Introduction to Supervised Machine Learning

February 4, 2019

1 Classification and Regression

Machine learning develops algorithms that discover patterns in data. We consider the following examples of two different types of supervised machine learning, classification and regression, drawn from computer vision.

Examples:

- **Classification:** The US Postal Service (USPS) uses digit recognition, a machine learning technique, to read handwritten ZIP codes on envelopes. Many thousands or even millions of images of the digit '0' are provided, together with the label '0', and similarly for the other nine digits. These images are provided by the USPS, and someone painstakingly looks at each image and records a label for it in a file. The machine learning algorithm must identify some commonalities among all images of the same digit (these commonalities are the "pattern" in the data) so that if a previously unseen image shows up without a label, it can be automatically labeled without human intervention. The way the machine learning algorithm captures the "pattern" in practice is by computing a function that takes an image as input and produces the appropriate label as output.
- **Regression:** YouTube may be interested in an automatic method to predict the median age of the viewers of each given video clip, so that it can target ads it shows with each video more specifically and cost-effectively. In this scenario, YouTube may collect data through surveys, perhaps by asking viewers to state their ages voluntarily when they access a new clip. Individual responses may not be very reliable, but if there are enough of them per video clip, the empirical median might be an accurate summary. The machine learning algorithm must identify some pattern that connects properties of each video to the median age of its viewers. Once that pattern is found, it can be used to predict median ages of viewers without including the annoying survey.

1.1 Classification

Let us formalize what happens in the classification example above: The machine learning algorithm takes a *training set* as input, of the form

$$T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \quad \text{with} \quad \mathbf{x}_n \in X \quad \text{and} \quad y_n \in Y \ ,$$

where each *data point* $\mathbf{x}_n$ is an image and the corresponding y_n is that image's *label* (a digit between 0 and 9 in the example). Each of the thousands or millions of ordered pairs $(\mathbf{x}_n, y_n)$ is called a *training sample*. The set X is the *data space*, and in the example it is the set of *all possible* images (of some appropriate format), not just the set of training data points. The *label set* Y is the set of all digits

$$Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

in the example. The fact that the elements of Y are numbers (in this case) is irrelevant. Think of labels as abstract identifiers.

The output of the machine learning algorithm is a *function* h that takes an image $\mathbf{x}$ and returns a label y, that is,[1]

$$h \ : \ X \rightarrow Y \quad \text{such that} \quad h(\mathbf{x}) = y \ . \tag{1}$$

This function is the *classifier* or *predictor* learned by the machine learning algorithm. Importantly, while the classifier h "works" even if $\mathbf{x}$ is one of the training samples $\mathbf{x}_n$ (and hopefully yields the appropriate label y_n at least most of the time), in real life h will be typically applied to new, previously unseen images $\mathbf{x}$ for which no labels are available, and produce labels y for them that are hopefully correct most of the time: All the manual work that goes into taking and labeling images in order to make the training set T pays off in the form of a classifier h that can do the labeling work for you from now on. The machine learning algorithm somehow found the "pattern" in the labeled digit images (the "data"), and captured that pattern in the classifier h.

For computer scientists, there is little difference between an algorithm and a function: Every algorithm that takes inputs and produces outputs can be represented by a mathematical function, and every mathematical function can be implemented, at least approximately, by some algorithm. Thus, the machine learning algorithm itself can be viewed as a function that takes a set T as input and produces a function h as output. In order to do so, the algorithm needs to know what functions h with the signature given in expression 1 it is allowed to choose from. The set $\mathcal{H}$ of such functions is called the *hypothesis space* for the given learning problem.[2] Then, the signature of the machine learning function λ is

$$\lambda \ : \ \mathcal{T} \rightarrow \mathcal{H} \quad \text{such that} \quad \lambda(T) = h$$

and we say that λ *learns* or *trains* the classifier h from the training set T. Applying a classifier to data unseen during training is called *testing* the classifier. Applying a classifier to any data is called *inference*.

If a function that outputs a function makes your head spin, think of both λ and h as algorithms: You write an algorithm λ that somehow looks at the training samples in the training set T and produces a piece of code h. That piece of code can be run on new images $\mathbf{x}$ to guess their labels, $y = h(\mathbf{x})$.

Even this "algorithmic" view is a bit abstract. What happens most of the time is that the machine learning algorithm outputs the vector $\boldsymbol{\theta}$ of *parameters* of a parameterized algorithm $h_{\boldsymbol{\theta}}$. Think of $h_{\boldsymbol{\theta}}$ as a piece of code that is known ahead of time (we write that code), but that is missing the values of some parameters $\boldsymbol{\theta}$. The machine learning algorithm λ knows about $h_{\boldsymbol{\theta}}$, and only outputs the vector $\boldsymbol{\theta}$. Hopefully this more concrete view of things eases your anxiety. Mathematically, it is notationally simpler and conceptually more general to think of λ as producing the whole function h.

It is useful to observe that if the label set Y has K labels, a classifier defines a *partition* of X into K subsets $X_1, \ldots, X_K$ by the following rule[3]:

$$\mathbf{x} \in X_y \ \Leftrightarrow \ h(\mathbf{x}) = y \quad \text{for} \quad y \in Y \ .$$

Thus, another way to view the process of learning a classifier is that the training algorithm λ partitions the set X of all possible inputs (not just the set of training data points!) into K subsets,

[1]The "right arrow" description of a function h in terms of its domain X and codomain Y, as given in expression 1, is called the *signature* of the function.

[2]For this and other terminology in machine learning, we defer to common use in the literature. All terms have some historical justification, even when they sound a bit strange.

[3]The double arrow reads "if and only if."

with each subset assigned to one of the labels. The classifier h is a piecewise-constant function, and the "pieces" are the subsets X_y.

Practical Aspects: Making a Training Set. It is useful to examine what it takes to produce the images $\mathbf{x}_n$ for the classification example above. One could place a camera above one of the conveyer belts that move envelopes around in a USPS distribution center, and take a picture of each envelope. That picture, however, contains much more than a digit, and someone must therefore sit at a computer, display each envelope image on the screen in turn, drag a bounding box around each digit with a mouse, and save each cropped digit as a separate file. Another file will record the name of the image and the corresponding label (what digit the image represents). This collection of data is the training set T.

Different images may have different quality depending on factors such as lighting, the ink used to write the digit, the color of the envelope paper, the size of the digit, and so forth. In addition, the digit images may be cropped inconsistently if the operator gets tired, or multiple operators are employed to crop and label images. These inconsistencies often make learning harder, and training sets are sometimes *curated* because of this. In the USPS example, curation involves running manual procedures to make the images more uniform in terms of the factors mentioned above.[4] Curation helps machine learning research by making the problem easier for initial study. However, curation cannot be used to train real-life machine learning classifiers, because the whole point of computing h is to eliminate manual intervention during deployment (testing). In these cases, only automatic *preprocessing* of the images is a viable option, because the preprocessing can then be applied during both training and testing. Preprocessing techniques are beyond the scope of these notes.

In the days of Google Images and web crawlers, collecting data for training is often inexpensive if no curation is needed. Labeling, on the other hand, is laborious, time-consuming, and error-prone. A common method for labeling large amounts of data is to publish it to the Amazon Mechanical Turk, an online marketplace for the type of repetitive work involved in labeling. People around the world access the marketplace searching for easy jobs, and you pay them a small amount (often 1-3 cents) per label. Even so, labels are not always of good quality, and various schemes are devised to address this issue. For instance, multiple people could be employed to label each training sample, and a majority vote could then be taken to determine the most likely label. In any case, the cost of labeling is high, and many machine learning algorithms require large amounts of data. This high cost is arguably the main hurdle to a broader use of machine learning.

1.2 Regression

Formally, the problem posed in the YouTube regression example given earlier is nearly identical to the classification problem: Discover a pattern in the training data (video clips and corresponding recorded median viewer ages) to learn a function h that predicts median ages for previously unseen (and unlabeled) video clips. As a result, most of the considerations and terminology from the previous section still hold.

The difference between classification and regression is that the function h outputs a *categorical* variable for classification and a *real-valued* variable for regression. This means that the label set Y is finite and unstructured for classification, while for regression we have

$$Y = \mathbb{R} \; ,$$

the set of real numbers, endowed with a metric (it makes sense to talk about the "distance" between two ages). Incidentally, while a natural distance exists for digits, the distance is irrelevant

[4]Think calibrating, touching up, and re-cropping or resizing images in Photoshop.

for classification: Knowing that $|2-3|$ is smaller than $|2-5|$ does not help distinguish the two digits from each other. Thus, any distance that may exist in Y is not *used* in classification, even it there happens to be a natural one.

In terms of definitions, the difference between classification and regression is minor. In practice, however, the finite nature of the label set often suggests techniques for classification that could not be applied to regression. Conversely, regression techniques typically rely on the chosen metric for Y, and these techniques cannot be used for classification. The separation between regression and classification is not always strong, as some machine learning algorithms can be applied to either problem with minor changes.

The predictor h output by a machine learning algorithm for regression is called a *regressor*, rather than a classifier, and the word *value* is used for Y instead of label. All other considerations made earlier for classification hold for regression as well. A *target* is either a label or a value.

2 Training, Validation, and Testing

This Section gives a first, tentative formalization of training as a problem of empirical risk minimization. It then shows that this formulation is insufficient for machine learning, and introduces the concept of validation and testing to address these limitations.

Training as Empirical Risk Minimization When a predictor h predicts value $h(\mathbf{x})$ for a data point $\mathbf{x}$ and the true value associated with $\mathbf{x}$ is y, we experience a *loss* $\ell(y, h(\mathbf{x}))$. The loss is zero when $h(\mathbf{x}) = y$ and positive otherwise. The loss function has signature

$$\ell \;:\; Y \times Y \to \mathbb{R}_0^+$$

where $\mathbb{R}_0^+$ is the set of nonnegative real numbers. For instance, the *zero-one loss* is very commonly used for classification:

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

For regression, the *square loss* is a popular choice:

$$\ell(y, h(\mathbf{x})) = (h(\mathbf{x}) - y)^2 \;.$$

The *empirical risk* over the training set T of data point/value pairs is the average loss over that set:

$$L_T(h) \stackrel{\text{def}}{=} \frac{1}{|T|} \sum_{(\mathbf{x},y)\in T} \ell(y, h(\mathbf{x})) \;.$$

Given a training set T and a hypothesis space $\mathcal{H}$, *Empirical Risk Minimization* (ERM) is the problem of finding a predictor in $\mathcal{H}$ with the lowest empirical risk[5] on T:

$$\text{ERM}_T(\mathcal{H}) \in \arg\min_{h\in\mathcal{H}} L_T(h) \;.$$

[5]We are assuming that the minimum in this expression exists. Discussion of this assumption is beyond the scope of these notes.

Notation: This notation should be familiar by now. To recall, the quantity $\min_{h\in\mathcal{H}} L_T(h)$ is a single (non-negative) number, the smallest achievable risk over all choices of h in $\mathcal{H}$. The notation $\arg\min_{h\in\mathcal{H}} L_T(h)$ represents the *set* of all functions in $\mathcal{H}$ that achieve that minimal risk (there could be more than one such function). Finally, $\text{ERM}_T(\mathcal{H})$ is one of these functions, it does not matter which.

ERM is a *fitting* problem, as it seeks a function h that best fits the data in T.

Generalization What is the point of fitting a predictor to a set of data if we only look at how well the predictor does on that data, as the risk L_T does? A predictor estimates a value $\hat{y}$ from a data point $\mathbf{x}$, but we already know the correct answer y: What is the point of estimating the answer again?

This question goes to the key difference between data fitting and machine learning: Data fitting is asked to do well on the training set, and is used when something about the predictor itself is of interest. Perhaps an economist postulates that price and demand are linearly correlated, and wants to determine the regression coefficients. The goal of such a study is not so much to predict new prices for new levels of demand, but rather to use the slope of the regression line to understand the effects of a demand fluctuation on changes of price.

In contrast, machine learning focuses on prediction, and is asked to do well *on previously unseen data.* The parameters of the predictor are not of interest *per se*, and the goal is instead to estimate values $\hat{y}$ corresponding to new data points $\mathbf{x}$. A predictor that does this well is said to *generalize well.* What can we say about new data points, that is, data we have not yet seen? Specifically, how can we formalize the notion of "doing well on previously unseen data?" Since prediction is the goal, rather than fitting, the empirical risk $L_T(\hat{h})$ is not a good measure of performance. How else can we measure prediction performance? Part of this note focuses on this question.

Hyper-Parameters A closely related issue is how to determine what are called the hyper-parameter(s) of a predictor. For instance, when the hypothesis space $\mathcal{H}$ is a set of polynomials, one needs to specify the maximum allowed degree k, which is called the hyper-parameter of a polynomial regressor. Once k is given, training finds the best *parameters*, that is, coefficients $c_0, \ldots, c_k$ for the polynomial. How can we compute a good value for k?

A first idea is to incorporate k into the optimization performed for training the predictor, that is, optimize jointly over parameters and hyper-parameters. This approach would eliminate the distinction between parameters and hyper-parameters. However, there are two difficulties in doing so. The first one is technical: The coefficients of a polynomial are real-valued, while k is a positive integer, so we may need different techniques. However, this issue is minor: We could just try values of $k = 1, 2, \ldots$ and look for one that yields good results.

The second difficulty, on the other hand, is fundamental and unsurmountable: *The answer from optimization would be trivial and useless.* Specifically, in the case of polynomials, we know that we can interpolate $k + 1$ points exactly with a polynomial of degree k, so the bigger k, the better, up to a limit: Just set $k = N - 1$, where N is the size of the training set, and the resulting training error will be, again, exactly zero.

In other words, for polynomial fitting we can drive $L_T(h)$ to zero with a suitable choice of hyper-parameter, but this does not tell us anything about how well we will do *on data we have not yet seen.* Again, the ERM risk is not adequate for determining hyper-parameters.

Thus, we face at least three broad problems, which we discuss in the sections that follow:

1. How to think about "previously unseen data." The answer is probabilistic, in the form of what is called a *generative data model* (Section 2.1).
2. How to choose good hyper-parameters, if $\mathcal{H}$ has any. This problem is called *model selection*, and a common technique for it is called *validation* (Section 2.2).
3. How to evaluate the generalization performance of a predictor. This is called *testing* (Section 2.3).

2.1 Previously Unseen Data: A Generative Data Model

A predictor has no chance to do well on data that is different from the data it was trained on unless the training data bears some resemblance or connection to the new data. For instance, it would not be fruitful to train a predictor on human faces and evaluate its performance on handwritten digits, or even on cat muzzles. Performance is likely to be very bad in those cases. More subtly, if the human faces in the training set are all frontal snapshots and the resulting classifier is tested on profiles of human heads, we cannot hope for the classifier to do well. Generalization performance may be poor even if the test images are of the same type as the training images, but perhaps taken under systematically different lighting. We need some way to formalize the notion of "resemblance or connection" between two data sets.

A probabilistic formulation of machine learning provides the conceptual link between "seen" and "unseen" data. Specifically, one assumes that all samples $(\mathbf{x}, y)$, both those in the training set and those seen after training, are drawn independently and at random from some joint probability distribution $p(\mathbf{x}, y)$ called the *generative model of the data*, or "model" for short. The model is the link between training and testing data. Of course, when we deploy a trained predictor we only see the data point $\mathbf{x}$, and it is the predictor's job to guess the corresponding target $h(\mathbf{x})$. However, we can think of the true target y as being "out there:" we just don't have access to it. Thus, it makes mathematical sense to ask what the loss $\ell(y, h(\mathbf{x}))$ is even during deployment. In this spirit, the goal of machine learning is not to minimize the empirical risk, bur rather the *risk*:

$$L_p(h) = \mathbb{E}_p[\ell(y, h(\mathbf{x}))] \; ,$$

that is, the statistical expectation of the loss over the model p. This is a measure of how poorly h does not just over the training set, but rather over all possible data drawn from the model. Sometimes, the risk is called *statistical risk* for emphasis, when it is necessary to distinguish it from the empirical risk. The problem of machine learning is then (Statistical) Risk Minimization, that is, the problem of finding

$$\mathrm{RM}_p(\mathcal{H}) \in \arg\min_{h \in \mathcal{H}} L_p(h) \; ,$$

a much taller order than ERM. The predictor $\mathrm{RM}_p(\mathcal{H})$ is an *optimal predictor in* $\mathcal{H}$ and the risk it achieves is the *lowest risk on* $\mathcal{H}$, denoted by

$$L_p(\mathcal{H}) \overset{\text{def}}{=} \min_{h \in \mathcal{H}} L_p(h) \; .$$

Notation: Here and elsewhere, the subscript for risks and predictors (that is for L, ERM, and RM) denotes what the risk is computed on, so the notational distinction between ERM and RM is redundant: A risk computed on a data set is always an empirical average, and a risk computed over a model is always a statistical mean. Nonetheless, this distinction is maintained for both emphasis and consistency with the literature. The term in parentheses denotes either a predictor (h) or a hypothesis space ($\mathcal{H}$). In the latter case, the risk is the smallest over all predictors in that hypothesis space. Thus, $L_p(\mathcal{H})$ is a minimum over a set, while $L_p(h)$ is not.

If the optimal predictor $\text{RM}_p(\mathcal{H})$ in $\mathcal{H}$ does well (on average, on all possible data), then it does well also on the training set T only to the extent that T is a statistically representative sample of the data model p. If this is not the case, then there is no immediate relation between the performance of $\text{RM}_p(\mathcal{H})$ and that of $\text{ERM}_T(\mathcal{H})$: The training set T may just be a "fluke," that is, have low probability in p, so how well we do on T may not say much about how we do on other data drawn from p.

Unfortunately, T being a "fluke" is rather common: The data points $\mathbf{x}$ often represent complex objects (images or emails or web pages, for instance) and therefore live in a data space $X \subseteq \mathbb{R}^d$ with many dimensions ($d \gg 1$). It is then effectively impossible for the training set T to fill any significant part of X to any reasonable extent, a problem called the *curse of dimensionality.* As an example, building a grid with a mere 10 cell boundaries per dimension requires 100 cells on the plane ($\mathbb{R}^2$), but 10^{100} in a space with 100 dimensions ($\mathbb{R}^{100}$). Since the number of atoms in the universe is estimated to be between 10^{78} and 10^{82}, working with a grid in 100 dimensions is clearly impossible. Similar difficulties arise when estimating probability densities or enumerating class boundaries. In short, Machine learning is quite often starved of data.

The Model is Unknown If the empirical risk $L_T(h)$ on the training set T is not a good indicator of performance, why not just use as indicator the lowest statistical risk $L_p(\mathcal{H})$ on the hypothesis space $\mathcal{H}$ according to the model p? Given a data model p and a hypothesis space $\mathcal{H}$, we could just find an optimal predictor on $\mathcal{H}$ and its corresponding statistical risk, and we are done.

The fly in the ointment is that *we typically do not know the data model* p: What is, for instance, the probability distribution over the set of all possible images? Or over all possible English sentences? All we usually have is data, and it may well be that the training set T is all the data we have. Even if we have large amounts of data, estimating probability distributions over spaces with many dimensions is hopeless. Thus, for all but the simplest cases, *the optimal predictor $RM_p(\mathcal{H})$ is beyond computing.*

A deeper question then arises: If we cannot know p, what use is there in introducing it? There are several answers to this question.

- While $\text{RM}_p(\mathcal{H})$ itself may be beyond reach, we can try and *estimate* it. There will be some uncertainty around the estimate, but that's better than nothing. For the estimate we need data, but we know how to use data to estimate statistical means—and risks are means—even if we don't know the underlying distribution.

- We may be able to *bound* some of these quantities. For instance, can we compute two numbers that bound the lowest risk on a given hypothesis space $\mathcal{H}$ from above and from below, and *over all possible choices of model* p? To do so, we don't need to know a particular p, but just how to marginalize over all of them.

- Perhaps most importantly, it is hard to find a way to link training and testing data that is conceptually cleaner and simpler than introducing p. In some sense, assuming a probabilistic model is conceptually the right thing to do, and we can think of p as the holy grail we only know of indirectly and keep referring to.

A useful mental picture of the model p is that of an oracle that gives out samples from $X \times Y$ upon request. The oracle knows the distribution, but we do not. In addition, samples do not come for free, and a larger set of samples comes at a higher price than a small one. While the actual price is not part of the model, this cost consideration reflects the reality that collecting data points, especially with their targets, is laborious and potentially expensive.

Practical Aspects: Making a Training Set. To get a sense of the difficulty of collecting training samples, it may be useful to examine what it takes to produce the training images $\mathbf{x}_n$ for the task of recognizing handwritten digits.

One could place a camera above one of the conveyer belts that move envelopes around in a USPS distribution center, and take a picture of each envelope. That picture, however, contains much more than a digit, and someone must therefore sit at a computer, display each envelope image on the screen in turn, drag a bounding box around each digit with a mouse, and save each cropped digit as a separate file. Automatic methods won't do: How can we find the digits on the envelope automatically if we haven't yet trained a system that knows how to recognize a digit?

Another file will list the names (file names) of all the images and the corresponding labels (what digit each image represents). This collection of data (images with bounding boxes and labels) is the training set T, and associating a label to every training image is called *annotation* or *labeling*.

Different images may have different quality depending on factors such as lighting, the ink used to write the digit, the color of the envelope paper, the size of the digit, and so forth. In addition, the digit images may be cropped inconsistently if the operator gets tired, or multiple operators are employed to crop and label images. These inconsistencies often make learning harder, and training sets are sometimes *curated* because of this. In the USPS example, curation involves running manual procedures to make the images more uniform in terms of the factors mentioned above.[6] Curation helps machine learning research by making the problem easier for initial study. However, curation cannot be used to train real-life machine learning classifiers, because the whole point of computing h is to eliminate manual intervention during deployment (testing). In these cases, only automatic *preprocessing* of the images is a viable option to make the digits in them easier to recognize, because the preprocessing can then be applied during both training and testing. Preprocessing techniques are beyond the scope of these notes.

In the days of Google Images and web crawlers, *collecting* data for training is often inexpensive if no curation or preprocessing is needed or can be performed by automatic means. Annotation, on the other hand, involves humans and is laborious, time-consuming, and error-prone. A common method for annotating large amounts of data is to publish it to the Amazon Mechanical Turk, an online marketplace for the type of repetitive work involved in labeling. People around the world access the marketplace searching for easy jobs, and you pay them a small amount (often 1-3 cents) per label. Even so, labels are not always of good quality, and various schemes are devised to address this issue. For instance, multiple people could be employed to label each training sample, and a majority vote could then be taken to determine the most likely label. In any case, the cost of labeling is high, and many machine learning algorithms require large amounts of data. This high cost is arguably the main hurdle to a broader use of machine learning.

[6]Think calibrating, touching up, and re-cropping or resizing images in Photoshop.

2.2 Validation: Model Selection

If the hypothesis space $\mathcal{H}$ depends on one or more hyper-parameters, a method is needed to select these. The problem of finding good hyper-parameters is called *model selection*, and we saw that optimizing hyper-parameters over the training set is not an option for accomplishing this task. A very popular technique for model selection is called *validation*, and uses a data set V of the same structure as T (datapoint/value pairs), but disjoint from T. The set V is called the *validation set.*

When the set of possible hyper-parameters is finite, validation can be understood as a nested optimization. Specifically, let Π be a set of possible hyper-parameter vectors. If $\boldsymbol{\pi}$ is a vector of hyper-parameters out of Π (a vector because there may be more than one hyper-parameter), let $\mathcal{H}_{\boldsymbol{\pi}}$ be the corresponding hypothesis space. Then, validation computes

$$\hat{\boldsymbol{\pi}} = \arg\min_{\boldsymbol{\pi}\in\Pi} L_V(\mathrm{ERM}_T(\mathcal{H}_{\boldsymbol{\pi}})) \ .$$

Notation: This formula is quite a mouthful, but the idea is simple, and can be understood by unpacking this expression from the inside-out: Pick each vector $\boldsymbol{\pi}$ of hyper-parameters out of Π in turn. The choice of $\boldsymbol{\pi}$ identifies a particular hypothesis space $\mathcal{H}_{\boldsymbol{\pi}}$. Search this hypothesis space (by a descent method or whatever other technique is appropriate for the given type of predictor) for an optimal estimator

$$\hat{h}_{\boldsymbol{\pi}} \in \mathsf{ERM}_T(\mathcal{H}_{\boldsymbol{\pi}})$$

on the *training* set T. Once $\hat{h}_{\boldsymbol{\pi}}$ is found, compute its empirical risk $L_V(\hat{h}_{\boldsymbol{\pi}})$ on the *validation* set V. When done performing this computation over all choices $\boldsymbol{\pi}$ out of Π, return the hyper-parameter vector $\hat{\boldsymbol{\pi}}$ that has the smallest empirical risk on V. Another way of viewing this validation procedure is through a loop, as shown in Algorithm 1.

Algorithm 1 Model Selection by Validation

```
procedure VALIDATION($\mathcal{H}, \Pi, T, V, \ell$)
    $\hat{L} = \infty$                                              ▷ Stores the best risk so far on $V$
    for $\boldsymbol{\pi} \in \Pi$ do
        $h \in \arg\min_{h'\in\mathcal{H}_{\boldsymbol{\pi}}} L_T(h')$          ▷ Use loss $\ell$ to compute best predictor $\mathrm{ERM}_T(\mathcal{H}_{\boldsymbol{\pi}})$ on $T$
        $L = L_V(h)$                                      ▷ Use loss $\ell$ to evaluate the predictor's risk on $V$
        if $L < \hat{L}$ then
            $(\hat{\boldsymbol{\pi}}, \hat{h}, \hat{L}) = (\boldsymbol{\pi}, h, L)$       ▷ Keep track of the best hyper-parameters, predictor, and risk
        end if
    end for
    return $(\hat{\boldsymbol{\pi}}, \hat{h}, \hat{L})$                       ▷ Return best hyper-parameters, predictor, and risk estimate
end procedure
```

Logically, since the goal of validation is to select a good set of hyper-parameters, the validation procedure only needs to return $\hat{\boldsymbol{\pi}}$, the best set it found. Practically, however, it would be wasteful to then train a predictor on $\mathcal{H}_{\hat{\boldsymbol{\pi}}}$ again, since that predictor was trained during the procedure. Because of this, Algorithm 1 also returns the optimal predictor on the optimal hypothesis space. The validation loss is also returned for informational purposes.

A complication arises when the set Π of possible hyper-parameters is not finite. Even so, this set is often discrete. The validation procedure then scans Π in some order and finds the first *local* optimum, that is, a vector $\hat{\boldsymbol{\pi}}$ whose validation risk $L_V(\hat{h}_{\boldsymbol{\pi}})$ is smaller than previously encountered risk values, and such that subsequent risks are higher. This idea assumes that the dependence of $\mathcal{H}_{\boldsymbol{\pi}}$ on $\boldsymbol{\pi}$ is somehow simple and regular. When Π is not even countable, a grid approximation to it is often scanned in search of a good set of hyper-parameters.

An example of validation with a discrete but infinite set Π is the choice of polynomial degree in polynomial regression. In this case, Π is scanned in the order $k = 1, 2, \ldots$, and the value $\hat{k}$ corresponding to the first (significant) local minimum in the validation risk sequence $L_V(\hat{h}_k)$ is returned. Figure 1 (a) shows the training and validation risk for a polynomial data fitting problem. The blue dots in panel (b) of the Figure are the training data, and the orange dots are the validation data. The Least Squares loss is used to compute risks.

From panel (a) we see that the training risk decreases monotonically. This stands to reason, because a polynomial of higher degree can fit a given set of points better (or at least no worse). The validation risk, on the other hand, is high for $k = 1, 2$, where the training risk is high as well. The degrees of these two polynomials (shown in yellow and purple in panel (b)) are too low to fit the data well. A high training risk is a symptom of underfitting.

The validation risk is rather low for all degrees between $k = 3$ and $k = 8$ inclusive. This means that polynomials of these degrees generalize quite well to the given validation data. The validation risk is minimal when $k = 3$, and the procedure therefore returns $\hat{k} = 3$ as the solution. The corresponding polynomial is shown as a thick green line in panel (b).

The difference between the validation risks for $k = 3$ and $k = 4$ is small. With so few data points for validation, the choice between these two values is rather uncertain.

The gap between training and validation risk in Figure 1 (a) is always positive. This makes sense because the polynomial is fitted to the training data, and the fitting procedure never sees the validation data, on which it is therefore likely to perform less well.

Two dramatic events occur for $k = 9$: The training risk goes to zero, and the validation risk jumps suddenly to a large value. The first event occurs because $k = N - 1$: If the degree of a polynomial is at least the number of data points minus one, a perfect fit can be achieved. Thus, a training risk of zero has to do with polynomials and the size of T. The large upward jump in validation risk, on the other hand, when compared to a much smaller training loss (zero or otherwise), implies overfitting: The magenta polynomial ($k = 9$) in panel (b) follows the vagaries of the specific data sample too closely, and does poorly on previously unseen data as a consequence.

2.3 Testing: Measuring Generalization Performance

Suppose that you train a predictor on a data set T. This procedure may or may not involve validation, depending on whether the hypothesis space $\mathcal{H}$ depends on hyper-parameters or not. Either way, once the optimal predictor $\hat{h}$ has been found, the following question arises: How well can you expect your predictor to do on new data? That is, how well does $\hat{h}$ generalize? This question is what makes machine learning different from data fitting.

If no validation is used, the answer should be rather obvious by now, in light of our previous discussion: We cannot use T to measure generalization, because a low empirical risk is not necessarily an indication of good performance. Instead, we collect a new data set S, called the *test set*, and compute the predictor's empirical risk $L_S(\hat{h})$ on S as an estimate of its statistical risk. None of S is used for training, and all of it used exclusively for *testing*, that is, for measuring the generalization performance of the predictor.

If validation is used, on the other hand, one may be tempted to use the validation risk $L_V(\hat{h})$ as a measure of generalization performance, since this risk is computed on a data set V that has

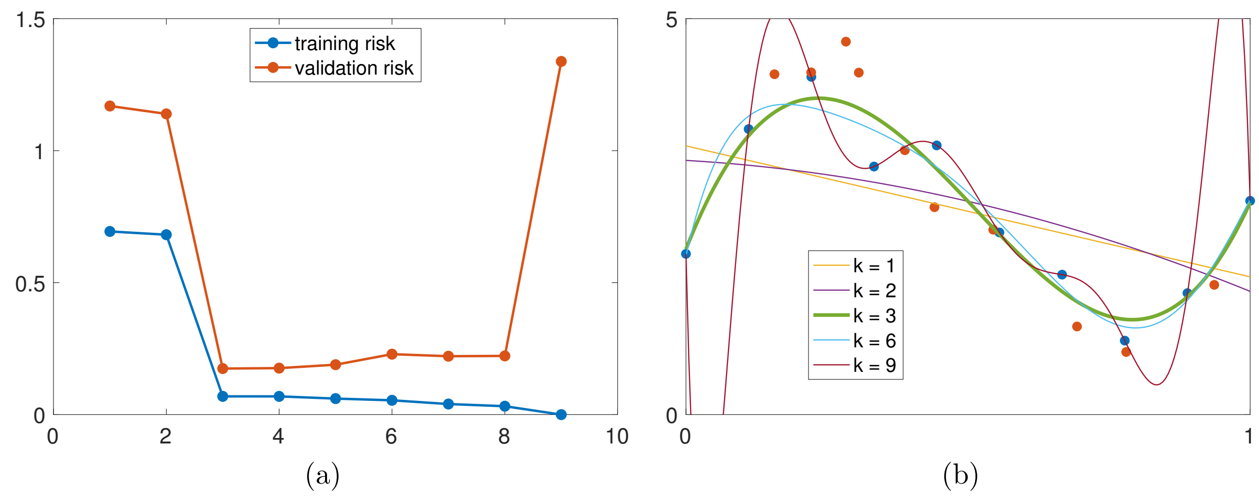

Figure 1: (a) Training and validation risk as a function of the degree k of a polynomial fit to the blue dots in (b). The lowest validation risk is achieved for $k = 3$, while the training risk keeps decreasing even beyond that. (b) Polynomials of degrees between 1 and 9 fit to the training data. Blue points are the training set and orange points are the validation set. Only some of the polynomials are shown, to reduce figure clutter. The polynomial that generalizes best to the validation data has degree $k = 3$ and is shown as a thicker, green line.

not been used for training $\hat{h}$.

However, just as training "taints" the training data, so that validation must be performed on a data set separate from the training set, validation "taints" the validation data. In other words, model selection, just like training, is an estimation procedure that finds an optimal hyper-parameter set $\hat{\boldsymbol{\pi}}$ by evaluating the performance of various choices of $\boldsymbol{\pi} \in \Pi$ on the validation set V. If V is too small, overfitting may occur during validation as well, and $\hat{\boldsymbol{\pi}}$ may adapt too much to the idiosyncrasies of V. Perhaps if we were to use a different V we would obtain a different $\hat{\boldsymbol{\pi}}$. In summary, the issues in validation are analogous to those encountered in training as far as overfitting is concerned.

Thus, we generally need the following three sets:

- A *training set* T to train the predictor given a specific set of hyper-parameters (if any).
- A *validation set* V to choose good hyper-parameters, if the hypothesis space $\mathcal{H}$ depends on any.
- A *test* set S to evaluate the generalization performance of the predictor $\hat{h}$ learned by training on T (and, optionally, by hyper-parameter validation on V).

All these sets contain data point/value pairs $(\mathbf{x}, y)$, that is, they are all subsets of $X \times Y$.

If collecting a training set is difficult (and we saw that it is), collecting *three* data sets is even harder, and may even be unrealistic in applications where each sample is very costly. Even when expense is not an overriding issue, using a separate training set T, validation set V, and testing set S means that we forgo using all the available data $T \cup V \cup S$ for training, and we therefore make the predictor intentionally worse than it could be, in order to hold out data for validation and testing.

Because of these considerations, some *resampling* techniques have been developed that allow using a single data set for both training and model selection. The most popular of these techniques is called cross-validation, and is discussed in the Appendix. On the other hand, *the availability of a test set S, entirely disjoint from all other data sets used during training, is indispensable.* No research article in machine learning is ever accepted if there is even the suspicion that any part of S has been used during training, either directly or indirectly.

3 The State of the Art of Image Classification

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition that pits image recognition and object detection algorithms against each other on a standardized image database that contains 1.4 million images divided into 1,000 object classes. Designing the database and the various aspects of the competition is a significant research effort in its own right [5]. The competition includes three tasks:

Image Classification: The database maintainers label each image manually with the presence of *one* instance out of 1000 object categories, even if more objects are present in the image. These annotations are called the *ground truth.* Competing classification algorithms return a *list* of up to five objects for each test image, and the list is deemed correct if it includes the ground-truth label. Algorithms are trained on a subset of 1.2 million images for which the ground-truth annotations are made public, and about 50,000 images with annotations are made public for validation. The competition organizers withhold a set of 100,000 images to test the algorithms submitted by the contestants, and measure the *error rate* as the fraction of incorrect outputs.[7]

Single-Object Localization: In addition to image classification, algorithms also provide an axis-aligned rectangular bounding box around *one* instance of the recognized object class. The box is correct if the area of the intersection of computed and true box is at least 1/2 of the area of their union. As above, a list of up to five object-box pairs is returned.

Object Detection: Same as single-object localization, but *every* instance of the object class is to be found exactly once. Missing, duplicate, and false detections are penalized.

Winners of the competition for each task are announced and invited to contribute insights at either the International Conference on Computer Vision (ICCV) or the European Conference on Computer Vision (ECCV) in alternate years.

Classification, detection, and localization go hand-in-hand: To classify an image one must first detect and localize an object that is deemed to be "of interest," and a correctly detected and well localized object is of course easier to classify correctly. Conversely, a detector often works by running a classifier at all or many windows in the image. This note only looks at performance figures in image classification, with the *caveat* that the best systems often perform well in more than one category, because of these interdependencies.

The classification task is very difficult for at least the following reasons:

[7]Other measures of error have been proposed in the past and are given as secondary measures in current challenges as well.

- Images are "natural," that is, they are not contrived for the databased but rather downloaded from photo sharing sites like Flickr. Because of this, the objects of interest are on arbitrary backgrounds, and may appear in very different sizes, viewpoints, and lighting conditions in different images, and may be only partially visible.
- There are 1,000 categories, and distinctions between categories are often very subtle. For instance, Siberian husky and Eskimo dog are different categories, but dogs of the two breeds look very similar.
- At the same time, variations of appearance within one category can be significant (how many lamps can you think of?)
- What part of the image is of interest to the labeler may be based on very high-level semantic considerations that a machine learning algorithm may not have access to. For instance, a picture of a group of people examining a fishing rod was labeled as "reel."

Because of these difficulties, the image classification error was 28.2 percent in 2010, even after many years of research and experimentation with smaller databases. The winner of the 2017 challenge, on the other hand, achieved a 2.3% error rate [2]. This dramatic, more than tenfold improvement was achieved through the use of deep convolutional nets. As a reference point, the winning architecture returns the ensemble prediction of several networks. Networks are between 50 and 200 layers deep, and have millions of parameters each. The change from 17% to 2.3% from a previous system [3] was not just achieved by increasing the number of layers, but by several architectural insights developed over the five years between the two architectures. These include residual networks [1] and so-called squeeze-and-excitation networks [2]. We will look at the basics of deep convolutional neural networks, but there is no room in a one-semester course to explore all the nuances of these architectures. Please read the papers if you are curious.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[3] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1106–1114, 2012.

[4] C. Nadeau and Y. Bengio. Inference for the generalization error. In *Advances in Neural Information Processing Systems (NIPS)*, pages 307–313, 2000.

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, pages 1–42, April 2015.

A Cross Validation

In *K-fold cross-validation*, the training data are split once and for all into K (approximately) equal-sized sets V_k chosen at random. For every $k = 1, \ldots, K$, one trains the predictor on all training data points *except* those in V_k. The empirical risk of the predictor on V_k is then recorded. The mean of the K empirical risks is taken as an estimate of the generalization risk, and their variance, if desired, gives an idea of the uncertainty about this estimate. This variance is not available when standard validation (no cross-validation) is used, so this is another advantage of cross-validation. Algorithm 2 details model selection by cross-validation procedurally. Of course, V is no longer a parameter to this procedure.

Algorithm 2 Model Selection by K-Fold Cross-Validation

```
procedure CrossValidation($\mathcal{H}, \Pi, T, K, \ell$)
    $\{V_1, \ldots, V_K\} = \text{Split}(T, K)$    ▷ Split $T$ in $K$ approximately equal-sized sets at random
    $\hat{L} = \infty$    ▷ Will hold the lowest risk over $\Pi$
    for $\boldsymbol{\pi} \in \Pi$ do
        $s, s_2 = 0, 0$ ▷ Will hold sum of risks and their squares to compute risk mean and variance
        for $k = 1, \ldots, K$ do
            $T_k = T \setminus V_k$    ▷ Use all of $T$ except $V_k$ as training set
            $h \in \arg\min_{h' \in \mathcal{H}_{\boldsymbol{\pi}}} L_{T_k}(h')$    ▷ Use the loss $\ell$ to compute $h = \text{ERM}_{T_k}(\mathcal{H}_{\boldsymbol{\pi}})$
            $L = L_{V_k}(h)$    ▷ Use the loss $\ell$ to compute the risk of $h$ on $V_k$
            $(s, s_2) = (s + L, s_2 + L^2)$    ▷ Keep track of quantities to compute risk mean and
variance
        end for
        $L = s/K$    ▷ Sample mean of the risk over the $K$ folds
        if $L < \hat{L}$ then
            $\sigma^2 = (s_2 - s^2/K)/(K-1)$    ▷ Sample variance of the risk over the $K$ folds
            $(\hat{\boldsymbol{\pi}}, \hat{L}, \hat{\sigma}^2) = (\boldsymbol{\pi}, L, \sigma^2)$    ▷ Keep track of the best hyper-parameters and their risk
statistics
        end if
    end for
    $\hat{h} = \arg\min_{h \in \mathcal{H}_{\hat{\boldsymbol{\pi}}}} L_T(h)$    ▷ Train predictor afresh on all of $T$ with the best hyper-parameters
    return $(\hat{\boldsymbol{\pi}}, \hat{h}, \hat{L}, \hat{\sigma}^2)$    ▷ Return best hyper-parameters, predictor, and risk statistics
end procedure
```

Leave-one-out cross-validation (LOOCV) is K-fold cross-validation with $K = N$: In each split, each sample is held out in turn, and the predictor is trained on the remaining $N - 1$ samples and evaluated on the lone held-out sample.

The cross-validation risk is generally a biased estimate of the statistical risk, because the data sets used for training and validation are drawn from the same set T. However, since training and validation set are distinct within each split, the cross-validation risk is a better estimate of the generalization risk than the training risk.

Choosing a Good Value for K: Since the training set in each split is $(K-1)/K$ times the size of T, each predictor is somewhat worse than it would be if all of T were used, and the cross-validation risk estimate is therefore somewhat pessimistic (that is, biased upward). This bias decreases monotonically as the number

K of splits increases. It is smallest for LOOCV, since then $(K-1)/K = (N-1)/N \approx 1$. Nadeau and Bengio [4] show theoretically and empirically that the variance of a K-fold cross validation decreases with K as well. Thus, LOOCV ($K = N$) would seem to be the method of choice, as it leads to the smallest bias and variance. However, LOOCV is very expensive, because training is repeated N times. In addition, Nadeau and Bengio show that the improvements deriving from greater and greater values of K tend to saturate once K is around 10. As a result of their studies, they recommend $K = 15$ as a good compromise between accuracy and precision, on one hand, and computational cost on the other.