Chapter 10

Evaluation of Learning Algorithms

The majority of this book has focused on algorithm derivation and obtaining models, but we have yet to address how to evaluate these models. The maximum likelihood formalism for deriving learning algorithms provides some consistency results, where in the limit of samples we can discuss the convergence point of an estimator. In practice, however, we would like to evaluate the algorithms based on a finite sample. Imagine a setting where you learn two models, say using logistic regression with two different regularization parameters. Which of these two models is "better"? What does it even mean to say better? Do you want to say the model is better for this problem (data setting), or across multiple problems? Are we trying to compare algorithms or models obtained from a specific instance of an algorithm? How can we be confident that the measured performance accurately reflects the performance we expect to see on new data? These questions are largely separate from our previous questions of effectively optimizing a specified objective, and rather start to inquire about the properties of that objective and about empirical properties of learned models.

In this chapter, we provide theoretical and empirical tools to better evaluate the properties of learning algorithms. We begin with some basic finite-sample theoretical results, that relate the complexity of the model class to the number of samples required to obtain a reasonable estimate of expected error (generalization error). This section will also introduce the ideas of optimizing over a function class, and our goals for obtaining the best model in terms of generalization error. The area dealing with these types of theoretical characterizations is called *statistical learning theory*. We will discuss one result using *concentration inequalities* and *Rademacher complexity* to characterize model-class complexity; for further information, you could consider this tutorial on the topic [6].

Then, we will discuss how to compare algorithms empirically. In most real-world settings, we will choose between algorithms based on their performance on available data. We want this choice to be reflective of how well those algorithms will perform on new data. Towards this goal, we will discuss how to split data and how to use statistical significance tests to provide some level of confidence that one algorithm or model is better than another, under some specific criteria. We will rarely be able to make strong conclusions based on experiments, but we can build up some evidence on the algorithm properties.

These tools are arguably the most critical aspects of properly using machine learning algorithms in practice. One can learn a complex model, but without any understanding of how it is expected to perform in practice on new data, it is not viable to actual use these models. Whether an algorithm is used for scientific purposes or deployed in real systems, have an understanding of its properties both theoretically and empirically is key to obtain expected outcomes. This chapter only begins to scratch the surface of these tools, with the goal to pique your interest and direct you towards more material for learning about evaluation.

10.1 A brief introduction to generalization bounds

Our goal throughout this book has been to obtain a function, based on a set of examples, that predicts accurately: produces low expected error across the space of possible examples. We cannot, however, measure the expected error. Statistically, we know that with a sufficient sample, we can approximate an expectation. Here, we quantify this more carefully for learned functions.

Our goal more precisely is to select a function from a function class $\mathcal{H}$ to minimize a loss function $\ell : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ in expectation over all pairs $(\mathbf{x}, y)$

$$\min_{f \in \mathcal{H}} \mathbb{E}[\ell(f(\boldsymbol{X}), Y)].$$

For example, in linear regression, $\mathcal{H} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \text{ for any } \mathbf{w} \in \mathbb{R}^d\}$. This space of functions $\mathcal{H}$ represents all possible linear functions of inputs $\mathbf{x} \in \mathbb{R}^d$, to produce a scalar output. Our goal in linear regression was to minimize a proxy to the true expected error, i.e., the sample error: $\frac{1}{n}\sum_{i=1}^n \ell(f(\mathbf{x}_i), y_i)$. Now a natural question to ask is: does this sample error provide an accurate estimate of the true expected error? And what does it tell us about the true generalization performance, i.e., true expected error?

Let us start with a simple example, using linear regression. Assume a bounded function class $\mathcal{H}$, where $\mathcal{H} = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}, \text{ for any } \mathbf{w} \in \mathbb{R}^d \text{ such that } \|\mathbf{w}\|_2 \leq B_w\}$ for some finite scalar $B_w > 0$. Assume the input features come from a bounded space, such that for all $\mathbf{x}$, $\|\mathbf{x}\|_2 \leq B_x$ for some finite scalar $B_x > 0$, and further that the outputs $y \in [-B_y, B_y]$ for some $B_y > 0$. Assume we use loss $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$, which is (locally) Lipschitz continuous for our bounded region, with Lipschitz constant $c = B_y + B_x B_w$. This is because $|\hat{y}| \leq B_x B_w$ and

$$\left|\frac{d\ell(\hat{y}, y)}{d\hat{y}}\right| = |\hat{y} - y| \leq |\hat{y}| + |y| \leq B_y + B_x B_w.$$

Further, because $y \in [-B_y, B_y]$, we know the loss is bounded as

$$\ell(\hat{y}, y) = \tfrac{1}{2}(\hat{y} - y)^2 \leq \tfrac{1}{2}(B_y^2 + B_x^2 B_w^2).$$

For approximate error

$$\widehat{\mathrm{Err}}(f) = \frac{1}{n}\sum_{i=1}^n \ell(f(\mathbf{x}_i), y_i)$$

and true error

$$\mathrm{Err}(f) = \mathbb{E}[\ell(f(\boldsymbol{X}), Y)] = \int_{\mathcal{X} \times \mathcal{Y}} p(\mathbf{x}, y)\ell(f(\mathbf{x}), y) d\mathbf{x} dy$$

using Equation 10.2 below, we get that with probability $1 - \delta$, for $\delta \in (0, 1]$,

$$\mathrm{Err}(f) \leq \widehat{\mathrm{Err}}(f) + \frac{2cB_xB_w}{\sqrt{n}} + \tfrac{1}{2}(B_y^2 + B_x^2 B_w^2)\sqrt{\frac{\ln(1/\delta)}{2n}}. \tag{10.1}$$

With increasing samples n, the second two terms disappear and the sample error approaches the true expected error. This bound show the rate at which this discrepancy disappears. For a higher confidence—small δ making $\ln(1/\delta)$ larger—more samples are need

for the third term to be small. This third term is obtained using *concentration inequalities*, which enable us to state the rate at which a sample mean gets close to its expected value. For possibly large values of features or learned weights, the second term can be big and can again require the more samples. The second term reflects the properties of our function class: a simpler class, with small bounded weights, can have a more accurate estimate of the loss on a smaller number of samples. More generally, this complexity measure is called the *Rademacher complexity.*[1] For the linear functions above, with bounded ℓ_2 norms for $\mathbf{x}$, $\mathbf{w}$, the Rademacher complexity is bounded as $R_n(\mathcal{H}) \leq B_x B_y/\sqrt{n}$ (see [9, Equation 3]).

In the next few sections, we provide a generalization result for more general functions, as well as required background to determine that result.

10.1.1 Concentration inequalities

We will examine the use of concentration inequalities with one common example: Hoeffding's inequality. For the generalization bound below, a generalization is used, called McDiarmid's inequality.

For i.i.d. random variables $X_1, \ldots, X_n$, such that $0 \leq X_i \leq 1$, let $\bar{X} = \frac{1}{n}\sum_{i=1}^n X_i$ be the sample average. Then Hoeffding's inequality states that for any ϵ

$$\Pr(\bar{X} - \mathbb{E}[\bar{X}] \geq \epsilon) \leq \exp(-2n\epsilon^2).$$

We start by setting this probability value to δ, so that we can say with probability δ, $\Pr(\bar{X} - \mathbb{E}[\bar{X}] \geq \text{function}(\delta))$. We can solve for ϵ in terms of δ, to get

$$\delta = \exp(-2n\epsilon^2) \qquad \implies \epsilon = \pm\sqrt{\frac{\ln(1/\delta)}{2n}}.$$

We can either set ϵ to $\sqrt{\frac{\ln(1/\delta)}{2n}}$ or $-\sqrt{\frac{\ln(1/\delta)}{2n}}$, to bound $\bar{X}$ to be near $\mathbb{E}[\bar{X}]$ from both above and below. We get that with probability $1-\delta$, $\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \leq |\epsilon| = \sqrt{\frac{\ln(1/\delta)}{2n}}$.

This concentration inequality makes few assumptions about the random variables, and does not require any distributional assumptions. Consequently, the rate of convergence to the true mean is only $1/\sqrt{n}$. Faster rates can be obtained with more assumptions.

10.1.2 Complexity of a function class

Rademacher complexity of a function class characterizes the overfitting ability of functions, on a particular sample. Function class that are typically more complex are more likely to be able to fit random noise, and so have higher Rademacher complexity. The empirical Rademacher complexity, for a sample $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ —where typically we consider $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ — is defined as[2]

$$\hat{R}_n(\mathcal{H}) = \mathbb{E}\left[\max_{f\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^n \sigma_i f(\mathbf{x}_i)\right]$$

[1] If you have heard of VC dimension, we will discuss the connection between Rademacher and VC dimension below. They both play a role in identifying the complexity of a function class.

[2] Here we are being a bit loose and using maximum instead of supremum, to avoid burdening the reader with new terminology. We usually deal with function classes $\mathcal{H}$ where using the supremum is equivalent to using the maximum. The supremum is used when a set does not contain a maximal point (e.g., $[0, 1)$), where the supremum provides the closest upper bound (e.g., 1 for $[0, 1)$).

where the expectation is over i.i.d. random variables $\sigma_1, \ldots, \sigma_n$ chosen uniformly from $\{-1, 1\}$. This choice reflects how well the function class can correlate with this random noise. Consider for example if $f(\mathbf{x})$ predicts 1 or -1, as in binary classification. If there exists a function in the class of functions that can perfectly match the sign of the randomly sampled σ_i, then that function produces the highest value $\sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i)$. The empirical Rademacher complexity for a function class is high, if for any randomly sampled σ_i, there exists such a function within the function class (can be a different function for each $\sigma_1, \ldots, \sigma_n$). The Rademacher complexity is the expected empirical Rademacher complexity, over all possibly samples of n instances.

For function classes with high Rademacher complexity, error on the training set is unlikely to be reflective of the generalization error, until there is a sufficient number of samples. This is reflected in the generalization bound in Section 10.1.3.

Connection to VC dimension: The complexity of a function class can also be characterized by the VC dimension. The idea of VC dimension to characterize the number of points that can be separated (or shattered) by a function class. Simple functions have low VC dimension, because they are not complex enough to separate many points. More complex functions, that enable complex boundaries, have higher VC dimension. For example, for functions of the form $f((x_1, x_2)) = \text{sign}(x_1 w_1 + x_2 w_2 + w_0)$, the VC dimension is 3; more generally, for $\mathbf{x} \in \mathbb{R}^d$, the VC dimension is $d + 1$. VC dimension is a similar idea to Rademacher complexity, but it is restricted to binary classifiers. For this reason, we directly discuss the Rademacher complexity, which for binary classifiers can be bounded in terms of the VC dimension. By Sauer's Lemma, we can typically bound the Rademacher complexity of a hypothesis class by $\sqrt{\frac{2\text{VC-dimension} \ln n}{n}}$.

10.1.3 Generalization bounds

The generalization bound for a class of models can be obtained by combining the concentration inequalities to bound deviation from the mean for fewer samples, and using the Rademacher complexity to bound the difference between the sample error and true expected error across all functions in the function class. We additionally need to restrict the set of losses. We assume that the losses are Lipschitz with constant c, meaning that they do not change too quickly in a region, with c indicating the rate of change. Further, we also assume that the loss is bounded by b, i.e., attains values in $[-b, b]$. As above, if $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ is i.i.d., then with probability $1 - \delta$, for every $f \in \mathcal{H}$,

$$\mathbb{E}[\ell(f(\boldsymbol{X}), Y)] \leq \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i) + 2cR_n(\mathcal{H}) + b\sqrt{\frac{\ln(1/\delta)}{2n}} \tag{10.2}$$

For a more precise theorem statement and a proof, see [2, Theorem 7] and [9, Theorem 1].

10.2 Comparison of Learning Algorithms

To empirically evaluate algorithms, we can consider a setting with one or more algorithms on one or more datasets. Depending on the setting, different evaluations will be employed. For a nice overview of evaluation for machine learning algorithms, see [8].

For now, let us start with a simple case, where we compare two algorithms and use the binomial test. Suppose we have a set of learning problems $\mathcal{D}_1, \mathcal{D}_2, \ldots, D_m$ and wish to compare learning algorithms a_1 and a_2. We can carry out such a comparison using a counting test as follows: for each data set both algorithms are evaluated in terms of the chosen performance measure and the algorithm with a higher performance accuracy is awarded a win, while the other one is given a loss (in case of exactly the same performance, we can provide a win/loss randomly).

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$		$\mathcal{D}_{m-1}$	$\mathcal{D}_m$
a_1	1	0	1	1	$\cdots$	0	1
a_2	0	1	0	0		1	0

Table 10.1: A counting test where learning algorithms a_1 and a_2 are compared on a set of m independent data sets. An algorithm with a better performance on a particular data set collects a win (1), whereas the other algorithm collects a loss (0).

We are now interested in providing statistical evidence that say algorithm a_1 is better than algorithm a_2. Suppose a_1 has k wins out of m and algorithm a_2 has $m - k$ wins, as shown in Table 10.1. We would like to evaluate the null hypothesis H_0 that algorithms a_1 and a_2 have the same performance by providing an alternative hypothesis H_1 that algorithm a_1 is better than a_2. In short,

$$H_0\text{: quality}(a_1) = \text{quality}(a_2)$$
$$H_1\text{: quality}(a_1) > \text{quality}(a_2)$$

If the null hypothesis is true, the win/loss on each data set will be equally likely and determined by minor variation. Therefore, the probability of a win on any data set will be roughly equal to $p = 1/2$. We can now express the probability that algorithm a_1 collected k wins or more under the null hypothesis using binomial distribution

$$P = \sum_{i=k}^{m} \binom{m}{i} p^i (1-p)^{m-i}$$

and refer to it as the P-value. This value is the probability of k wins, plus the probability of $k+1$ wins, up to the probability of m wins, under the null hypothesis. A typical approach in these cases is to establish a significance value, say, $\alpha = 0.05$ and reject the null hypothesis if $P \leq \alpha$. If the P-value is greater than α we say that there is insufficient evidence for rejecting H_0. For sufficiently low P-values, we may conclude that there is sufficient evidence that algorithm a_1 is better than algorithm a_2.

The choice of the significance threshold α is somewhat arbitrary. Typically, 5% is a reasonable value, but lower values indicate that the particular situation of k wins out of m was so unlikely, that we can consider the evidence for rejecting H_0 very strong. Being able to reject the null hypothesis provides some confidence that the result did not occur by chance.

More generally, we can consider other statistical significance tests based on the distributions of the performance measures. In the above example, a binomial distribution was appropriate. If instead we considered the actual errors on the datasets, then we have pairs

of real values. In this case, a common choice is the paired t-test, if both errors appear to be distributed normally and if they have similar variance. The paired t-test takes in the sampled differences between the algorithms (line 3 in Table 10.2), $d_1, \ldots, d_m$. Because again our null hypothesis is that the algorithms perform equally, under the null hypothesis the mean of these differences is 0. If the differences are normally distributed, then for the sample average $\bar{d} = \frac{1}{m}\sum_{i=1}^{m} d_i$ and sample standard deviation $S_d = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(d_i - \bar{d})^2}$, the random variable $t = \frac{\bar{d}-0}{S_d/\sqrt{m}}$ is distributed according to the Student's t-distribution. The Student's t-distribution is approximately like a normal distribution, with a degrees-of-freedom parameter $m-1$ that makes the distribution look more like a normal distribution as m becomes larger.

We can now ask about the probability of this random variable T, relative to the computed statistic. If we only care about knowing if algorithm 1 is better than algorithm 2, we conduct a one-tailed test. If the probability that T is larger than t, i.e., $p = \Pr(T > t)$, is small, then we obtain some evidence that algorithm 1 is better than algorithm 2. To test if algorithm 1 is better than algorithm 2, we can swap the order of the difference; if $p = \Pr(T > -t)$ is small, then we obtain some evidence that algorithm 2 is better than algorithm 1. These are both one-tailed tests, reflecting the probabilities at one end of the tails of the distribution. A two-tailed test instead asks if the two algorithms are different; in this case, one would use $p = \Pr(T > |t|)$.

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$		$\mathcal{D}_{m-1}$	$\mathcal{D}_m$
a_1	0.11	0.08	0.15	0.12	$\cdots$	0.07	0.09
a_2	0.10	0.09	0.11	0.12	$\cdots$	0.10	0.09
d	0.01	-0.01	0.04	0.0	$\cdots$	-0.03	0.0

Table 10.2: A table of errors for two learning algorithms a_1 and a_2 are compared on a set of m independent data sets. The last row contains the differences, which are used for the paired t-test.

If the paired samples are not normally distributed, other tests are more suitable. Further, there are some tests that do not make distributional assumptions, and rather are non-parametric. For a summary of which test to use in different settings, see [8, Section 6.3]

10.3 Obtaining samples of error

A key step in comparing algorithms is to obtain valid measures of performance for the comparison. So far, we have assumed that these are given. One approach to obtain unbiased samples of the error is to keep a hold-out test set. Imagine m samples are set in reserve, on which the algorithms are not trained and which we cannot look at until we are ready to evaluate. We can train two models on the training set, and then obtain m paired samples of error. We can then use the paired t-test to make claims about if the two models are statistically significantly different for the problem.

However, there are two key disadvantages to using a hold-out test set. First, usually we want to use all the data for training. Unless there is more data than can be used, keeping a hold-out test set is typically not practical. Even in this age of huge datasets,

we still typically want to learn on as much (quality) data as possible. Second, once this hold-out test set has been used for evaluation, we cannot use it again because it will not provide an unbiased estimate of the expected error. For example, after getting performance of your models on that test set, one could go back and adjust meta-parameters such as the regularization parameters. However, once you have done this, the test-set has influenced the learned models and is likely to produce an optimistic estimate of performance on new data. Therefore, this hold-out test-set can only be used once.

An alternative approach to obtain estimates of error is to use resampling techniques from the whole dataset. Two common resampling techniques are k-fold cross-validation and bootstrap resampling. In the first, the data is partitioned into k disjoint sets (folds). The model is trained on $k-1$ of the folds, and tested on the other fold; this is repeated k times where each fold acts as the test fold. This approach simulates the common learning setting where the training and test sets are disjoint. The resulting k performance estimates are mostly independent, with some dependency introduced due to dependencies between the training sets across the k runs. There is some additional bias introduced from the fact that we do not run the model on the entire training set, but rather get an estimate of the error for the algorithm trained on $n-(n/k)$. For any final models that will be put into production after performing these evaluations, we will likely train on the entire set of n instances.

The bootstrap resample treats the data uses the idea behind bootstrapping: the data constitutes a reasonable model of the data. By sampling from the data, it is like sampling from the distribution that generated the data. To generate training/test splits, the data is sampled with replacement to create the training set, and the remaining unused samples used for test. If k resamples are obtained, we again get k performance measures and can obtain a sample average of performance across different splits and use statistical significance test.

To better understand the properties of these two approaches, see the thorough and accessible explanation in [7, Chapter 5].

10.4 Performance measures for Classification Models

In classification, there are a variety of performance measures to reflect the relative importance of incorrect predictions for either class. For example, it can be more detrimental to predict a patient is not sick if they are actually sick (False Negative), resulting in a decision not to run further diagnostics and so causing serious complications from not treating the illness. When training and evaluating classification algorithms, these preferences need to be encoded. Table 10.3 summarizes some of the terminology for discussing performance of classification models.

Name	Symbol	Definition
Classification error	$error$	$error = \frac{fp+fn}{tp+fp+tn+fn}$
Classification accuracy	$accuracy$	$accuracy = 1 - error$
True positive rate	tpr	$tpr = \frac{tp}{tp+fn}$
False negative rate	fnr	$fnr = \frac{fn}{tp+fn}$
True negative rate	tnr	$tnr = \frac{tn}{tn+fp}$
False positive rate	fpr	$fpr = \frac{fp}{tn+fp}$
Precision	pr	$pr = \frac{tp}{tp+fp}$
Recall	rc	$rc = \frac{tp}{tp+fn}$

Table 10.3: Some classification measures.