**Introduction**

Completed100 XP

* 2 minutes

The way we train models is by no means a perfectly automated process. Training's blind reliance on data can lead it to learn things that aren't helpful in the end, or to not effectively learn things that are actually useful. The following learning material walks through some simple reasons why underfitting and overfitting take place, and what you can do about it.

**Scenario: Training avalanche rescue dogs**

Throughout this module, we’ll be using the following example scenario to explain underfitting and overfitting. This scenario is designed to provide an example for how you might meet these concepts while programming for yourself. Keep in mind that these principles generally apply to almost all types of models, not just those we work with here.

It’s time for your charity to train a new generation of dogs in how to find hikers swept up by avalanches. There's debate in the office as to which dogs are best; is a large dog better than a smaller dog? Should the dogs be trained when they're young or when they're more mature? Thankfully, you have statistics on rescues performed over the last few years that you can look to. Training dogs is expensive, though, and you need to be sure that your dog-picking criteria are sound.

**Prerequisites**

* Familiarity with machine learning models

**Learning objectives**

In this module, you'll:

* Define feature normalization.
* Create and work with test datasets.
* Articulate how testing models can both improve and harm training.

# Normalization and standardization

Completed100 XP

* 4 minutes

Feature Scaling is a technique that changes the range of values that a feature has. Doing so helps models learn faster and more robustly.

## Normalization versus standardization

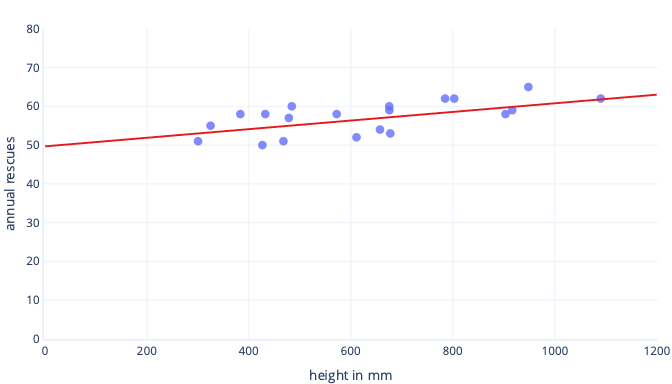
Normalization means to scale values so that they all fit within a certain range, typically 0–1. For example, if you had a list of people’s ages that were 0, 50, and 100 years, you could normalize by dividing the ages by 100, so that your values were 0, 0.5, and 1.

Standardization is similar, but instead, we subtract the mean (also known as the average) of the values and divide by the standard deviation. If you’re not familiar with standard deviation, not to worry, this means that after standardization, our mean value is zero, and about 95% of values fall between -2 and 2.

There are other ways to scale data, but the nuances of these are beyond what we need to know right now. Let’s explore why we apply normalization or standardization.

## Why do we need to scale?

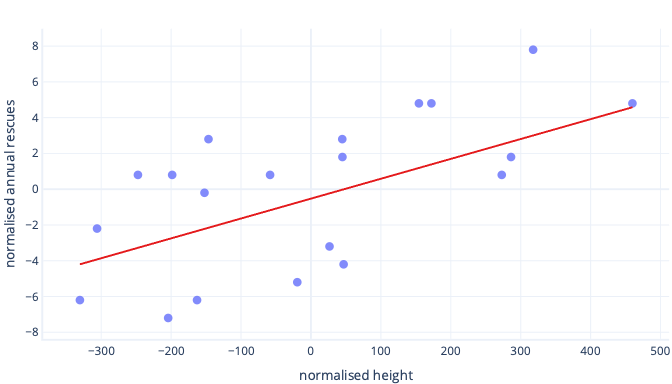
There are many reasons we normalize or standardize data before training. You can understand these more easily with an example. Let’s say we want to train a model to predict whether a dog will be successful at working in the snow. Our data are shown in the following graph as dots, and the trend line we're trying to find is shown as a solid line:



### Scaling gives learning a better starting point

The optimal line in the preceding graph has two parameters: the intercept, which is 50, the line at x=0, and slope, which is 0.01; each 1000 millimeters increases rescues by 10. Let’s assume we start training with initial estimates of 0 for both of these parameters.

If our training iterations are altering parameters by around 0.01 per iteration on average, it takes at least 5000 iterations before the intercept is found: 50 / 0.01 = 5000 iterations. Standardization can bring this optimal intercept is closer to zero, which means we can find it much faster. For example, if we subtract the mean from our label—annual rescues—and our feature—height—the intercept is -0.5, not 50, which we can find about 100 times faster.



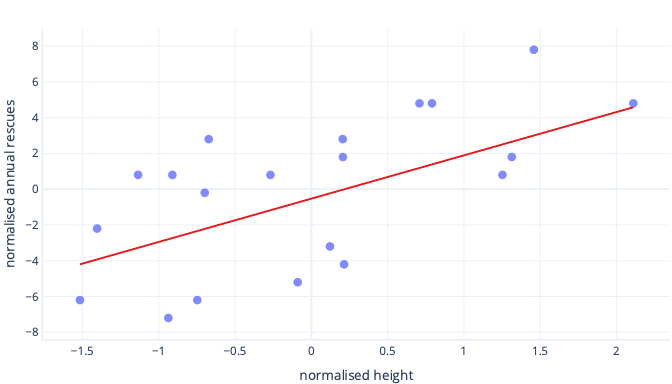
There are other reasons that complex models can be very slow to train when the initial guess is far from the mark, but the solution is still the same: offset features to something closer to the initial guess.

### Standardization lets parameters train at the same speed

In our newly offset data, we have an ideal offset of -0.5 and an ideal slope of 0.01. Although offsetting helps speed things up, it's still much slower to train the offset than to train the slope. This can slow things down and make training unstable.

For example, our initial guesses for offset and slope are both zero. If we're changing our parameters by about 0.1 on each iteration, we'll find the offset quickly, but it will be very difficult to find the correct slope, because increases in slope will be too large (0 + 0.1 > 0.01) and can overshoot the ideal value. We can make the adjustments smaller, but this will slow down how long it takes to find the intercept.

What happens if we scale our feature of height?



The slope of the line is now 0.5. Pay attention to the x-axis. Our optimal intercept of -0.5 and slope of 0.5 are the same scale! It's now easy to pick a sensible step size, which is how fast the gradient descent updates parameters.

### Scaling helps with multiple features

When we work with multiple features, having these on a different scale can cause issues in fitting, similarly to how we just saw with the intercept and slope examples. For example, if we're training a model that accepts both height in mm and weight in metric tons, many kinds of models will struggle to appreciate the importance of the weight feature, simply because it's so small relative to the height features.

## Do I always need to scale?

We don’t always need to scale. Some kinds of models, including the preceding models with straight lines, can be fit without an iterative procedure like gradient descent, so they don't mind features being the wrong size. Other models do need scaling to train well, but their libraries often perform feature scaling automatically.

Generally speaking, the only real downsides to normalization or standardization are that it can make it harder to interpret our models and that we have to write slightly more code. For this reason, feature scaling is a standard part of creating machine learning models.

# Test and training datasets

Completed100 XP

* 4 minutes

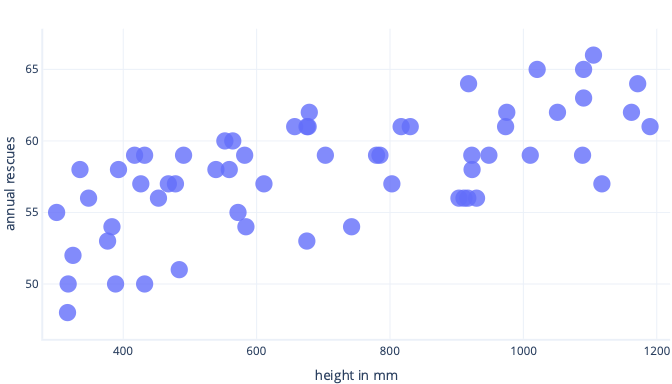
The data we use to train a model is often called a training dataset. We’ve already seen this in action. Frustratingly, when we use the model in the real world, after training we don’t know for certain how well our model will work. This uncertainty is because it’s possible that our training dataset is different to data in the real world.

## What is overfitting?

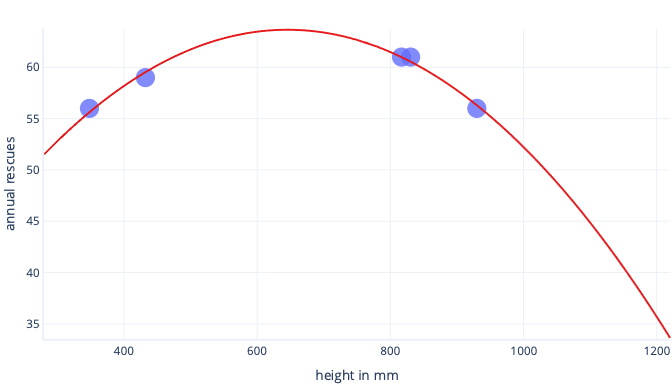
A model is overfit if it works better on the training data than it does on other data. The name refers to the fact that the model has been fit so well that it's memorized details of the training set rather than finding broad rules that will apply to other data. Overfitting is common, but not desirable. At the end of the day, we only care how well our model works on real-world data.

## How can we avoid overfitting?

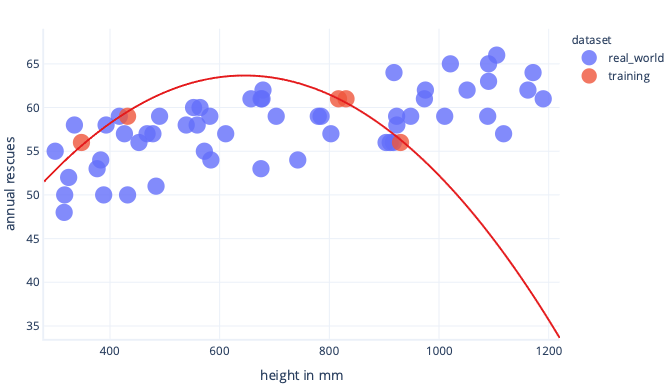
We can avoid overfitting several ways. The simplest way is to have a simpler model, or to use a dataset that's a better representation of what is seen in the real world. To understand these methods, consider a scenario where real-world data look like so:



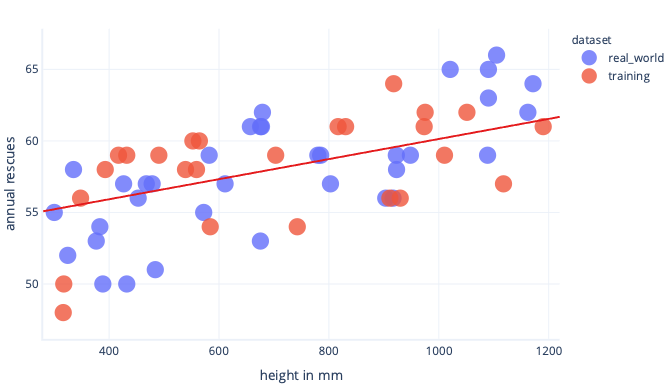
Let’s say we collect information about only five dogs, though, and use that as our training dataset to fit a complex line. If we can do so, we can fit it very well:



When this is used in the real world, though, we'll find it makes predictions that turn out to be wrong:



If we have a more representative dataset and a simpler model, the line we fit turns out to make better (although not perfect) predictions:



A complimentary way we can avoid overfitting is to stop training after the model has learned general rules, but before the model is overfit. This requires detecting when we're beginning to overfit our model, though. We can do this using a test dataset.

## What is a test dataset?

A test dataset, also called a validation dataset, is a set of data similar to the training dataset. In fact, test datasets are usually created by taking a large dataset and splitting it. One portion is called the training dataset, and the other is called the test dataset.

The job of the training dataset is to train the model; we’ve seen training already. The job of the test dataset is to check how well the model works; it doesn't contribute to training directly.

### OK, but what’s the point?

The point of a test dataset is twofold.

First, if test performance stops improving during training, we can stop; there's no point in continuing. If we do continue, we can end up encouraging the model to learn details about the training dataset that aren't in the test dataset, which is overfitting.

Secondly, we can use a test dataset after training. This gives us an indication of how well the final model will work when it sees "real-world" data it hasn't seen before.

### What does that mean for cost functions?

When we use both training and test datasets, we end up calculating two cost functions.

The first cost function is using the training dataset, just like we've seen before. This cost function is fed to the optimizer and used to train the model.

The second cost function is calculated using the test dataset. We use this to check how well the model might work in the real world. The result of the cost function isn't used to train the model. To calculate this, we pause training, look at how well the model is performing on a test dataset, and then resume training.

# Nuances of test sets

Completed100 XP

* 6 minutes

Test sets are considered best practice for most aspects of machine learning, though the field is still relatively young, and so exactly how and when is often debated. Let’s go through some things to consider.

## Test sets can be misleading

Although test sets are helpful to identify overtraining, they can provide us with false confidence. Specifically, test sets are only useful if they reflect data that we expect to see in the real world. For example, our test set is very small, so it won't be representative of the variety of data that we're likely to see in the real world. Test datasets are also only as good as their source. If our test dataset comes from a biased source, our metrics won't reflect how things will behave in the real world.

For example, let’s say we're trying to find the relationship between number of rescues and the age a dog started training. If our test set was only three dogs, it's possible that these dogs aren't a good representation of the wide variety of working dogs in the real world. Also, imagine that we obtained our test set from a single breeder who doesn't know how to work with puppies. Our model might predict that older dogs are best to train, and our test dataset would confirm this, when in fact other trainers might have enormous success with younger animals.

## Test sets aren't free

We’ve already seen that the more training data we have, the less likely our model will overfit. Similarly, the larger the test sets, the more we feel we can trust our test results. However, we usually work with finite amounts of data, and a datapoint can't be in both the training and the test set. This means that as we get larger test sets, we get smaller training datasets and vice versa. Exactly how much data should be sacrificed to appear in the test dataset depends on individual circumstances, with anything between 10-50% being relatively common, depending on the volume of data available.

## Train and test isn't the only approach

It’s worth keeping in mind that train-and-test is common, but not the only widely used approach. Two of the more common alternatives are the hold-out approach and statistical approach methods.

### The hold-out approach

The hold-out approach is like train-and-test, but instead of splitting a dataset into two, it's split into three: training, test (also known as validation), and hold-out. The training and test datasets are as we’ve described previously. The hold-out dataset is a kind of test set that's used only once, when we're ready to deploy our model for real-world use. In other words, it's not used until we've finished experimenting with different kinds of training regimens, different kinds of models, and so on.

This approach tackles the fact that we usually experiment with different models and training regimens. For example, we fit a model, find it doesn't work well with the test dataset, change some aspects of the model being trained, and try again until we get a good result. This means we're purposefully altering our model to work for a particular set of data, just like normal training does with the training dataset. By doing this, we can end up with a model that's essentially too overtrained to work on our test dataset.

The idea of a third dataset is that we can test for this, too. This approach means splitting the data three ways, which means we start with even less training data. If we don't have a lot of data to work with, this approach can reduce our ability to obtain a good model.

### Statistical approaches

Simpler models that have originated in statistics often don't need test datasets. Instead, we can calculate what degree the model is overfit directly as statistical significance: a p-value.

These statistical methods are powerful, well established, and form the foundation of modern science. The advantage is that the training set doesn't ever need to be split, and we get a much more precise understanding of how confident we can be about a model. For example, a p-value of 0.01 means there's a very small chance that our model has found a relationship that doesn't actually exist in the real world. By contrast, a p-value of 0.5 means that while our model might look good with our training data, it will be no better than flipping a coin in the real world.

The downside to these approaches is that they're only easily applied to certain model types, such as the linear regression models with which we've been practicing. For all but the simplest models, these calculations can be extremely complex to perform properly, and so are out of scope for the current course. They also suffer the same limitation regarding data selection; if our training data is biased, our p-values will be misleading.

**Summary**

Completed100 XP

* 3 minutes

In this module, you learned about how to test machine learning models. You also learned about normalization and standardization, and reviewed the hold-out approach.

Now that you've reviewed this module, you should be able to:

* Define feature normalization.
* Create and work with test datasets.
* Articulate how testing models can both improve and harm training.

**Use these resources to discover more**

**Tip**

To open a hyperlink, right-click and choose **Open in new tab or window**. That way you can see the resource and easily return to the module.

* [ML.NET Tutorial](https://dotnet.microsoft.com/learn/ml-dotnet/get-started-tutorial/intro)
* [Azure Machine Learning](https://azure.microsoft.com/services/machine-learning/)