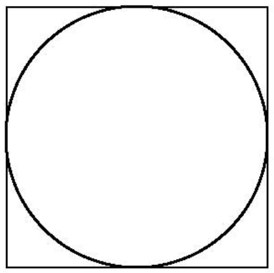
Let us again suppose that our data space is $\mathcal{X} = Q = [0,1]^d$, the unit cube. Suppose further that $x = (1/2, \ldots, 1/2)$, the center point of the cube, and we take the largest radius r possible, which is $r = 1/2$. If the data points are sampled uniformly from Q, then the odds of no training points being within $1/2$ of x are

$$\mathbb{P}(x_i \notin B, \forall 1 \leq i \leq N) = \left(1 - \frac{\mathrm{vol}(B)}{\mathrm{vol}(Q)}\right)^N = (1 - \mathrm{vol}(B))^N,$$

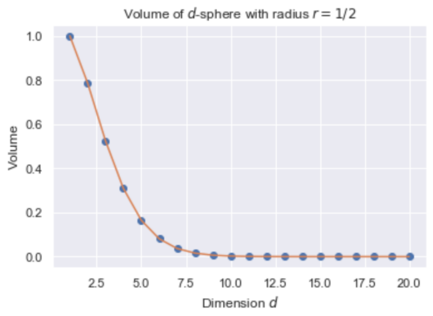
where B is the ball of radius $1/2$ centered at $x = (1/2, \ldots, 1/2)$ and we used the fact that $\mathrm{vol}(Q) = 1$. The volume of a ball of radius r, B_r^d, in d-dimensions, is

$$\mathrm{vol}(B_r^d) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d \, .$$

In two dimensions, $\mathrm{vol}(B) \approx 0.79$, and so with $N = 100$ training points sampled from Q, the odds that no training points are in B is approximately $(0.21)^{100} \approx 0$; in other words, we are nearly guaranteed to have some training points close to x. Indeed, Figure 1.12a plots a circle inscribed in a square in two dimensions, and we see that the area of the circle dwarfs the area of the corners inside the square, but outside the circle.



(a) *A circle inscribed in a square.*



(b) *Volume of the d-dimensional ball of radius* $1/2$ *as a function of the dimension d.*

Figure 1.12: *Another illustration of the curse of dimensionality. Left: In low dimensions, it is likely that a test point x centered in a box will have training points sampled within a small radius of the point x. Right: In high dimensions, however, the volume of the d-sphere of radius* $r = 1/2$ *inscribed in the unit cube is dwarfed by the cube's unit volume, indicating that almost certainly the training points will be be situated in the "corners" of the cube and thus far from the test point x centered in the middle of the cube.*

On the other hand, in high dimensions the volume of the ball B rapidly decreases while the volume of the cube Q remains fixed at one; see Figure 1.12b. It thus follows that it is very unlikely for a training point to lie within the ball of radius $1/2$ centered at the test point x, making the model (1.15) useless. Indeed, if $d = 10$ and $N = 100$, then the odds of no training points lying within B are over 90%. When $d = 20$, it is essentially guaranteed!

Example 1.11. Instead of sampling points from the cube Q, let us restrict to the ball B. Surely in this case the situation must be more favorable? In fact the answer is still no. Let us assume now that B has a radius of $r = 1$ and is centered at the origin. We consider the test point $x = (0, \ldots, 0)$ at the origin, and sample training points $\{x_1, \ldots, x_N\}$ from $\mathcal{X} = B$ according to the uniform distribution over B (so again p_X is constant). We are interested in the 1-nearest neighbor distance from x. One can show that over all possible draws of the

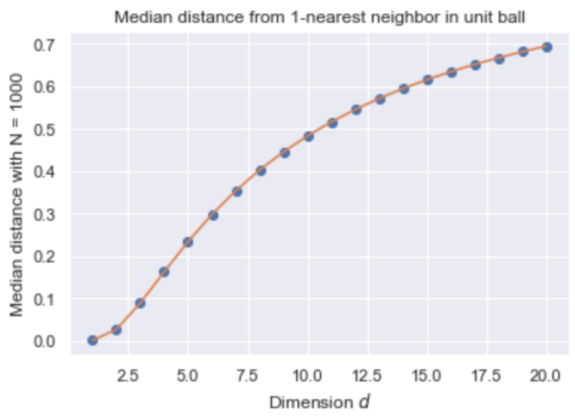


Figure 1.13: *The median distance of the 1-nearest neighbor to the origin in the unit ball as a function of dimension d, assuming the training points are sampled from the uniform distribution. As the dimension d increases, even the 1-nearest neighbor becomes very far from the test point at the origin.*

training set, the median distance from the origin x to closest training point is:

$$\text{dist}_{\text{1-NN}}(d, N) = \left(1 - (1/2)^{1/N}\right)^{1/d} .$$

As with the other examples, in low dimensions we are fine. Indeed, for $d = 2$ and $N = 1000$, we have $\text{dist}_{\text{1-NN}}(2, 1000) \approx 0.03$. On the other hand, if we go to $d = 20$ dimensions, the situation becomes far worse, as $\text{dist}_{\text{1-NN}}(20, 1000) \approx 0.70$! Keep in mind, the distance to the boundary is one. Figure 1.13 plots $\text{dist}_{\text{1-NN}}(d, 1000)$ for $1 \leq d \leq 20$. Since this is the 1-nearest neighbor distance, it means that all k-nearest neighbors will be far from x in high dimensions, thus leading to poor models.

Example 1.12. In this example we can see the ramifications of the analysis contained in the previous examples. Suppose that $\mathcal{X} = [-1, 1]^d$, the cube of side-length two centered at the origin. Suppose additionally that labels y are derived from x according to:

$$y = F(x) = e^{-8\|x\|_2^2} ,$$

with no measurement error. We are going to use a 1-nearest neighbor model to estimate the label y of new test points. Let us consider a test point $x = (0, \ldots, 0)$ at the origin, which has label $f((0, \ldots, 0)) = 1$. Draw a training set $T = \{(x_i, y_i)\}_{i=1}^N$ and let $x_0 \in T$ be the point closest to x, and let $y_0 = F(x_0) = e^{-8\|x_0\|_2^2}$, which is the estimate for the label of x. Then using the bias-variance decomposition (Theorem 1.6), the expected test error at x is:

$$\text{Err}(\text{1-NN}, x) = \underbrace{(1 - \mathbb{E}_T[y_0])^2}_{\text{bias}} + \underbrace{\mathbb{E}_T\left[(y_0 - \mathbb{E}_T[y_0])^2\right]}_{\text{variance}} .$$

Suppose that the number of training points is $N = 1000$. In this case, the model will be biased because we know that $y_0 \leq 1$ no matter what. And indeed, the bias error will dominate, and grow large as the dimension d increases, since like in Example 1.11 and Figure 1.13 the median (and mean) distance of the 1-nearest neighbor from the origin will grow with the dimension. By dimension $d = 10$, more than 99% of the training samples will be, on average, at a distance greater than 0.5 from the origin, leading to a severe underestimate of the label since the $F(x) = e^{-8\|x\|_2^2}$ decays very rapidly; see Figure 1.14. Note that the bias does not always dominate. For example, if $F(x)$ only depends on a few dimensions of the data, e.g., $F(x) = (1/2) \cdot (x(1) + 1)^3$, then the variance error will dominate and will grow rapidly with the dimension; see Figure 1.15.

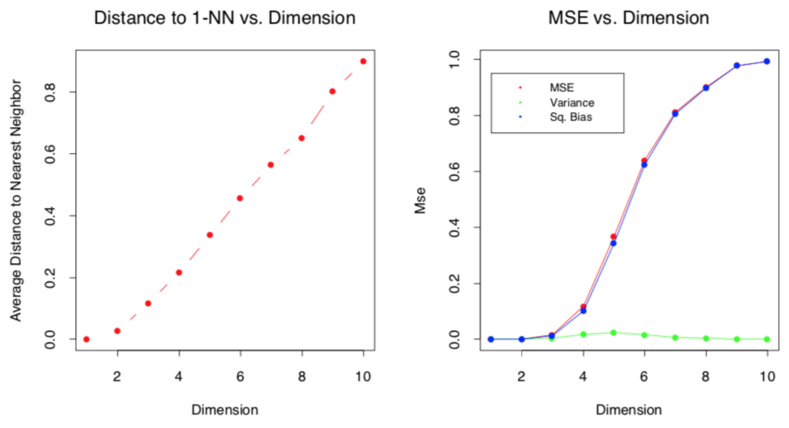


Figure 1.14: *Plots illustrating Example 1.12. Left: The average distance of the 1-nearest neighbor to the origin over many draws of the training set with $N = 1000$ training points, as a function of the dimension d. Right: The expected test error at the origin for the label function $y = e^{-8\|x\|_2^2}$ (MSE), as a function of the dimension, and broken down into its bias squared component and its variance component. Figure taken from [6].*

Another way of thinking about the curse of dimensionality is to consider how many training points N would be required to avoid it. Indeed, suppose in one dimension we require $N = 100$ training points to densely sample the space $\mathcal{X}$, e.g., $\mathcal{X} = [0, 1]$. Then to have the same sampling density for $\mathcal{X} = Q = [0, 1]^d$, we would require 100^d training points!

Now, in practice, we are very rarely confronted with a supervised learning problem in a high dimensional cube Q or ball B with a uniform sampling density. Indeed, consider the MNIST data base, consisting of 28×28 gray-scale images, where each pixel takes a value between 0 and 255. Suppose these gray-scale values are divided by 255 so they are normalized to lie in the interval $[0, 1]$. Then $\mathcal{X} = [0, 1]^{784}$, which is very high dimensional! But, the sampling density $p_X(x)$ is not uniform. On the contrary, looking back at Figure 1.1, one

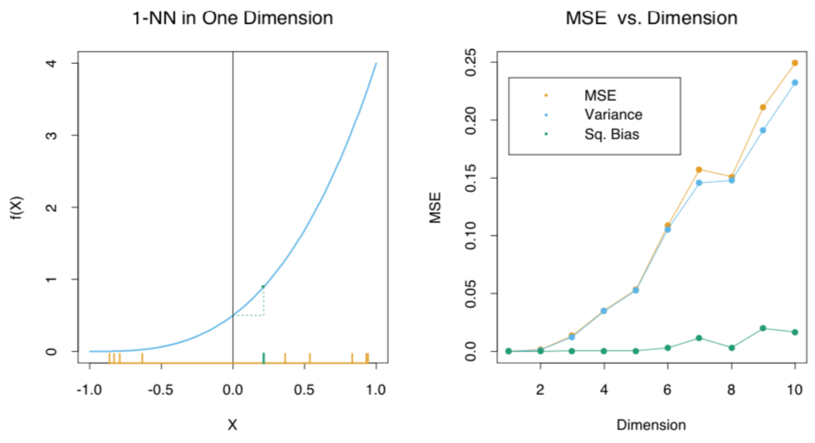


Figure 1.15: *Additional plots illustrating Example 1.12. In this figure the label function is* $y = (1/2) \cdot (x(1) + 1)^3$, *which only depends on the first dimension of the data point* x. *The variance rapidly increases with the dimension, though, leading to another manifistation of the curse of dimensionality. Figure taken from [6].*

sees that the MNIST digit images are highly structured. This implies that $p_X(x)$ is (essentially) supported on a much lower dimensional set contained within Q. This is what makes learning possible, but the challenge is that we must take advantage of this fact without being able to precisely know what the supporting set of $p_X(x)$ is.

Another advantage one may have is prior knowledge on the labeling function $y = F(x) + \varepsilon$. Indeed, if $F(x) = \langle x, \theta_0 \rangle$ is linear, and we know this fact but we do not know the parameters θ_0, we can still leverage this knowledge to restrict our model class to the class of linear models. Suppose this is the case, and that the labels are corrupted versions of $F(x)$, i.e.,

$$y = F(x) + \varepsilon = \langle x, \theta_0 \rangle + \varepsilon\,, \quad x \in \mathbb{R}^d\,, \; \varepsilon \sim \mathcal{N}(0, \sigma^2)\,.$$

Suppose we take our model class to be all possible linear models

$$\mathcal{F} = \{f(x;\theta) = \langle x, \theta \rangle : \theta \in \mathbb{R}^d\}\,.$$

Given a training set $T = \{(x_i, y_i)\}_{i=1}^N$ we obtain a model from $\mathcal{F}$ by minimizing the squared loss:

$$\widehat{\theta}_T = \operatorname*{arg\,min}_{\theta \in \mathbb{R}^d} \frac{1}{N} \sum_{i=1}^N (y_i - \langle x_i, \theta \rangle)^2\,.$$

From equation (1.5) we know that

$$\widehat{\theta}_T = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}\,,$$

where $\mathbf{X}$ is the $d \times N$ matrix containing the training point x_i on the i^{th} column, and $\mathbf{y}$ is the $N \times 1$ vector containing y_i in the i^{th} entry.

Also let $\boldsymbol{\varepsilon}$ be the $N \times 1$ vector with ε_i as its i^{th} entry. Now let $x \in \mathbb{R}^d$ be a test point, which we consider as a $d \times 1$ vector to be consistent with $\mathbf{X}$. Our prediction for its label is $f(x; \widehat{\theta}_T)$, which we can write as:

$$\begin{aligned}
f(x; \widehat{\theta}_T) &= \langle x, \widehat{\theta}_T \rangle \\
&= \langle x, (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y} \rangle \\
&= \langle x, (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}(\mathbf{X}^T\theta_0 + \boldsymbol{\varepsilon}) \rangle \\
&= \langle x, \theta_0 + (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\boldsymbol{\varepsilon} \rangle \\
&= \langle x, \theta_0 \rangle + \langle x, (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\boldsymbol{\varepsilon} \rangle \\
&= F(x) + \langle \mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x, \boldsymbol{\varepsilon} \rangle \\
&= F(x) + \sum_{i=1}^{N} (\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(i)\varepsilon_i \,.
\end{aligned}$$

From this calculation and the fact that $\mathbf{X}$ is independent of ε_i, we conclude that

$$\mathbb{E}_T[f(x; \widehat{\theta}_T)] = F(x) + \sum_{i=1}^{N} \mathbb{E}_T[(\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(i)]\mathbb{E}_T[\varepsilon_i] = F(x)\,,$$

showing that the least squares fit is unbiased estimator for linear models since from Theorem 1.6 we have:

$$\text{bias } = F(x) - \mathbb{E}_T[f(x; \widehat{\theta}_T)]\,.$$

Therefore, if we apply Theorem 1.6 (bias-variance trade-off), we will have only the irreducible error σ^2 and the variance error,

$$\begin{aligned}
\text{Err}(\mathcal{F}, x) &= \sigma^2 + \text{Var}_T\left[\sum_{i=1}^{N} (\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(i)\varepsilon_i\right] \\
&= \sigma^2 + \mathbb{E}_T\left[\left(\sum_{i=1}^{N} (\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(i)\varepsilon_i\right)^2\right] \\
&= \sigma^2 + \mathbb{E}_T\left[\sum_{i,j=1}^{N} (\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(i)\varepsilon_i(\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(j)\varepsilon_j\right] \\
&= \sigma^2 + \sum_{i,j=1}^{N} \mathbb{E}_T[(\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(i)(\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(j)]\,\underbrace{\mathbb{E}_T[\varepsilon_i\varepsilon_j]}_{\sigma^2\delta(i-j)} \\
&= \sigma^2 + \sigma^2\sum_{i=1}^{N} \mathbb{E}_T[(\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)(i)^2] \\
&= \sigma^2 + \sigma^2\mathbb{E}_T\left[\|\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x\|_2^2\right]\,.
\end{aligned}$$

Now, we also have:

$$\begin{aligned}\|\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x\|_2^2 &= (\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x)^T\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x\\ &= x^T(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x = x^T(\mathbf{X}\mathbf{X}^T)^{-1}x\end{aligned}$$

Therefore

$$\mathbb{E}_T\left[\|\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}x\|_2^2\right] = \mathbb{E}_T[x^T(\mathbf{X}\mathbf{X}^T)^{-1}x] = x^T\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]x\,.$$

Now, this quantity is a real number. We can write any real number $a \in \mathbb{R}$ as $\text{Trace}[a]$, where we view a as a 1×1 matrix. Recall that $\text{Trace}[A] = \sum_{k=1}^{m} A(k,k)$ for an $m \times m$ matrix A. We also note that for matrices A, B, C we have

$$\text{Trace}[ABC] = \text{Trace}[BCA]$$

whenever the matrix multiplications make sense. Therefore we have:

$$\begin{aligned}x^T\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]x &= \text{Trace}\left[x^T\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]x\right]\\ &= \text{Trace}\left[\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]xx^T\right]\,. \qquad (1.16)\end{aligned}$$

It thus follows that:

$$\text{Err}(\mathcal{F}, x) = \sigma^2\left(1 + \text{Trace}\left[\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]xx^T\right]\right)^{8}\,.$$

Now let us compute $\text{Err}(\mathcal{F})$, which we write as:

$$\text{Err}(\mathcal{F}) = \mathbb{E}_X\left[\text{Err}(\mathcal{F}, X)\right]\,.$$

Therefore we need to compute the expectation with respect to a test point $X = x$ of the quantity (1.16). Since the trace is just a summation and expectation is linear, we can interchange them. Let us also assume that $\mathbb{E}_X[X] = (0, \ldots, 0)$ so that $\text{cov}(X) = \mathbb{E}[XX^T]$. We then have

$$\begin{aligned}\mathbb{E}_X\left\{\text{Trace}\left[\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]XX^T\right]\right\} &= \text{Trace}\left\{\mathbb{E}_X\left[\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]XX^T\right]\right\}\\ &= \text{Trace}\left\{\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]\mathbb{E}_X[XX^T]\right\}\\ &= \text{Trace}\left\{\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]\text{cov}(X)\right\}\,. \qquad (1.17)\end{aligned}$$

At this point we know that

$$\mathbb{E}_T[\mathbf{X}\mathbf{X}^T] = N\text{cov}(X)\,,$$

where we have the factor N instead of $N-1$ because we are assuming that $\mathbb{E}_X[X] = (0, \ldots, 0)$ and that we know this fact. Now, we would like to assert that

$$\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}] \propto N^{-1}\text{cov}(X)^{-1}\,.$$

[8] Thanks for Yani Udiani for helping to make this part on the trace clearer.

However, as was pointed out in class, this is not always true[9]. One might try to argue if N is very large, then $\mathbf{X}\mathbf{X}^T \approx N\text{cov}(X)$ with a small variance, but at this time I am not sure if this can be made rigorous. For now, one regime in which we can make things rigorous is the following. Suppose that each training data point $\{x_i\}_{i=1}^N$ is sampled independently from the normal distribution with zero mean and $d \times d$ covariance matrix $\Sigma = \text{cov}(X)$, i.e.,

$$x_i \sim \mathcal{N}(\mathbf{0}, \Sigma) \, ,$$

where $\mathbf{0} = (0, \ldots, 0)$. In this case we have [7],

$$\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}] = (N-d-1)^{-1}\Sigma^{-1} = (N-d-1)^{-1}\text{cov}(X)^{-1} \, .$$

Therefore, picking up from (1.17), we have:

$$\begin{aligned}
\text{Trace}\left\{\mathbb{E}_T[(\mathbf{X}\mathbf{X}^T)^{-1}]\text{cov}(X)\right\} &= \frac{1}{N-d-1}\text{Trace}\left\{\text{cov}(X)^{-1}\text{cov}(X)\right\} \\
&= \frac{1}{N-d-1}\text{Trace}[\mathbf{I}] \\
&= \frac{d}{N-d-1} \, .
\end{aligned}$$

To conclude, when we have a linear label model $y = \langle x, \theta_0 \rangle + \varepsilon$, and when our data points $x \in \mathbb{R}^d$ are sampled from the normal distribution with mean $\mathbf{0}$, we have[10]:

$$\text{Err}(\mathcal{F}) = \sigma^2 \left(1 + \frac{d}{N-d-1}\right) ,$$

where $\mathcal{F}$ is the model class of all possible linear models. We thus see that to have a small expected test error (modulo the irreducible error) the number of training points N must grow essentially linearly in the dimension d, thus circumventing the curse of dimensionality (recall the earlier examples of the cube Q with the uniform distribution and 1-nearest neighbor, in which the number of training points grew as a power of d).

On the other hand, we imposed a very heavy assumption on the data generation process, namely that the labels y be linear functions (plus noise) of the data points x. The k-nearest neighbors algorithm imposes no such restriction and in the limit of infinite training data can approximate nearly any model, but N must grow exponentially fast in the dimension d, which is unrealistic. The field of machine learning has developed a number of algorithms "in between" linear regression/classification and k-nearest neighbors, with the goal of increasing the capacity of the model class while not too drastically increasing the complexity of the search space and thus being restricted by the curse of dimensionality. In Section 1.5 we briefly discuss some of the non-deep learning approaches along these lines.

[9]Thanks to Anna Little and Dylan Molho for pointing out an error in the original calculation, and thanks to Mohit Bansil, Anna Little, Gautam Sreekumar, and Ali Zare for valuable discussions thereafter.

[10]If anyone can generalize this calculation further, I would be interested in seeing that!

1.5 Towards deep learning

1.5.1 Dictionaries and kernels

One way to incorporate nonlinearity is through dictionaries and kernels. We give a brief overview here. A *dictionary* takes in a data point $x \in \mathcal{X}$ and maps it into a Hilbert space $\mathcal{H}$. Often the Hilbert space is usually $\mathcal{H} = \mathbb{R}^m$, so let us assume that is the case here. The resulting representation of x is denoted:

$$\Phi : \mathcal{X} \to \mathbb{R}^m \,, \quad x \mapsto \Phi(x) = (\phi_k(x))_{k=1}^m \,, \quad \phi_k : \mathcal{X} \to \mathbb{R} \,.$$

Now with the representation $\Phi(x)$, we can apply linear regression but to the representation:

$$f(x;\theta) = \langle \Phi(x), \theta \rangle = \sum_{k=1}^{m} \theta(k)\phi_k(x) \,. \tag{1.18}$$

More generally, any algorithm that only depends upon $\langle x, \theta \rangle$ can be recast be replacing x with $\Phi(x)$. The resulting model class $\mathcal{F}$ or hypothesis class $\mathcal{P}$ thus depends strongly on the choice of dictionary Φ. The constituent elements of $\Phi(x)$, the functions $\phi_k(x)$, are often referred to as "features." Because of the importance of these features, feature engineering has been and continues to be an important topic in machine learning, although "hand crafted" features are becoming less popular. Nevertheless, if one can come up with a dictionary $\Phi(x)$ for which the label $y(x)$ is a linear function of $\Phi(x)$, then the linear model over the dictionary given by (1.18) will work. Note that the dimension of the problem is now m instead of d, but from Section 1.4.3 we know that linear models circumvent the curse of dimensionality in that the number of training points need only grow linearly with the dimension. In this case that means we need $N = O(m)$ training points. If m is not too much larger than d, then we will be in an favorable situation.

Kernels and deep learning do not specify the features directly, but they do so in different ways. To understand kernel methods, let us define a kernel $K(x, x')$ in terms of a dictionary $\Phi(x)$ as:

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle \,.$$

The *kernel* $K(x, x')$ measures the similarity between x and x' through the lens of Φ; the larger $K(x, x')$, the more similar x and x'. A kernel regression computes:

$$f(x;\alpha) = \sum_{i=1}^{N} \alpha(i) K(x, x_i) \,, \quad \alpha \in \mathbb{R}^N \,, \tag{1.19}$$

where we have assumed the kernel is symmetric, meaning $K(x, x') = K(x', x)$. In other words, $f(x;\alpha)$ predicts the label of x by comparing x to each training point x_i through $K(x, x_i)$ and taking a weighted sum of the resulting similarity measures. Notice that we can rewrite (1.19) as:

$$\begin{aligned} f(x;\alpha) = \sum_{i=1}^{N} \alpha(i) K(x, x_i) &= \sum_{i=1}^{N} \alpha(i) \langle \Phi(x), \Phi(x_i) \rangle \\ &= \sum_{i=1}^{N} \alpha(i) \sum_{k=1}^{n} \phi_k(x) \phi_k(x_i) \\ &= \sum_{k=1}^{m} \left[\sum_{i=1}^{N} \alpha(i) \phi_k(x_i) \right] \phi_k(x) \\ &= \sum_{k=1}^{m} \theta(k) \phi_k(x) \, , \end{aligned} \tag{1.20}$$

where we have set

$$\theta(k) = \sum_{i=1}^{N} \alpha(i) \phi_k(x_i) \, .$$

Since $\alpha \in \mathbb{R}^N$ and $\theta \in \mathbb{R}^m$ are learned from the training data, we see that (1.18) and (1.19) are equivalent.

Notice, though, that (1.19) can be defined for any kernel K, not just those for which $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$. On the other hand, many algorithms depend on just the inner product $\langle x, x' \rangle$ to measure the similarity between two data points. If we want to replace these inner products with $K(x, x')$ and still have the algorithm behave well, we need to at least know that $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$ for some Φ, but we do not need to know Φ. This is the key observation of *kernel learning* [8]. Kernel learning allows one to implicitly use very complicated representations $\Phi(x)$ via what are often simple kernels $K(x, x')$. For example, the polynomial kernel

$$K(x, x') = (\langle x, x' \rangle + c)^m \, ,$$

implicitly defines the polynomial model class of degree m. Another example is the Gaussian kernel

$$K(x, x') = e^{-\|x - x'\|_2^2 / 2\sigma^2} \, ,$$

which implicitly defines a nonlinear infinite dimensional dictionary $\Phi(x)$!

Because of this observation and the usefulness of kernels and the simplicity of incorporating them into machine learning, there is a well developed mathematical theory for determining when a kernel $K(x, x')$ implicitly defines a dictionary $\Phi(x)$ and thus can be written as $K(x, x') = \langle \Phi(x), \Phi(x') \rangle$. The summary of this theory is that K must be a *reproducing kernel* and thus generate a *reproducing kernel Hilbert space (RKHS)*. This is also not easy to verify directly, but it turns out that K is a reproducing kernel if and only if it is *postitive semi-definite*, which means that for any N and any $\{x_i\}_{i=1}^N \subset \mathcal{X}$ and any $c \in \mathbb{R}^N$,

$$\sum_{i,j=1}^{N} c(i) c(j) K(x_i, x_j) \geq 0 \, .$$

For more details we refer the reader to [8]; see also the course CMSE 820 (my old version here or newer versions by Prof. Yuying Xie), which has thus far covered this topic every year.

Kernels allow us to implicitly use a nonlinear dictionary $\Phi(x)$ without having to specify $\Phi(x)$, but the dictionary is not learned. The kernel $K(x,x')$ specifies a unique representation $\Phi(x)$, and as the above calculation (1.20) shows the learning aspect specifies the parameters of the model, but not the representation. Deep learning takes an alternate approach, in which the features themselves are parameterized, and these parameters are learned. Let $\theta = (\boldsymbol{\theta}, \mathbf{w}) \in \mathbb{R}^n$ be the parameters of a deep network in which the last (output) layer computes a linear regression. Then, drawing an analogy with (1.18) we can write this network as:

$$f(x;\theta) = \langle \Phi(x;\boldsymbol{\theta}), \mathbf{w} \rangle \,,$$

where the vector $\boldsymbol{\theta}$ parameterizes the representation $\Phi(x;\boldsymbol{\theta})$ and the vector $\mathbf{w}$ gives the weights of the linear regression over this representation. Thus when fitting the optimal $\widehat{\theta}$ to training data, we not only learn how to combine the features via $\mathbf{w}$, but we learn the features themselves through the parameter vector $\boldsymbol{\theta}$. This will allow for very powerful learning algorithms, but because of the features dependence on the parameters $\boldsymbol{\theta}$ it has made understanding why deep learning works a much more difficult problem than standard dictionary or kernel approaches. Nevertheless, some progress has been made and new results are arriving every month. In the rest of this course we will explore some of these results.

1.5.2 Logistic regression and nonlinearity

Logistic regression is a learning algorithm for categorical classes (i.e., a finite number of classes), that adds a nonlinear function to linear models in order change the output into a probability. Given a new test point $x \in \mathcal{X}$, it does not make a hard decision about which class x belongs to, but rather, for each possible class, gives the probability of x belonging to that class. It is a very simple version of a neural network.

The way logistic regression does this is with the *sigmoid function*, which is defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}}\,, \quad z \in \mathbb{R}\,.$$

Suppose now that $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0,1\}$, i.e., our data points are d-dimensional vectors and we are working on a binary classification problem. Logistic regression defines a parameterized hypothesis space $\mathcal{P}$ of conditional probability density functions $p_{Y|X}(y \mid x, \theta)$, where $\theta \in \mathbb{R}^d$ and

$$\begin{aligned} p(y=1 \mid x,\theta) &= \sigma(\langle x,\theta\rangle) = \frac{1}{1+e^{-\langle x,\theta\rangle}} \\ p(y=0 \mid x,\theta) &= 1 - p(y=1 \mid x,\theta)\,. \end{aligned}$$

To solve for the parameters θ, we can use maximum likelihood estimation (MLE) from Section 1.1.3, or maximum a posteriori (MAP) from Section 1.3.2. To keep things simple, we explain with MLE. Suppose we are given a training set $T = \{(x_i, y_i)\}_{i=1}^N$ in which the x_i's are sampled independently from $p_X(x)$. Recall that the MLE model is:

$$\begin{aligned}\widehat{\theta} = \arg\max_{\theta \in \mathbb{R}^d} p_{X,Y}(T \mid \theta) &= \arg\max_{\theta \in \mathbb{R}^d} \prod_{i=1}^N p(y_i \mid x_i, \theta) \\ &= \arg\max_{\theta \in \mathbb{R}^d} \prod_{i=1}^N [\sigma(\langle x_i, \theta \rangle)]^{y_i} [1 - \sigma(\langle x_i, \theta \rangle)]^{1-y_i} \\ &= \arg\max_{\theta \in \mathbb{R}^d} \sum_{i=1}^N [y_i \log \sigma(\langle x_i, \theta \rangle) + (1 - y_i) \log(1 - \sigma(\langle x_i, \theta \rangle))] \,.\end{aligned}$$

Unlike linear regression, $\widehat{\theta}$ cannot be solved for analytically and in closed form. However, one can compute the gradient of

$$\mathcal{L}(\theta) = -\sum_{i=1}^N [y_i \log \sigma(\langle x_i, \theta \rangle) + (1 - y_i) \log(1 - \sigma(\langle x_i, \theta \rangle))]$$

analytically, which allows for efficient minimization using tools from optimization (e.g., gradient descent). Indeed,

$$\nabla_\theta \mathcal{L}(\theta) = \sum_{i=1}^N [\sigma(\langle x_i, \theta \rangle) - y_i] x_i \,.$$

Logistic regression can be extended to more than two classes; the resulting algorithm is called *softmax regression*. Suppose $\mathcal{Y} = \{1, \ldots, M\}$, i.e., we have M classes. For each class, we learn weights $\theta_m \in \mathbb{R}^d$, $1 \leq m \leq M$. Softmax regression then assigns probabilities as:

$$p\left(y = m_0 \mid x, \{\theta_m\}_{m=1}^M\right) = \frac{e^{-\langle x, \theta_{m_0} \rangle}}{\sum_{m=1}^M e^{-\langle x, \theta_m \rangle}} \,.$$

Softmax regression is often used in the last layer of a neural network trainied for classification, but the relevance goes further. Logistic regression and softmax regression show the importance of nonlinearity, and in particular, having a nonlinear function of a linear transformation. Neural networks build upon this by cascading may successive alternations of linear (or affine) functions and nonlinear functions, such as the sigmoid.

Bibliography

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations*, 2015.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[4] Larry Greenemeier. AI versus AI: Self-taught AlphaGo Zero vanquishes its predecessor. *Scientific American*, October 18, 2017.

[5] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. arXiv:1803.08823, 2018.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer-Verlag New York, 2nd edition, 2009.

[7] J. Hartlap, P. Simon, and P. Schneider. Why your model parameter confidences might be too optimistic - unbiased estimation of the inverse covariance matrix. *Astronomy and Astrophysics*, 464(1):399–404, 2007.

[8] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. The MIT Press, 2002.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*, San Diego, CA, USA, 2015.