# CHAPTER 1

# Introduction

The main focus of machine learning (ML) is *making decisions or predictions based on data*. There are a number of other fields with significant overlap in technique, but difference in focus: in economics and psychology, the goal is to discover underlying causal processes and in statistics it is to find a model that fits a data set well. In those fields, the end product is a model. In machine learning, we often fit models, but as a means to the end of making good predictions or decisions.

This description paraphrased from a post on 9/4/12 at `andrewgelman.com`

As ML methods have improved in their capability and scope, ML has become arguably the best way–measured in terms of speed, human engineering time, and robustness–to approach many applications. Great examples are face detection, speech recognition, and many kinds of language-processing tasks. Almost any application that involves understanding data or signals that come from the real world can be nicely addressed using machine learning.

One crucial aspect of machine learning approaches to solving problems is that human engineering plays an important role. A human still has to *frame* the problem: acquire and organize data, design a space of possible solutions, select a learning algorithm and its parameters, apply the algorithm to the data, validate the resulting solution to decide whether it's good enough to use, try to understand the impact on the people who will be affected by its deployment, etc. These steps are of great importance.

and often undervalued

The conceptual basis of learning from data is the *problem of induction*: Why do we think that previously seen data will help us predict the future? This is a serious long standing philosophical problem. We will operationalize it by making assumptions, such as that all training data are so-called i.i.d.(independent and identically distributed), and that queries will be drawn from the same distribution as the training data, or that the answer comes from a set of possible answers known in advance.

This means that the elements in the set are related in the sense that they all come from the same underlying probability distribution, but not in any other ways.

In general, we need to solve these two problems:

- **estimation:** When we have data that are noisy reflections of some underlying quantity of interest, we have to aggregate the data and make estimates or predictions about the quantity. How do we deal with the fact that, for example, the same treatment may end up with different results on different trials? How can we predict how well an estimate may compare to future results?

- **generalization:** How can we predict results of a situation or experiment that we have never encountered before in our data set?

We can describe problems and their solutions using six characteristics, three of which characterize the problem and three of which characterize the solution:

1. **Problem class:** What is the nature of the training data and what kinds of queries will be made at testing time?
2. **Assumptions:** What do we know about the source of the data or the form of the solution?
3. **Evaluation criteria:** What is the goal of the prediction or estimation system? How will the answers to individual queries be evaluated? How will the overall performance of the system be measured?
4. **Model type:** Will an intermediate model of the world be made? What aspects of the data will be modeled in different variables/parameters? How will the model be used to make predictions?
5. **Model class:** What particular class of models will be used? What criterion will we use to pick a particular model from the model class?
6. **Algorithm:** What computational process will be used to fit the model to the data and/or to make predictions?

Without making some assumptions about the nature of the process generating the data, we cannot perform generalization. In the following sections, we elaborate on these ideas.

Don't feel you have to memorize all these kinds of learning, etc. We just want you to have a very high-level view of (part of) the breadth of the field.

## 1.1 Problem class

There are many different *problem classes* in machine learning. They vary according to what kind of data is provided and what kind of conclusions are to be drawn from it. Five standard problem classes are described below, to establish some notation and terminology.

In this course, we will focus on classification and regression (two examples of supervised learning), and we will touch on reinforcement learning, sequence learning, and clustering.

### 1.1.1 Supervised learning

The idea of *supervised* learning is that the learning system is given inputs and told which specific outputs should be associated with them. We divide up supervised learning based on whether the outputs are drawn from a small finite set (classification) or a large finite ordered set or continuous set (regression).

#### 1.1.1.1 Regression

For a regression problem, the training data $\mathcal{D}_n$ is in the form of a set of $n$ pairs:

$$\mathcal{D}_n = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\},$$

where $x^{(i)}$ represents an input, most typically a $d$-dimensional vector of real and/or discrete values, and $y^{(i)}$ is the output to be predicted, in this case a real-number. The $y$ values are sometimes called *target values*.

Many textbooks use $x_i$ and $t_i$ instead of $x^{(i)}$ and $y^{(i)}$. We find that notation somewhat difficult to manage when $x^{(i)}$ is itself a vector and we need to talk about its elements. The notation we are using is standard in some other parts of the ML literature.

The goal in a regression problem is ultimately, given a new input value $x^{(n+1)}$, to predict the value of $y^{(n+1)}$. Regression problems are a kind of *supervised learning*, because the desired output $y^{(i)}$ is specified for each of the training examples $x^{(i)}$.

#### 1.1.1.2 Classification

A classification problem is like regression, except that the values that $y^{(i)}$ can take do not have an order. The classification problem is *binary* or *two-class* if $y^{(i)}$ (also known as the *class*) is drawn from a set of two possible values; otherwise, it is called *multi-class*.

### 1.1.2 Unsupervised learning

*Unsupervised* learning doesn't involve learning a function from inputs to outputs based on a set of input-output pairs. Instead, one is given a data set and generally expected to find some patterns or structure inherent in it.

#### 1.1.2.1 Clustering

Given samples $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$, the goal is to find a partitioning (or "clustering") of the samples that groups together similar samples. There are many different objectives, depending on the definition of the similarity between samples and exactly what criterion is to be used (e.g., minimize the average distance between elements inside a cluster and maximize the average distance between elements across clusters). Other methods perform a "soft" clustering, in which samples may be assigned 0.9 membership in one cluster and 0.1 in another. Clustering is sometimes used as a step in the so-called density estimation (described below), and sometimes to find useful structure or influential features in data.

#### 1.1.2.2 Density estimation

Given samples $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$ drawn i.i.d. from some distribution $\Pr(X)$, the goal is to predict the probability $\Pr(x^{(n+1)})$ of an element drawn from the same distribution. Density estimation sometimes plays a role as a "subroutine" in the overall learning method for supervised learning, as well.

The capital X is a typical practice to emphasize this is a so-called random variable. Small letters are often used in probability too; those are typically reserved to denote the realized values of random variables. It might help to concretely think of coin-tosses; there, the toss outcome is a random variable and it may be realized as a "head". This paragraph actually talks about both a random variable and a realization of it, can you spot that from the notation and do you feel the difference?

#### 1.1.2.3 Dimensionality reduction

Given samples $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^D$, the problem is to re-represent them as points in a d-dimensional space, where $d < D$. The goal is typically to retain information in the data set that will, e.g., allow elements of one class to be distinguished from another.

Dimensionality reduction is a standard technique that is particularly useful for visualizing or understanding high-dimensional data. If the goal is ultimately to perform regression or classification on the data after the dimensionality is reduced, it is usually best to articulate an objective for the overall prediction problem rather than to first do dimensionality reduction without knowing which dimensions will be important for the prediction task.

### 1.1.3 Sequence learning

In sequence learning, the goal is to learn a mapping from *input sequences* $x_0, \dots, x_n$ to *output sequences* $y_1, \dots, y_m$. The mapping is typically represented as a *state machine*, with one function $f_s$ used to compute the next hidden internal state given the input, and another function $f_o$ used to compute the output given the current hidden state.

It is supervised in the sense that we are told what output sequence to generate for which input sequence, but the internal functions have to be learned by some method other than direct supervision, because we don't know what the hidden state sequence is.

### 1.1.4 Reinforcement learning

In reinforcement learning, the goal is to learn a mapping from input values (typically assumed to be states of an agent or system; for now, think e.g. the velocity of a moving car) to output values (typically we want control actions; for now, think e.g. if to accelerate or hit the brake). However, we need to learn the mapping without a direct supervision signal to specify which output values are best for a particular input; instead, the learning problem is framed as an agent interacting with an environment, in the following setting:

- The agent observes the current state $s_t$ .
- It selects an action $a_t$.
- It receives a reward, $r_t$, which typically depends on $s_t$ and possibly $a_t$.
- The environment transitions probabilistically to a new state, $s_{t+1}$, with a distribution that depends only on $s_t$ and $a_t$.
- The agent observes the current state, $s_{t+1}$.
- ...

Note it's standard practice in reinforcement learning to use $s$ and $a$ instead of $x$ and $y$ to denote the machine learning model's input and output. The subscript $t$ denotes the timestep, and captures the sequential nature of the problem.

The goal is to find a policy $\pi$, mapping $s$ to $a$, (that is, states to actions) such that some long-term sum or average of rewards $r$ is maximized.

This setting is very different from either supervised learning or unsupervised learning, because the agent's action choices affect both its reward and its ability to observe the environment. It requires careful consideration of the long-term effects of actions, as well as all of the other issues that pertain to supervised learning.

### 1.1.5 Other settings

There are many other problem settings. Here are a few.

In *semi-supervised* learning, we have a supervised-learning training set, but there may be an additional set of $x^{(i)}$ values with no known $y^{(i)}$. These values can still be used to improve learning performance (if they are drawn from $\Pr(X)$ that is the marginal of $\Pr(X, Y)$ that governs the rest of the data set).

In *active* learning, it is assumed to be expensive to acquire a label $y^{(i)}$ (imagine asking a human to read an x-ray image), so the learning algorithm can sequentially ask for particular inputs $x^{(i)}$ to be labeled, and must carefully select queries in order to learn as effectively as possible while minimizing the cost of labeling.

In *transfer* learning (also called *meta-learning*), there are multiple tasks, with data drawn from different, but related, distributions. The goal is for experience with previous tasks to apply to learning a current task in a way that requires decreased experience with the new task.

## 1.2 Assumptions

The kinds of assumptions that we can make about the data source or the solution include:

- The data are independent and identically distributed (i.i.d.).
- The data are generated by a Markov chain (i.e. outputs only depend only on the current *state*, with no additional *memory*).
- The process generating the data might be adversarial.

- The "true" model that is generating the data can be perfectly described by one of some particular set of hypotheses.

The effect of an assumption is often to reduce the "size" or "expressiveness" of the space of possible hypotheses and therefore reduce the amount of data required to reliably identify an appropriate hypothesis.

## 1.3 Evaluation criteria

Once we have specified a problem class, we need to say what makes an output or the answer to a query good, given the training data. We specify evaluation criteria at two levels: how an individual prediction is scored, and how the overall behavior of the prediction or estimation system is scored.

The quality of predictions from a learned model is often expressed in terms of a *loss function*. A loss function $\mathcal{L}(g, a)$ tells you how much you will be penalized for making a guess $g$ when the answer is actually $a$. There are many possible loss functions. Here are some frequently used examples:

- **0-1 Loss** applies to predictions drawn from finite domains.

$$\mathcal{L}(g, a) = \begin{cases} 0 & \text{if } g = a \\ 1 & \text{otherwise} \end{cases}$$

If the actual values are drawn from a continuous distribution, the probability they would ever be equal to some predicted $g$ is 0 (except for some weird cases).

- **Squared loss**

$$\mathcal{L}(g, a) = (g - a)^2$$

- **Absolute loss**

$$\mathcal{L}(g, a) = |g - a|$$

- **Asymmetric loss** Consider a situation in which you are trying to predict whether someone is having a heart attack. It might be much worse to predict "no" when the answer is really "yes", than the other way around.

$$\mathcal{L}(g, a) = \begin{cases} 1 & \text{if } g = 1 \text{ and } a = 0 \\ 10 & \text{if } g = 0 \text{ and } a = 1 \\ 0 & \text{otherwise} \end{cases}$$

Any given prediction rule will usually be evaluated based on multiple predictions and the loss of each one. At this level, we might be interested in:

- Minimizing expected loss over all the predictions (also known as *risk*)
- Minimizing maximum loss: the loss of the worst prediction
- Minimizing or bounding regret: how much worse this predictor performs than the best one drawn from some class
- Characterizing asymptotic behavior: how well the predictor will perform in the limit of infinite training data
- Finding algorithms that are probably approximately correct: they probably generate a hypothesis that is right most of the time.

There is a theory of rational agency that argues that you should always select the action that *minimizes the expected loss*. This strategy will, for example, make you the most money in the long run, in a gambling setting. As mentioned above, expected loss is also sometimes called risk in ML literature, but that term means other things in economics or other parts of decision theory, so be careful...it's risky to use it. We will, most of the time, concentrate on this criterion.

Of course, there are other models for action selection and it's clear that people do not always (or maybe even often) select actions that follow this rule.

## 1.4 Model type

Recall that the goal of a ML system is typically to estimate or generalize, based on data provided. Below, we examine the role of model-making in machine learning.

### 1.4.1 Non-parametric models

In some simple cases, in response to queries, we can generate predictions directly from the training data, without the construction of any intermediate model, or more precisely, without the learning of any parameters.

For example, in regression or classification, we might generate an answer to a new query by averaging answers to recent queries, as in the *nearest neighbor* method.

### 1.4.2 Parametric models

This two-step process is more typical:

1. "Fit" a model (with some a-prior chosen parameterization) to the training data
2. Use the model directly to make predictions

In the *parametric models* setting of regression or classification, the model will be some hypothesis or prediction rule $y = h(x; \Theta)$ for some functional form $h$. The term *hypothesis* has its roots in statistical learning and the scientific method, where models or hypotheses about the world are tested against real data, and refined with more evidence, observations, or insights. Note that the parameters themselves are only part of the assumptions that we're making about the world. The model itself is a hypothesis that will be refined with more evidence.

The idea is that $\Theta$ is a set of one or more parameter values that will be determined by fitting the model to the training data and then be held fixed during testing.

Given a new $x^{(n+1)}$, we would then make the prediction $h(x^{(n+1)}; \Theta)$.

We write $f(a; b)$ to describe a function that is usually applied to a single argument $a$, but is a member of a parametric family of functions, with the particular function determined by parameter value $b$.

The fitting process is often articulated as an optimization problem: Find a value of $\Theta$ that minimizes some criterion involving $\Theta$ and the data. An optimal strategy, if we knew the actual underlying distribution on our data, $\Pr(X, Y)$ would be to predict the value of $y$ that minimizes the *expected loss*, which is also known as the *test error*. If we don't have that actual underlying distribution, or even an estimate of it, we can take the approach of minimizing the *training error*: that is, finding the prediction rule $h$ that minimizes the average loss on our training data set. So, we would seek $\Theta$ that minimizes

This notation describes a so-called "joint distribution"; roughly, as the name suggests, it captures how both random variables X and Y "contribute" to the chance of something happening.

$$\mathcal{E}_n(h; \Theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(h(x^{(i)}; \Theta), y^{(i)}) \enspace ,$$

where the loss function $\mathcal{L}(g, a)$ measures how bad it would be to make a guess of $g$ when the actual value is $a$.

We will find that minimizing training error alone is often not a good choice: it is possible to emphasize fitting the current data too strongly and end up with a hypothesis that does not generalize well when presented with new $x$ values.

## 1.5 Model class and parameter fitting

A model *class* $\mathcal{M}$ is a set of possible models, typically parameterized by a vector of parameters $\Theta$. What assumptions will we make about the form of the model? When solving a regression problem using a prediction-rule approach, we might try to find a linear function $h(x; \theta, \theta_0) = \theta^T x + \theta_0$ that fits our data well. In this example, the parameter vector $\Theta = (\theta, \theta_0)$.

For problem types such as classification, there are huge numbers of model classes that have been considered...we'll spend much of this course exploring these model classes, especially neural networks models. We will almost completely restrict our attention to model classes with a fixed, finite number of parameters. Models that relax this assumption are called "non-parametric" models.

How do we select a model class? In some cases, the ML practitioner will have a good idea of what an appropriate model class is, and will specify it directly. In other cases, we may consider several model classes and choose the best based on some objective function. In such situations, we are solving a *model selection* problem: model-selection is to pick a model class $\mathcal{M}$ from a (usually finite) set of possible model classes, whereas *model fitting* is to pick a particular model in that class, specified by (usually continuous) parameters $\Theta$.

## 1.6 Algorithm

Once we have described a class of models and a way of scoring a model given data, we have an algorithmic problem: what sequence of computational instructions should we run in order to find a good model from our class? For example, determining the parameter vector which minimizes the training error might be done using a familiar least-squares minimization algorithm, when the model $h$ is a function being fit to some data $x$.

Sometimes we can use software that was designed, generically, to perform optimization. In many other cases, we use algorithms that are specialized for ML problems, or for particular hypotheses classes. Some algorithms are not easily seen as trying to optimize a particular criterion. In fact, a historically important method for finding linear classifiers, the perceptron algorithm, has this character.