# 6 Machine learning

The term *machine learning* is used to describe algorithms that can solve a problem or task by *learning from data* as opposed to being programmed explicitly for the job. In order for a machine learning problem to be well specified, we must determine the following three elements:

*Task* Which task to solve

*Data* Which data to use

*Loss* How to measure how well we solve the task

Once these elements have been specified, the machine learning problem is well defined. The next step is to create an algorithm that uses the data to solve the task in a way that minimizes the loss.

> Example 6.1 Predicting a person's weight from their height
>
> Let us say, that we want to create a machine learning system for the *task* of predicting how much a person weighs based on a measurement of their height. The *data* we would need to gather are measurements of height and weight for a representative sample of people. Finally we could for example define our *loss* as the absolute difference between our estimated weight and the true weight.
>
> Once we have made this decisions, the machine learning problem is completely specified. This means that we can now design and compare different methods.

## 6.1 Types of learning problems

We can distinguish between different types of learning problems, including supervised, unsupervised, and reinforcement

learning.

**Definition 6.1 Supervised, unsupervised, and reinforcement learning**

| | |
|---|---|
| *Supervised* | Learning a mapping from features (input) to a target (output). |
| *Unsupervised* | Learning about patterns and structure in a data set. |
| *Reinforcement* | Learning to take actions in an environment. |

Although not all learning problems can be neatly classified into one of these three categories, it is a useful way to categorize many different problems and algorithms. In the following, we will go through each type of learning problem and mention some of the most important methods within each category, also outlined in Figure 6.1.

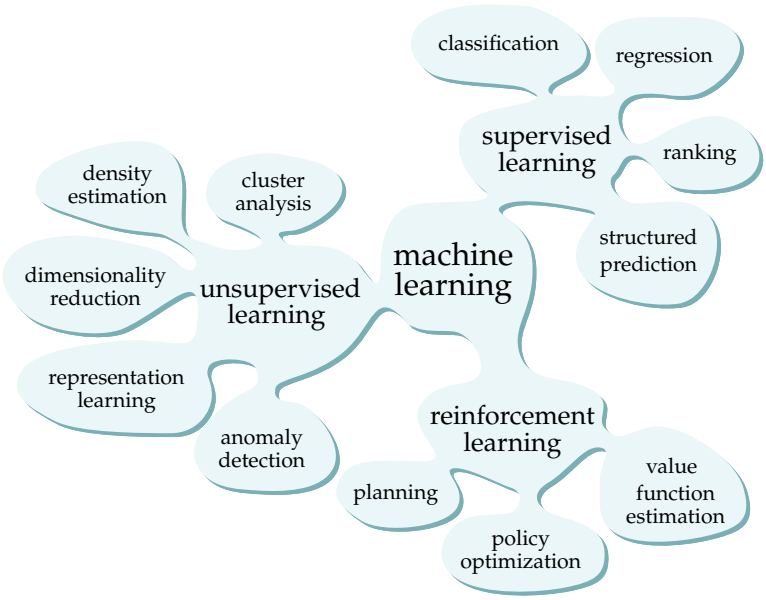


Figure 6.1: Mind map of different learning problems within machine learning.

### 6.1.1 *Supervised learning*

In supervised learning our data is a set of features $x$ and targets $y$. The goal is to learn a mapping between $x$ and $y$ in the form of a function from the features to the target, $y \approx f(x)$.

The training data $\mathcal{D}$ often comes in the form of $N$ *exchangeable* pairs of $x$- and $y$-values:

$$\mathcal{D}_{\text{s}} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_N, y_N)\}.$$

By exchangeable, we mean that we do not care about the order in which the data occur in our collection. In other words, the numbering of the observations is arbitrary. All we care about is that we have $N$ examples of an $x$ and a $y$ that go together.

The features $x$ can be almost anything, including continuous or discrete numbers, vectors, strings, etc. Each target $y$ is usually single continuous or discrete value.

Based on the training data, we want to optimally determine a function $f(x)$, so that when we get a new observation $x^*$ we can simply map it through our function to obtain our prediction $y^*$,

$$y^* = f(x^*).$$

*Regression* In the case where the target $y$ that we want to predict is a *continuous value* the learning problem is called *regression*. Some examples of regression could be to predict a person's weight, the fuel consumption of a vehicle, or the price of a stock.

*Classification* When the quantity $y$ that we want to predict takes a value in a discrete set, the learning problem is called *classification*. Examples of classification include predicting whether or not it will rain tomorrow, determining the value 0–9 of a hand-written digit, or identifying an object in a photograph from a fixed set of possibilities.

*Structured prediction* When the target $y$ is not a scalar, but a more complicated, structured object, the supervised learning problem is called *structured prediction*.

*Ranking* The supervising target could also be in the form of an ordering of the observations. In that case we could be interested in learning a function that can rank our observations, so that if we come with a new observation or a new set of observations, they can be ranked in the context of the existing data set or be ranked in similar manner as the existing data.

### 6.1.2 *Unsupervised learning*

In unsupervised learning our observed data is a set of features $x$. As opposed to the supervised setting, here we have no

targets $y$ to predict. Rather, the goal is to learn a description of the patterns and structure in the data observations.

The data $\mathcal{D}_{\text{u}}$ often comes in the form of $N$ exchangeable observations:

$$\mathcal{D}_{\text{u}} = \{x_1, x_2, x_3, \ldots, x_N\}.$$

As before, these observations could be in the form of numbers, vectors, strings, or anything else imaginable.

*Cluster analysis* One way to characterize structure in such a data set is to divide the observations into different groups in such a way that observations that are similar go in the same group. For example, we might have a data set where each observation corresponds to a customer in a supermarket and represents the customer's purchase habits. A cluster analysis could then be used to identify groups of customers with similar behavior, which then could be targeted more precisely in a marketing campaign.

*Anomaly detection* Sometime our goal is to detect observations which fall outside the typical pattern. For example, a bank might be interested in analyzing the patterns of its customers' credit card transactions to find unexpected events. Transactions carried out in a unusual currency, at an unusual time of day, or with an unusual amount might be an indication of fraud, which could be looked further into. This is similar to cluster analysis, where we group observations into just two categories: normal and anomalous.

*Dimensionality reduction* The goal of dimensionality reduction is to represent each of the observations in a compact form such that we loose as little information about the observations as possible. We could learn a (low-dimensional) vector representation $z_n$ for each observation $x_n$ so that $x_n \approx f(z_n)$ for some fixed function $f(z)$. In a way, cluster analysis can be seen as a special case of dimensionality reduction, where each observations is represented only by which cluster it belongs to.

*Representation learning* As opposed to dimensionality reduction, where we learn a compact representation optimized for reconstructing our data, in *representation learning* we learn a representation that is optimized for some secondary supervised task. In a sense, representation learning is supervised,

but the focus is to learn a useful representation of the data and not to do well on the supervised task. The idea is that a representaton that is useful in some taks might also be useful in other related tasks, and that learning a representation can help transfer knowledge to other domains.

*Density estimation* Fitting a probability density to a data set can also be seen as an unsupervised learning task. The probability distribution could possibly be very complicated with lots of parameters to fit, so while it might seem like a simple problem to fit a distribution to data, it might actually be a very difficult learning problem. Once the data has been fitted, the probability distribution can be used for many purposes: For example, we can simulate new fake data from the distribution, or we can use it to measure how likely some new observation is to see if it fits in with the remaining data.

### *6.1.3 Reinforcement learning*

In reinforcement learning we have a dynamic setting where our machine learning system acts as an agent in an environment. The data is a sequence of environment states $s$, actions $a$, and rewards $r$,

$$\mathcal{D}_{\mathrm{r}} = (s_1, a_1, r_1), (s_2, a_2, r_2), \ldots.$$

At first we observe the initial state $s_1$, and then we must decide which action $a_1$ to take, after which we observe the associated reward $r_1$ and the new state $s_2$. Now we must decide which action $a_2$ to take, after which we observed the new reward $r_2$ and the new state $s_3$ and so on. Both the states and the actions can be in the form of continuous or discrete numbers or vectors or anything else imaginable, whereas the reward is typically a scalar quantity. The objective is to take actions to maximize the expected cumulative reward, which is the reward our agent will gather in the long run.

There is an inherent conflict between *exploration* and *exploitation* in reinforcement learning. After having observed a number of state-action-reward triplets, we might be able to come up with a reasonable policy that we can exploit to give a high reward. But maybe it would actually be better to take actions that giver a lower reward in the short term, but explores a larger area of the state space: Maybe that leads us to discover a different policy with an even better long term reward.

*Policy optimization* The most general reinforcement problem is to learn an optimal policy. A policy is a function $\pi(s) \to a$ that maps a state to an action. Since we can only learn about the system by interacting with it, learning an optimal policy is an iterative process which explores the system, learns about it, and eventually exploits what it has learned.

*Value estimation* If we knew the *value* (i.e. the expected reward) of each environmental state, an optimal policy would simply be to take the action which leads to the most valuable state. Some reinforcement learning algorithms focus on learning a value function $v(s)$ or a state-action value function $q(s, a)$ which then implicitly defines the optimal policy.

*Planning* In the general case, the only way to learn something about the environment is to interact with it. However, in some cases we have access to a simulation of the environment that allows us to run experiments that explore the state space. The problem of learning a policy (or value function) based on such a simulation model is known as *planning*.

## 6.2 Learning as optimization

Once we have a well defined machine learning problem with a given task, data, and loss, we can view what remains as an *optimization* problem: We need to find a solution that solves the task by minimizing the loss measured on the observed data.

We note that the optimization view is a very clear and direct way to think about machine learning, it is not the only way. Not all machine learning methods can be viewed as optimization problems, but some can better be understood from a statistical inference view or from a purely algorithmic view etc. However, in the following we will describe the learning problem from an optimization perspective.

### *6.2.1 Supervised learning as optimization*

In supervised learning, where we want to find a function that maps from features to targets, we can set up a loss function that measures how well our function fits the true targets. We can then optimize (minimize) that loss to give us the best fitting function. In practice, the functions we consider is often a parametric family: In other words, we consider all functions with

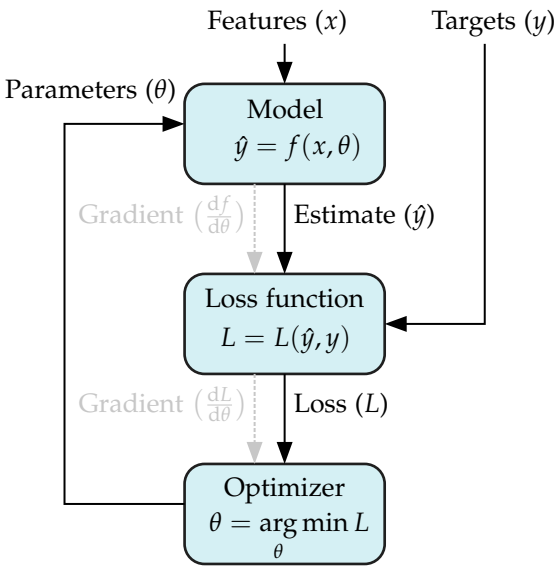


Figure 6.2: Supervised learning flow diagram. Input to the learning algorithm are features $x$ and targets $y$. The goal is to learn the parameters $\theta$ of a function $f_\theta(x)$ that maps from features to targets. A loss function $L(y, \hat{y})$ defines the loss of predicting $\hat{y}$ when the true target is $y$. The parameters are found by minimizing the loss. In some algorithms, the optimizer uses the gradient of the loss to adjust the parameters.

a particular mathematical formulation with some unknown parameters. Then the problem becomes that of selecting the optimal value for those parameters. A flow chart of supervised learning from the perspective of optimization is given in Figure 6.2.

### Example 6.2 Projectile motion

Let us say that we have a cannon that we can fire at different angles on a flat field (illustrated in Figure 6.3). We assume that the projectile always leaves the cannon with the same initial velocity. From physics we know that the laws of motion say that the height and distance of the projectle as a function of time are given by

$$y(t) = v_0 \sin(\phi) t - \tfrac{1}{2} g t^2,$$
$$x(t) = v_0 \cos(\phi) t,$$

where $v_0$ is the initial veolcity, $\phi$ is the angle of the cannon and $g = 9.81$ is the gravitational acceleration. We can then solve for the range $R$ (the distance where the projectile hits

the ground)

$$R = \frac{v_0^2 \sin(2\phi)}{g}$$

Now let us say that we have a data set that consists of measured ranges for different angles (the measurements do not exactly follow the theory because of measurement noise):

| $\phi$ | 10° | 20° | 30° | 40° | 50° | 60° |
|---|---|---|---|---|---|---|
| $R$ | 87 | 127 | 173 | 193 | 212 | 156 |

We would like to use supervised learning to find a function that allows us to compute the range for any angle. Our data set is $\mathcal{D}_s = \{(\phi_1, R_1), (\phi_2, R_2), \ldots, (\phi_6, R_6)\}$. Assuming that the formula for the range above holds, we choose the parameterized family of functions given by

$$\hat{R}(\phi, v_0) = \frac{v_0^2 \sin(2\phi)}{g}.$$

As $\phi$ is our input feature, the only unknown parameter is the initial velocity $v_0$.

We can now define our loss function: We decide to use the mean absolute difference between the measured and modelled range:

$$L(R, \hat{R}) = \frac{1}{6} \sum_{i=1}^{6} |R_i - \hat{R}_i|$$

Finally, to select the optimal value of $v_0$ we can simply plot the loss function and locate its minimum. In Figure 6.3 we can see that the minimum is located around $\hat{v}_0 = 45$.

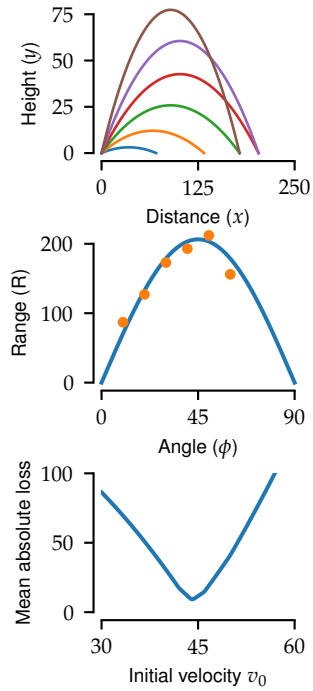


Figure 6.3: Projectile motion. Top: Six trajectories of projectiles fired with same initial velocity at different angles. Middle: The observed range of each of the six projectiles; solid line is the theoretical range as a function of the angle. Bottom: Mean absolute error between the model and the observations for different values of the parameter $v_0$.

### 6.2.2 *Unsupervised learning as optimization*

Many unsupervised learning methods can also be viewed as optimization problems. Here, the goal is to learn a representation of our data set. As before, we need to choose a loss function: Here, the loss measures how well our representation matches our data set. The representation depends on a set of parameters, which again we would like to optimize. Sometimes we have fixed number of parameters, that does not change with the

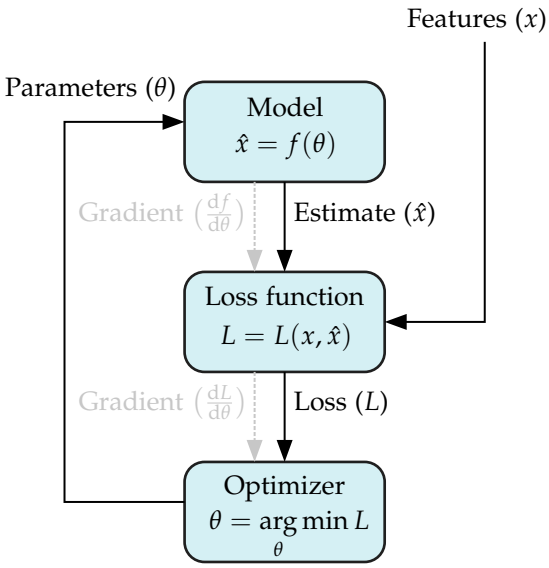


Figure 6.4: Unsupervised learning flow diagram. Input to the learning algorithm is a data set with features $x$. The goal is to learn a representation $f(\theta)$ of $x$ by optimizing a set of parameters $\theta$. A loss function measures the discrepancy between the representation $\hat{x}$ and the real data set $x$.

number of data points; however, very often we have specific parameters associated with each observation, such that the number of parameters increases with the size of the data. A flow chart of unsupervised learning form the perspective of optimization is given in Figure 6.4.

### Example 6.3 Two cannons

Imagine we have measured the range of 10 shots from two different cannons, illustrated in Figure 6.5. We do not know how many shots were fired from each cannon, but we expect that one of the cannons is more powerful than the other. Based on this data, we would like to determine which shot came from which cannon. To do this, we take an unsupervised learning approach, where we assume that each cannon can be represented by a fixed range, and that deviations from this range is measurement noise. Thus, we can formulate the following model:

$$\hat{R}_i = \begin{cases} R_\mathrm{A} & \text{if shot } i \text{ came from cannon A} \\ R_\mathrm{B} & \text{if shot } i \text{ came from cannon B} \end{cases}$$

The parameters of our model are the two values $R_\mathrm{A}$ and

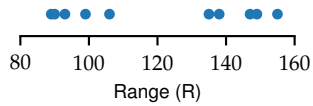


Figure 6.5: Measured ranges of ten cannon shots from two cannons with different power.

$R_B$ as well as the assignment of each shot to one of the cannons. Mathematically, the assignment can be formulated as an indicator variable

$$c_i = \begin{cases} 0 & \text{if shot } i \text{ came from cannon A,} \\ 1 & \text{if shot } i \text{ came from cannon B.} \end{cases}$$

Based on this, we can write our model as

$$\hat{R}_i = (1 - c_i) \cdot R_A + c_i \cdot R_B$$

Next, we must formulate a loss function. Here, we will use the mean squared difference between the measured and hypothesized range:

$$L(R, \hat{R}) = \frac{1}{10} \sum_{i=1}^{10} (R_i - \hat{R}_i)^2.$$

Minimizing this loss is perhaps not so easy. With ten shots from two cannons, the number of different ways we can assign each shot to a cannon is $2^{10} = 1024$. We could try out each combination, and then find the optimal values for $R_A$ and $R_B$ for each, and finally select the values that gives the lowest loss.

Doing this, we find out that there are actually two solutions that minimize the loss. In the first solution, the five shortest range shots are assigned to cannon A, and $\hat{R}_A = 95.4$, $\hat{R}_B = 144.8$. In the other solution, the five shortest range shots are assigned to cannon B and the estimated ranges for the two cannons are the same as before but switched between the two cannons.

## 6.3 Evaluating performance

So far we have said that in order for a machine learning problem to be well defined, we must specify a loss function that tells us how well the task is solved, and which we can attempt to minimize to get the to the best solution. In supervised learning, the loss would be a measure of how well we can use the features to predict the targets. In unsupervised learning, the loss would be a measure of how well the data is represented. In reinforcement learning, the loss is inversely related to the cumulative reward.

However, in all these cases, the loss we should *really* minimize is not the loss on the particular data we have available, but rather the loss on the data we will see in the future when we put our machine learning system into action. It is not so interesing how well we can characterize the data that we already have available: Of course, if our method is a poor fit for our available data, we cannot expect it to perform well on future data. But even if our method fits our data well, we are not guaranteed to have good performance on future data. In fact, it is trivial to construct a method that perfectly fits any data set, simply my memorizing all the data. The challenge is to solve the problem in a way that will *generalize* to new data.

### Example 6.4 Perfect fit?

Let us say we have a supervised learning task, where we want to find a function that maps from $x$ to $y$, and say we are given the following observations:

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| y | True | False | True | False | True | False | True |

Now I come up with a proposed solution:

$$f(x) = \begin{cases} \text{False} & \text{if } x \in \{1,3,5\} \\ \text{True} & \text{otherwise.} \end{cases}$$

In other words, my function simply outputs True if $x$ is equal to 1, 3, or 5, and outputs False in all other cases. This function fits the observed data perfectly.

My optimistic colleague proposes the following solution, which fits only $\frac{4}{7} \approx 57\%$ of the observed data,

$$g(x) = \text{True}.$$

Clearly, my solution $f$ is better than my colleague's $g$: My solution fits the observed data better, and both solutions will give the same result, True, for any other $x$ not included in the set of observations.

Next, my student comes up with another proposal,

based on an observation about a pattern in the data,

$$h(x) = \begin{cases} \text{True} & \text{if } x \text{ is even} \\ \text{False} & \text{otherwise.} \end{cases}$$

Her solution $h$ fits the observed data perfectly just like my solution: The only difference between $f$ and $h$ is what they predict for values of $x$ outside the observed data.

Now which solution is best? We cannot really tell without any further information or assumptions. The point is, that what really matters is how well the solution will generalize to new, unseen data.

### 6.3.1 *Training and test*

What we need is a way to measure how well our methods work on future data. One of the most simple and popular ways to measure this is *cross validation*. There are many different cross validation schemes, but the most simple way is to divide the observed data set in two parts: One part will be used for *training* the machine learning model, and the other part will be used for *testing* the learned solution. This approach is called the *hold out* method. Since we only use the training data to fit our model, we can use the test data to measure how well the method will generalize: The performance of the test data is an estimate of the generalization error.

Using cross validation, we can then compare different models: A too simple model can not be expected to fit the training data very well, and it will also not generalize well to new data. A too complex model will fit the training data very well, maybe even perfectly, but it will also not generalize well to new data. Somewhere in between these extremes, we can hope to find a model with suitable complexity: One that not only fits the training data reasonably, but which also will perform well on data in the future (see Figure 6.6 for an illustration).

It can be very beneficial to think about machine learning as a type of *inferential statistics*. As you may recall, in inferential statistics we observe a sample from a population, and we the use sample statistics to estimate parameters of the population.

In machine learning, we can think of the training data as a sample from a population. When we fit a machine learning model we are not just interested in finding a model that per-

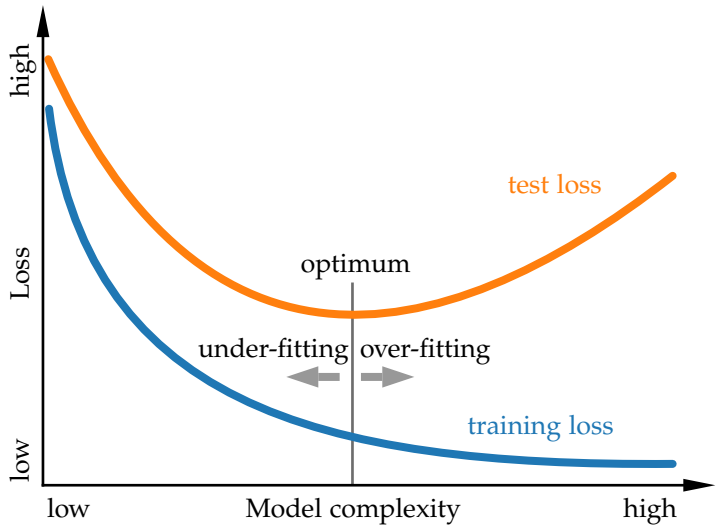


Figure 6.6: Overfitting and underfitting. A very simple model cannot be expected to fit the training data well, and does not have good generalization. This is known as *underfitting*. By increasing the complexity of the model, we can fit the training data arbitrarily well; however, while fitting the training data very well, a too complex model will also not have good generalization. This is known as *overfitting*. The optimal model complexity can be found somewhere between these extremes.

forms well on this particular training data sample. Ideally we would like a model that performs well on *any* sample from the population. A good way to measure how well our model's performance generalizes is to test it on a new, independent sample from the population, which is exactly the role of the test set.

## 6.4 Linear regression

One of the most simple and effective methods for supervised learning is linear least squares regression. The objective is to predict a continuous target variable $y$ based on a set of observed features $x$. In linear regression, we use a linear function to map between the features and the target. If we have just a single scalar feature $x$, the linear regression formula can be written as

$$\hat{y} = f(x) = w_0 + w_1 \cdot x.$$

In other words, we fit the target variable with a straight line with intercept $w_0$ and slope $w_1$. In *least squares* regression, the loss function is the mean squared error between the true and the predicted target:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

To fit the linear regression model, we choose the parameters $w_0$ and $w_1$ that minimize the loss. A nice property of the linear

model in combination with the squared error loss is that we can find the optimal parameters very efficiently by solving a system of equations. To do this, we can compute the derivative of the loss with respect to each of the parameters, set the derivative equal to zero, and solve for the parameters.

$$\begin{aligned}
\frac{\partial L}{\partial w_0} &= \frac{1}{N}\sum_{i=0}^{N} -2\big(y_i - (w_0 + w_1 \cdot x_i)\big) = 0 \\
\Rightarrow & w_0 + w_1 \cdot \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{1}{N}\sum_{i=1}^{N} y_i \\
\frac{\partial L}{\partial w_1} &= \frac{1}{N}\sum_{i=0}^{N} -2\big(y_i - (w_0 + w_1 \cdot x_i)\big)x_i = 0 \\
\Rightarrow & w_0 \frac{1}{N}\sum_{i=1}^{N} x_i + w_1 \cdot \frac{1}{N}\sum_{i=1}^{N} x_i^2 = \frac{1}{N}\sum_{i=1}^{N} y_i x_i
\end{aligned}$$

Thus, we have two equations in two unknowns:

$$\begin{aligned}
a \cdot w_0 + b \cdot w_1 &= c \\
d \cdot w_0 + e \cdot w_1 &= f
\end{aligned} \tag{6.1}$$

where $a = 1$, $b = d = \frac{1}{N}\sum_{i=1}^{N} x_i$, $c = \frac{1}{N}\sum_{i=1}^{N} y_i$, $e = \frac{1}{N}\sum_{i=1}^{N} x_i^2$, and $f = \frac{1}{N}\sum_{i=1}^{N} y_i x_i$.

This means that we can find the optimal intercept and slope by solving two equations in two unknowns, where the coefficients of the equations are computed from the data.

### Example 6.5 Linear regression

Consider the dataset with $N = 10$ observations shown in Figure 6.7. Let us say we would like to fit a linear regression model to predict $y$ from $x$. First, let us see what we can say about the parameters just from visually inspecting the figure. Since we have

$$y = w_0 + w_1 \cdot x,$$

the intercept $w_0$ is equal to the y-value for $x = 0$. From the figure approximatly read off the value

$$w_0 \approx -2.$$

The slope, $w_1$, can be approximated by reading off two

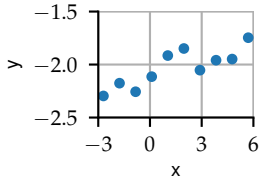


Figure 6.7: A simple linear regression example.

$(x, y)$-pairs from the grap: Since we have

$$y_A = w_0 + w_1 \cdot x_A, \qquad y_B = w_0 + w_1 \cdot x_B,$$

we may write

$$y_A - y_B = (w_0 + w_1 \cdot x_A) - (w_0 + w_1 \cdot x_B) = w_1(x_A - x_B),$$

and solve for the slope

$$w_1 = \frac{y_A - y_B}{x_A - x_B}.$$

From the figure we can imagine a straight line fit to the data and get two approximate $(x, y)$-pairs: $(x_A, y_A) \approx (-3, -2.3)$ and $(x_A, y_A) \approx (6, -1.7)$ which gives

$$w_1 \approx \frac{-2.3 - (-1.7)}{-3 - 6} \approx 0.07.$$

Next, let us use the data to compute the coefficients for the two equations in two unknowns from Equation 6.1. To do this, we of course need to have access to the actual data set, so let us just assume that we were able to find the following coefficients,

$$\begin{aligned} 1 \cdot w_0 + 1.50 \cdot w_1 &= -2.03 \\ 1.50 \cdot w_0 + 9.44 \cdot w_1 &= -2.66. \end{aligned}$$

Solving this system of equations gives us the exact solution,

$$w_0 \approx -2.11, \qquad w_1 \approx 0.054,$$

which is not too far of from our rough estimate.

## Problems

1. In an image recognition system, what could the task, data, and loss be?

2. Classify the following problems as unsupervised, supervised, or reinforcement learning.

   (a) Learn to control a car by recording what human drives do and training a system to do the same.

   (b) Learn to control a helicopter by trial and error while encouraging it not to crash.

   (c) Learn how human cognitive ability can best be described by a single number based on a test questionnaire.

3. Say I want to predict how much fuel my car will use on a trip based on data from previous trips. I have data from 20 previous trips where I recorded the length of the trip, whether it was in the city, on the highway, or on the motorway, and the number of people in the car, along with a lot of other features related to the trips. Now I fit two least squares linear regression models to predict the fuel consumption: The first model uses only the length of the trip as an input feature and gives a mean squared error of 1.32 on the 20 trips used as training data. The second model uses all of my recorded features as input and gives a mean squared error of 0.68. Which of the two regression models is best?

4. Say we are given the following values for the fuel consumption and the trip length

| Trip length ($x$) | 10 | 15 | 12 | 40 | 13 |
|---|---|---|---|---|---|
| Fuel consumption ($y$) | 0.4 | 0.7 | 0.6 | 1.9 | 0.6 |

   (a) Plot the data

   (b) Using least squares linear regression, fit the following model

$$\hat{y} = f(x) = w \cdot x.$$

   Hint: Write down the loss function. Compute the derivative of the loss with respect to the parameter $w$. Set the derivative equal to zero, and solve for $w$.

## Solutions

1. The task is to recognize an object in an image. The data is typically a large database of images with known object labels. The loss is related to the accuracy of the classifier: Most often classifiers output a probability for each class, and the loss is defined so that the probability of the correct class is maximized.

2. (a) Supervised learning. (b) Reinforcement learning. (c) Unsupervised learning.

3. We cannot tell from the performance on the training data: Even though the second model gives a lower training loss, it might not generalize to new data. We would need to use cross validation and test the models on a set of trips that are not used for training the model.

4.(a) See Figure 6.8

(b) $w \approx 0.047$.

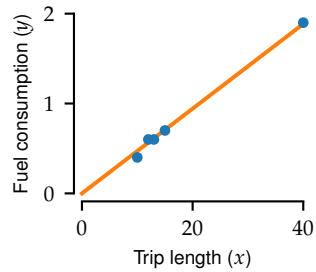


Figure 6.8: Plot of fuel consumption vsersus trip length