This is self-study key concepts documentation for ML practitioners.

The concepts will help you to

1. Reduce programming time
2. Customize and scale the products

Machine Learning changes the way to solve the problem. We use assertions. From mathematical science to the natural science. Observations and experiments.

**SECTION 1**

What is **(supervised) machine learning**? Concisely put, it is the following:

* ML systems learn how to combine input to produce useful predictions on never-before-seen data.

**Labels**

* A label is the thing we're predicting—the y variable in simple linear regression. The label could be the future price of wheat, the kind of animal shown in a picture, the meaning of an audio clip, or just about anything.

**Features**

* A feature is an input variable—the x variable in simple linear regression. A simple machine learning project might use a single feature, while a more sophisticated machine learning project could use millions of features, specified as:
* x1, x2, x3, …..., xN

In the spam detector example, the features could include the following:

* words in the email text
* sender's address
* time of day the email was sent
* email contains the phrase "one weird trick."

**Examples**

An **example** is a particular instance of data, x. (We put x in boldface to indicate that it is a vector.) We break examples into two categories:

* labeled examples
* unlabeled examples

A **labeled example** includes both feature(s) and the label. That is:

* labeled examples: {features, label}: (x, y)

Use labeled examples to **train** the model. In our spam detector example, the labeled examples would be individual emails that users have explicitly marked as "spam" or "not spam."

An **unlabeled example** contains features but not the label. That is:

* unlabeled examples: {features, ?}: (x, ?)

**Models**

A model **defines the relationship between features and label**. For example, a spam detection model might associate certain features strongly with "spam". Let's highlight two phases of a model's life:

**Training** means creating or **learning** the model.

* That is, you show the model labeled examples and enable the model to gradually learn the **relationships between features and label**.

**Inference** means **applying the trained model to unlabeled examples**.

* That is, you use the trained model to make **useful predictions (y')**.
* For example, during inference, you can predict medianHouseValue for new unlabeled examples.

**Regression vs. classification**

A **regression** model predicts **continuous** values.

For example, regression models make predictions that answer questions like the following:

* What is the value of a house in California?
* What is the probability that a user will click on this ad?

A **classification** model predicts **discrete** values.

For example, classification models make predictions that answer questions like the following:

* Is a given email message spam or not spam?
* Is this an image of a dog, a cat, or a hamster?

**Key Terms**

|  |  |
| --- | --- |
|  [classification model](https://developers.google.com/machine-learning/glossary#classification_model) |  [example](https://developers.google.com/machine-learning/glossary#example) |
|  [feature](https://developers.google.com/machine-learning/glossary#feature) |  [inference](https://developers.google.com/machine-learning/glossary#inference) |
|  [label](https://developers.google.com/machine-learning/glossary#label) |  [model](https://developers.google.com/machine-learning/glossary#model) |
|  [regression model](https://developers.google.com/machine-learning/glossary#regression_model) |  [training](https://developers.google.com/machine-learning/glossary#training) |

**SECTION 2**

**Linear Regression**

**Chart, scatter chart

Description automatically generated**

**Fig – Linear Relationship**

By general mathematical convention, the linear relationship between the dependent and in-dependent variables is represent in following equation:

**y = mx + c**

y = the value we're trying to predict.

m = is the slope of the line.

x = is the number of chirps per minute—the value of our input feature.

c = is the y-intercept.

By convention in machine learning, you'll write the equation for a model slightly differently:

**y1 = w1x1 + b**

y1 = is the predicted label (a desired output).

x1 = is a feature (a known input).

w1 = is the weight of feature 1. Weight is the same concept as the "slope" m in the traditional equation of a line.

b = is the bias (the y-intercept), sometimes referred to as w0

The model that relies on three features might look as follows:

**y' = w1x1 + w2x2 + w3x3 + b**

**Key Terms**

|  |  |
| --- | --- |
|  [bias](https://developers.google.com/machine-learning/glossary#bias) |  [inference](https://developers.google.com/machine-learning/glossary#inference) |
|  [linear regression](https://developers.google.com/machine-learning/glossary#linear_regression) |  [weight](https://developers.google.com/machine-learning/glossary#weight) |

**SECTION 3**

**Training and Loss**

**What is training? – Learning weights and bias**

**What is Loss? – Difference between predicted and actual values.**

Training a model simply means **learning (determining) good values for all the weights and the bias from labeled examples.**

In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called **empirical risk minimization**.

**Loss is the penalty for a bad prediction**. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater.

**The goal of training a model is to find a set of weights and biases that have low loss, on average, across all examples**.

For example, Figure 3 shows a high loss model on the left and a low loss model on the right. Note the following about the figure:

The arrows represent loss.

The blue lines represent predictions.

Chart, diagram

Description automatically generated

**Squared loss: a popular loss function : L2 Loss**

The linear regression models we'll examine here use a loss function called squared loss (also known as L2 loss). The squared loss for a single example is as follows:

= the square of the difference between the label and the prediction

= (observation - prediction(**x**))2

= (y - y')2

**Mean square error (MSE)** is the **average squared loss per example over the whole dataset**.

To calculate MSE, sum up all the squared losses for individual examples and then divide by the number of examples:

Text

Description automatically generated

**Key Terms**

|  |  |
| --- | --- |
|  [empirical risk minimization](https://developers.google.com/machine-learning/glossary#ERM) |  [loss](https://developers.google.com/machine-learning/glossary#loss) |
|  [mean squared error](https://developers.google.com/machine-learning/glossary#MSE) |  [squared loss](https://developers.google.com/machine-learning/glossary#squared_loss) |
|  [training](https://developers.google.com/machine-learning/glossary#training) |  |

**SECTION 4**

**Reducing Loss: An Iterative Approach**

The following figure suggests the iterative trial-and-error process that machine learning algorithms use to train a model:

Diagram

Description automatically generated

Iterative strategies are prevalent in machine learning, primarily because they scale so well to large data sets.

The "model" takes one or more features as input and returns one prediction (y’) as output. To simplify, consider a model that takes one feature and returns one prediction:

y’ = b + w1x1

**What initial values should we set for b and w1?**

For linear regression problems, it turns out that the starting values aren't important. We could pick random values, but we'll just take the following trivial values instead:

b = 0 and w1 = 0

Suppose that the first feature value is 10. Plugging that feature value into the prediction function yields:

y’ = 0 + 0\*10 = 0

The "Compute Loss" part of the diagram is the loss function that the model will use. Suppose we use the squared loss function. The loss function takes in two input values:

y' : The model's prediction for features x

y : The correct label corresponding to features x.

At last, we've reached the "Compute parameter updates" part of the diagram. It is here that the machine learning system examines the value of the loss function and generates new values for **b and w1**. For now, just assume that this mysterious box devises new values and then the machine learning system re-evaluates all those features against all those labels, yielding a new value for the loss function, which yields new parameter values. And the learning continues iterating until the algorithm discovers the model parameters with the lowest possible loss. Usually, you iterate until overall loss stops changing or at least changes extremely slowly. When that happens, we say that the model has **converged**.

**Key Point:**

A Machine Learning model is trained by starting with an initial guess for the weights and bias and iteratively adjusting those guesses until learning the weights and bias with the lowest possible loss.

**Key Terms**

|  |  |
| --- | --- |
|  [convergence](https://developers.google.com/machine-learning/glossary#convergence) |  [loss](https://developers.google.com/machine-learning/glossary#loss) |
|  [training](https://developers.google.com/machine-learning/glossary#training) |  |

**SECTION 5**

**Reducing Loss: Gradient Descent**

Suppose we had the time and the computing resources to calculate the loss for all possible values of w1. For the kind of regression problems, we've been examining, the resulting plot of loss vs. w1 will always be convex.

In other words, the plot will always be bowl-shaped, kind of like this:

**Diagram

Description automatically generated**

Convex problems have only one minimum; that is, only one place where the slope is exactly 0. That minimum is where the loss function converges.

Calculating the loss function for every conceivable value of w1 over the entire data set would be an inefficient way of finding the convergence point.

Let's examine a better mechanism—very popular in machine learning—called **gradient descent**.

The first stage in gradient descent is to pick a starting value (a starting point) for w1.

The starting point doesn't matter much; therefore, many algorithms simply set w1 to 0 or pick a random value.

The following figure shows that we've picked a starting point slightly greater than 0:

Diagram

Description automatically generated

The gradient descent algorithm then calculates the gradient of the loss curve at the starting point. Here in Figure 3, the gradient of the loss is equal to the derivative (slope) of the curve.

When there are multiple weights, the **gradient** is a vector of partial derivatives with respect to the weights.

Note that a gradient is a vector, so it has both of the following characteristics:

* a direction (+/-)
* a magnitude

The gradient always points in the direction of steepest increase in the loss function. The gradient descent algorithm takes a step in the direction of the negative gradient to reduce loss as quickly as possible.

Diagram

Description automatically generated

To determine the next point along the loss function curve, the gradient descent algorithm adds some fraction of the gradient's magnitude to the starting point as shown in the following figure:

Diagram

Description automatically generated

The gradient descent then repeats this process, edging ever closer to the minimum.

**Note:** When performing gradient descent, we generalize the above process to tune all the model parameters simultaneously.

For example, to find the optimal values of both w1 and the bias b, we calculate the gradients with respect to both w1 and b. Next, we modify the values of w1 and b based on their respective gradients. Then we repeat these steps until we reach minimum loss.

**Key Terms**

|  |  |
| --- | --- |
|  [gradient descent](https://developers.google.com/machine-learning/glossary#gradient_descent) |  [step](https://developers.google.com/machine-learning/glossary#step) |

<https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent#expandable-1>

The gradient of a function, denoted as follows, is the vector of partial derivatives with respect to all the independent variables:

Text

Description automatically generated

The number of dimensions in the vector is equal to the number of variables in the formula for f; in other words, the vector falls within the domain space of the function. For instance, the graph of the following function f(x,y):

Chart, surface chart

Description automatically generated

The gradient of f(x,y) is a two-dimensional vector that tells you in which (x,y) direction to move for the maximum increase in height. Thus, the negative of the gradient moves you in the direction of maximum decrease in height. In other words, the negative of the gradient vector points into the valley.

In machine learning, gradients are used in gradient descent. We often have a loss function of many variables that we are trying to minimize, and we try to do this by following the negative of the gradient of the function.

**SECTION 6**

**Reducing Loss: Learning Rate**

As noted, the gradient vector has both a direction and a magnitude.

**Gradient descent algorithms multiply the gradient by a scalar known as the learning rate (also sometimes called step size) to determine the next point.**

For example, if the gradient magnitude is 2.5 and the learning rate is 0.01, then the gradient descent algorithm will pick the next point 0.025 away from the previous point.

**Hyperparameters** are the knobs that programmers tweak in machine learning algorithms. Most machine learning programmers spend a fair amount of time tuning the learning rate.

**If you pick a learning rate that is too small, learning will take too long:**

**Diagram

Description automatically generated**

Conversely, if you specify a learning rate that is too large, the next point will perpetually bounce haphazardly across the bottom of the well like a quantum mechanics experiment gone horribly wrong:

Diagram

Description automatically generated

There's a [Goldilocks](https://wikipedia.org/wiki/Goldilocks_principle) learning rate for every regression problem.

The Goldilocks value is related to how flat the loss function is.

If you know the gradient of the loss function is small then you can safely try a larger learning rate, which compensates for the small gradient and results in a larger step size.

Diagram

Description automatically generated

**Key Terms**

|  |  |
| --- | --- |
|  [hyperparameter](https://developers.google.com/machine-learning/glossary#hyperparameter) |  [learning rate](https://developers.google.com/machine-learning/glossary#learning_rate) |
|  [step size](https://developers.google.com/machine-learning/glossary#step_size)  **NOTE:** In practice, finding a "perfect" (or near-perfect) learning rate is not essential for successful model training. The goal is to find a learning rate large enough that gradient descent converges efficiently, but not so large that it never converges. |  |

**SECTION 7**

**Reducing Loss: Stochastic Gradient Descent**

In gradient descent, a **batch** is the **total number of** **examples you use to calculate the gradient in a single iteration.**

So far, we've assumed that the batch has been the entire data set. When working at Google scale, data sets often contain billions or even hundreds of billions of examples.

Furthermore, Google data sets often contain huge numbers of features. Consequently, a batch can be enormous. A very large batch may cause even a single iteration to take a very long time to compute.

**A large data set with randomly sampled examples probably contains redundant data.**

**In fact, redundancy becomes more likely as the batch size grows. Some redundancy can be useful to smooth out noisy gradients, but enormous batches tend not to carry much more predictive value than large batches.**

What if we could get the right gradient on average for much less computation? By choosing examples at random from our data set, we could estimate (albeit, noisily) a big average from a much smaller one.

**Stochastic gradient descent (SGD)** takes this idea to the extreme--it uses only a single example (a batch size of 1) per iteration. Given enough iterations, SGD works but is very noisy. The term "stochastic" indicates that the one example comprising each batch is chosen at random.

**Mini-batch stochastic gradient descent (mini-batch SGD**) is a compromise between full-batch iteration and SGD. A mini-batch is typically between 10 and 1,000 examples, chosen at random. Mini-batch SGD reduces the amount of noise in SGD but is still more efficient than full-batch.

To simplify the explanation, we focused on gradient descent for a single feature. Rest assured that gradient descent also works on feature sets that contain multiple features.

**Key Terms**

|  |  |
| --- | --- |
|  [batch](https://developers.google.com/machine-learning/glossary#batch) |  [batch size](https://developers.google.com/machine-learning/glossary#batch_size) |
|  [mini-batch](https://developers.google.com/machine-learning/glossary#mini-batch) |  [stochastic gradient descent (SGD)](https://developers.google.com/machine-learning/glossary#SGD) |

**SECTION 8**