**Chapter 5**

**MACHINE LEARNING BASICS**

* 1. **Learning Algorithms**

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

* + 1. **The Task, T**

Many kinds of tasks can be solved with machine learning. Some of the most common machine learning tasks include the following:

Classification:

* In this type of task, the computer program is asked to specify which of k categories some input belongs to.
* To solve this task, the learning algorithm is usually asked to produce a function f: Rn → {1, . . ., k}. where f outputs a probability distribution over classes.
* An example of a classification task is object recognition. Object recognition is the same basic technology that allows computers to recognize faces

Classification with missing inputs:

Regression:

* In this type of task, the computer program is asked to predict a numerical value given some input.
* To solve this task, the learning algorithm is asked to output a function f: Rn → R.
* An example is prediction of future prices of securities.

Transcription:

* In this type of task, the machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe it into discrete, textual form.
* Example is speech recognition, where the computer program is provided an audio waveform and emits a sequence of characters or word ID codes describing the words that were spoken in the audio recording.

Machine translation:

* + 1. **The Performance Measure, P**

To evaluate the abilities of a machine learning algorithm, we must design a quantitative measure of its performance.

For tasks such as classification, classification with missing inputs, and transcription, we often measure the *accuracy* of the model. Accuracy is just the proportion of examples for which the model produces the correct output. We can also obtain equivalent information by measuring the *error rate*, the proportion of examples for which the model produces an incorrect output. We often refer to the error rate as the expected 0-1 loss. The 0-1 loss on a example is 0 if it is correctly classified and 1 if it is not. For tasks such as density estimation, it does not make sense to measure accuracy, error rate, or any other kind of 0-1 loss. Instead, we must use a different performance metric that gives the model a continuous-valued score for each example. The most common approach is to report the average log-probability the model assigns to some examples.

* + 1. **The Experience, E**

Machine learning algorithms can be broadly categorized as *unsupervised* or *supervised* by what kind of experience they can have during the learningprocess.

Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset. In the context of deep learning, we usually want to learn the entire probability distribution that generated a dataset, whether explicitly as in density estimation or implicitly for tasks like synthesis or denoising.

*Supervised learning algorithms* experience a dataset containing features, but each example is also associated with a *label* or *target*.

Roughly speaking, unsupervised learning involves observing several examples of a random vector x, and attempting to implicitly or explicitly learn the probability distribution p(x), or some interesting properties of that distribution, while supervised learning involves observing several examples of a random vector x and an associated value or vector y, and learning to predict y from x, usually by estimating p(y | x).

Unsupervised learning and supervised learning are not formally defined terms. The lines between them are often blurred. Many machine learning technologies can be used to perform both tasks. For example, the chain rule of probability states that for a vector x ∈ Rn, the joint distribution can be decomposed as

P(x) =

This decomposition means that we can solve the ostensibly unsupervised problem of modeling p(x) by splitting it into n supervised learning problems. Alternatively, we can solve the supervised learning problem of learning p(y | x) by using traditional unsupervised learning technologies to learn the joint distribution p(x, y) and inferring

P(y | x) =

Other variants of the learning paradigm are possible. For example, in semisupervised learning, some examples include a supervision target but others do not. In multi-instance learning, a complete collection of examples is labeled as containing or not containing an example of a class, but the individual members of the collection are not labeled.

Some machine learning algorithms do not just experience a fixed dataset. For example, *reinforcement learning* algorithms interact with an environment, so there is a feedback loop between the learning system and its experiences.

One common way of describing a dataset is with a *design matrix*. A design matrix is a matrix containing a different example in each row. Each column of the matrix corresponds to a different feature.

Of course, to describe a dataset as a design matrix, it must be possible to describe each example as a vector, and each of these vectors must be the same size. This is not always possible.

In cases like these, rather than describing the dataset as a matrix with m rows, we will describe it as a set containing m elements: {x(1), x(2), . . . , x(m)}. This notation does not imply that any two example vectors x(i) and x(j) have the same size.

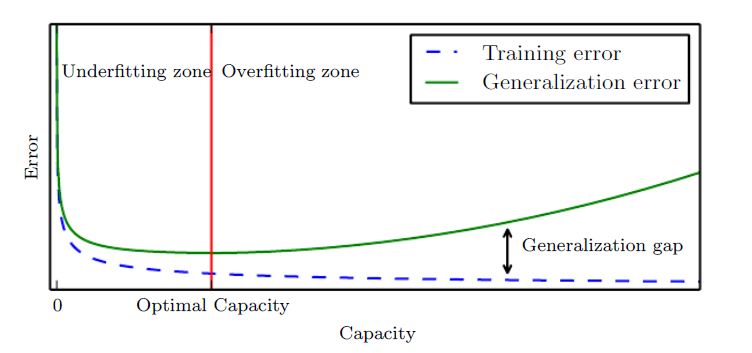
* + 1. **Example: Linear Regression**
  1. **Capacity, Overfitting and Underfitting**
* The ability to perform well on previously unobserved inputs is called **generalization**.
* The train and test data are generated by a probability distribution over datasets called the datagenerating process. We typically make a set of assumptions known collectively as the **i.i.d. assumptions** (independent and identically distributed).

These assumptions are that the examples in each dataset are independentfrom each other, and that the train set and test set are identically distributed, drawn from the same probability distribution as each other.

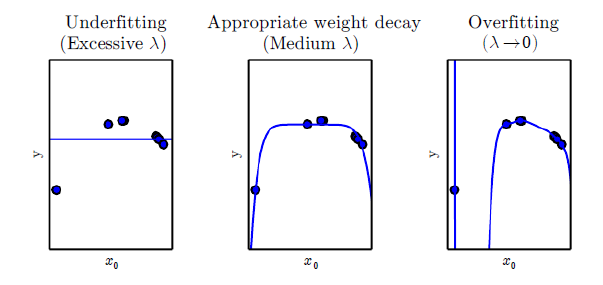
* We call that shared underlying distribution the data generating distribution, denoted pdata. This probabilistic framework and the i.i.d. assumptions allow us to mathematically study the relationship between training error and test error.
* These two factors correspond to the two central challenges in machine learning: **underfitting**and **overfitting**.
* Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set.
* Overfitting occurs when the gap between the training error and test error is too large.
* We can control whether a model is more likely to overfit or underfit by altering its **capacity**. Informally, a model’s capacity is its ability to fit a wide variety of functions.
* Models with low capacity may struggle to fit the training set.
* Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.
* One way to control the capacity of a learning algorithm is by choosing its **hypothesis space**, the set of functions that the learning algorithm can select as being the solution. For example, the linear regression algorithm has the set of all linear functions of its input as its hypothesis space. We can generalize linear regression to include polynomials, rather than just linear functions, in its hypothesis space. Doing so increases the model’s capacity. By introducing x2 as another feature provided to the linear regression model, we can learn a model that is quadratic as a function of x:

yˆ = b + w1x + w2x2

* Models with insufficient capacity are unable to solve complex tasks. Models with high capacity can solve complex tasks, but when their capacity is higher than needed to solve the present task they may overfit.
* The model specifies which family of functions the learning algorithm can choose from when varying the parameters in order to reduce a training objective. This is called the **representational capacity**of the model. In many cases, finding the best function within this family is a very difficult optimization problem. In practice, the learning algorithm does not actually find the best function, but merely one that significantly reduces the training error. These additional limitations, such as the imperfection of the optimization algorithm, mean that the learning algorithm’s effective capacitymay be less than the representational capacity of the model family.
* The most important results in statistical learning theory show that the discrepancy between training error and generalization error is bounded from above by a quantity that grows as the model capacity grows but shrinks as the number of training examples increases.
* The problem of determining the capacity of a deep learning model is especially difficult because the effective capacity is limited by the capabilities of the optimization algorithm, and we have little theoretical understanding of the very general non-convex optimization problems involved in deep learning.
* We must remember that while simpler functions are more likely to generalize (to have a small gap between training and test error) we must still choose a sufficiently complex hypothesis to achieve low training error. Typically, training error decreases until it asymptotes to the minimum possible error value as model capacity increases (assuming the error measure has a minimum value). Typically, generalization error has a U-shaped curve as a function of model capacity.

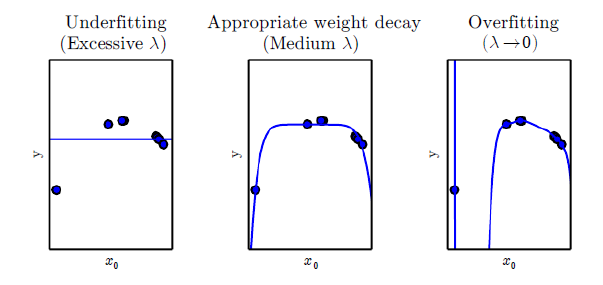


* To reach the most extreme case of arbitrarily high capacity, we introduce the concept of non-parametricmodels.
* **Parametric models** learn a function described by a parameter vector whose size is finite and fixed before any data is observed. **Non-parametric models** have no such limitation.
* Sometimes, non-parametric models are just theoretical abstractions (such as an algorithm that searches over all possible probability distributions) that cannot be implemented in practice. However, we can also design practical non-parametric models by making their complexity a function of the training set size. One example of such an algorithm is **nearest neighbor regression**. Unlike linear regression, which has a fixed-length vector of weights, the nearest neighbor regression model simply stores the X and y from the training set. When asked to classify a test point x, the model looks up the nearest entry in the training set and returns the associated regression target.
* Finally, we can also create a non-parametric learning algorithm by wrapping a parametric learning algorithm inside another algorithm that increases the number of parameters as needed. For example, we could imagine an outer loop of learning that changes the degree of the polynomial learned by linear regression on top of a polynomial expansion of the input.
* The ideal model is an oracle that simply knows the true probability distribution that generates the data. Even such a model will still incur some error on many problems, because there may still be some noise in the distribution. In the case of supervised learning, the mapping from x to y may be inherently stochastic, or y may be a deterministic function that involves other variables besides those included in x. The error incurred by an oracle making predictions from the true distribution p (x, y) is called the **Bayes error**.
* Training and generalization error vary as the size of the training set varies. Expected generalization error can never increase as the number of training examples increases. For non-parametric models, more data yields better generalization until the best possible error is achieved. Any fixed parametric model with less than optimal capacity will asymptote to an error value that exceeds the Bayes error.
  + 1. **The No Free Lunch Theorem**
* The no free lunch theorem for machine learning (Wolpert, 1996) states that, averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points. In other words, in some sense, no machine learning algorithm is universally any better than any other.
* Fortunately, these results hold only when we average over all possible data generating distributions.
* This means that the goal of machine learning research is not to seek a universal learning algorithm or the absolute best learning algorithm. Instead, our goal is to understand what kinds of distributions are relevant to the “real world” that an AI agent experiences, and what kinds of machine learning algorithms perform well on data drawn from the kinds of data generating distributions we care about.
  + 1. **Regularization**
* The behavior of our algorithm is strongly affected not just by how large we make the set of functions allowed in its hypothesis space, but by the specific identity of those functions.
* We can also give a learning algorithm a preference for one solution in its hypothesis space to another. This means that both functions are eligible, but one is preferred. The unpreferred solution be chosen only if it fits the training data significantly better than the preferred solution.
* For example, we can modify the training criterion for linear regression to include **weight decay**. To perform linear regression with weight decay, we minimize a sum comprising both the mean squared error on the training and a criterion J(w) that expresses a preference for the weights to have smaller squared L2 norm. Specifically,
* where λ is a value chosen ahead of time that controls the strength of our preference for smaller weights. When λ = 0, we impose no preference, and larger λ forces the weights to become smaller. Minimizing J (w ) results in a choice of weights that make a tradeoff between fitting the training data and being small. This gives us solutions that have a smaller slope, or put weight on fewer of the features. As an example of how we can control a model’s tendency to overfit or underfit via weight decay, we can train a high-degree polynomial regression model with different values of λ. See below figure for results.



**FIGURE**: *We vary the amount of weight decay to prevent these high-degree models from overfitting. (Left) With very large λ, we can force the model to learn a function with no slope at all. This underfits because it can only represent a constant function. (Center) With a medium value of λ, the learning algorithm recovers a curve with the right general shape. Even though the model is capable of representing functions with much more complicated shape, weight decay has encouraged it to use a simpler function described by smaller coefficients. (Right) With weight decay approaching zero (i.e., using the Moore-Penrose pseudoinverse to solve the underdetermined problem with minimal regularization), the degree-9 polynomial overfits significantly*

* More generally, we can regularize a model that learns a function f(x; θ) by adding a penalty called a **regularizer**to the cost function. In the case of weight decay, the regularizer is Ω(w) = wTw.
* **Regularization** is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
* The no free lunch theorem has made it clear that there is no best machine learning algorithm, and, in particular, no best form of regularization. Instead we must choose a form of regularization that is well-suited to the particular task we want to solve.
  1. **Hyperparameters and Validation Sets**
* Most machine learning algorithms have several settings that we can use to control the behavior of the learning algorithm. These settings are called **hyperparameters**. The λ value used to control the strength of weight decay is an example of a hyperparameter.
* It is important that the test examples are not used in any way to make choices about the model, including its hyperparameters. For this reason, no example from the test set can be used in the validation set. Therefore, we always construct the validation set from the trainingdata.
* Data used to guide the selection of hyperparameters is called the **validation set**. Typically, one uses about 80% of the training data for training and 20% for validation.



**Algorithm:** **The k-fold cross-validation algorithm**.

It can be used to estimate generalization error of a learning algorithm A when the given dataset D is too small for a simple train/test or train/valid split to yield accurate estimation of generalization error, because the mean of a loss L on a small test set may have too high variance.

The algorithm returns the vector of errors **e** for each example in D, whose mean is the estimated generalization error.

While these confidence intervals are not well-justified after the use of cross-validation, it is still common practice to use them to declare that algorithm A is better than algorithm B only if the confidence interval of the error of algorithm A lies below and does not intersect the confidence interval of algorithm B.

**Define** KFoldXV(D,A,L, k):

**Require:** D, the given dataset, with elements z(i)

**Require:** A, the learning algorithm, seen as a function that takes a dataset as

input and outputs a learned function

**Require:** L, the loss function, seen as a function from a learned function f and

an example z(i) ∈ D to a scalar ∈ R

**Require:** k, the number of folds

Split D into k mutually exclusive subsets Di , whose union is D.

**for** i from 1 to k **do**

fi = A(D \ Di)

**for** z(j) in Di **do**

ej = L(fi, z(j))

**end for**

**end for**

**Return** e

* 1. **Estimators, Bias and Variance**
     1. **Point Estimation**