The ‘Ingredients’ of Machine Learning Algorithms

The components that most machine learning algorithms have in common.

What’s a cost function, optimization, a model, or an algorithm? The esoteric nuances of machine learning algorithms and terminology can easily overwhelm the machine learning novice.

As I was reading the *Deep Learning* book by Yoshua Bengio, Aaron Courville, and Ian Goodfellow, I was ecstatic when I reached the section that explained the common “recipe” that almost all machine learning algorithms share — a dataset, a cost function, an optimization procedure, and a model.

In this article, I summarize each universal ‘ingredient’ of machine learning algorithms by dissecting them into their simplest components.

With these ‘ingredients’ in mind, you no longer have to view each new machine learning algorithm you encounter as an entity isolated from the others, but rather a unique combination of the four common elements described below.

There are many types of machine learning algorithms. we will use the **Linear Regression Algorithm** to learn about each of the four components.

1. Specification of Dataset

The first component of a machine learning model is the **dataset.**

Machine learning, as a type of applied statistics, is built on large quantities of **data.** As a result, your choice of *data features*, important data fed as input, can significantly influence the performance of your algorithm.

The art of choosing data features is so important that it has its own term: *feature engineering.*

Feature Engineering: What Powers Machine Learning

Extracting Features for Supervised Learning

Common Examples

- An X and y (an input and expected output) → **Supervised Learning**
- An X (only input) → **Unsupervised Learning**

A dataset of a *simple linear regression algorithm* could look like this:

x	y
1	6
2	4
3	2

In the **Linear Regression** example**,** our specified dataset would be our **X** values, and our **y** values (the predictors, and the observed data).

2. Model

The **model** can be thought of as the primary *function* that accepts your **X** (input) and returns your y-hat (predicted output).

Although your model may not always be a function in the traditional mathematical sense, it is very intuitive to think of a model as a function because, given some input, the model will *do something with the input* to perform the Task (T).

Common Examples

- Multi-Layer Perceptron (Basic Neural Network)
- Decision Tree
- K-Means (Clustering)

In the context of a *simple linear regression*, the **model** is:

$y = mx + b$

where y is the predicted output, x is the input, and **m and b are model parameters.**

Every model has **parameters,** *variables that help define a unique model, and whose values are estimated as a result of learning from data.* For instance, if we had the following simple dataset from section 1,

x	y
1	6
2	4
3	2

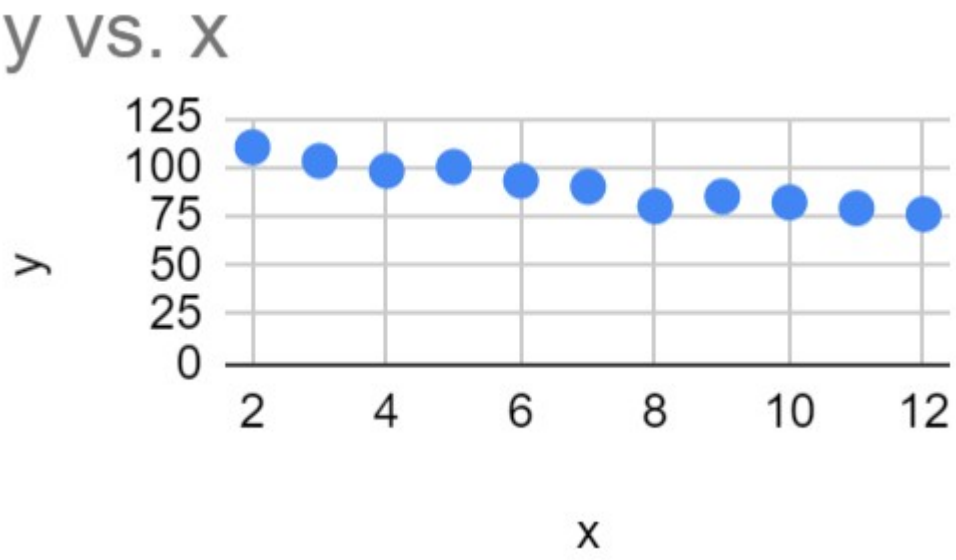
our optimal m and b in our linear model would be -2 and 8 respectively, to have a fitted model of y = -2x + 8. The specific values, -2 and 8 make our linear model unique to this dataset.

Since our dataset is relatively simple, it is easy to determine the parameter values that would result in a model that minimizes error (in this case, the ‘predicted’ value is = to the ‘actual value’).

Consider the dataset like the one below:

x	y
1	108
2	110
3	103
4	98
5	100
6	93
7	90
8	80
9	85
10	82
11	79
12	76

The graph of Fig 2.0 is displayed below.



Notice that finding the optimal m and b is no longer as straightforward as the previous example. In this case, we would have to *estimate* the best model parameters, m and b, that fit the data by optimizing a cost function.

3. Cost Function

What is a Cost Function?

The next universal component is the **cost function** or **loss function**, usually denoted as J(Θ).

A machine learning algorithm must have some cost function that, when optimized, makes the predictions of the ML algorithm *estimate* the *actual* values to the best of its ability. The optimization of the cost function is the process of learning.

In the most basic sense, a cost function is some function that measures the difference between the **observed/actual values** and the **predicted values** based on the model.

This makes intuitive sense. If our function measures some distance between the observed and predicted values, then, if minimized, the difference between observed and predicted will steadily decrease as the model learns, meaning that our algorithm's prediction is becoming a better estimate of the actual value.

Not all cost functions are able to be easily evaluated. However, we may use iterative numerical optimization (see Optimization Procedure) to optimize it.

There are common cost functions for each type of Task (T).

Common Examples

- Quadratic Cost Function (Classification, Regression) *not used frequently in practice, but excellent function to understand concept
- Cross-Entropy Cost Function a.k.a. Negative-log Likelihood (see the link below for more information on negative-log likelihood and maximum likelihood estimation).

- Sum of Squared Residuals between datapoint and centroid (K-means Clustering)
- Dice Loss (Segmentation)

Maximum Likelihood Estimation

Fundamentals of Machine Learning (Part 2)

In our *linear regression example*, our *cost function* can be the mean squared error:

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

This cost function measures the difference between the actual data (yi) and the values predicted by the model (mxi + b). We square this difference, and take the mean over the dataset by dividing by the number of data points. We can now use an *optimization procedure* to find the m and b that minimize the cost.

4. Optimization Procedure

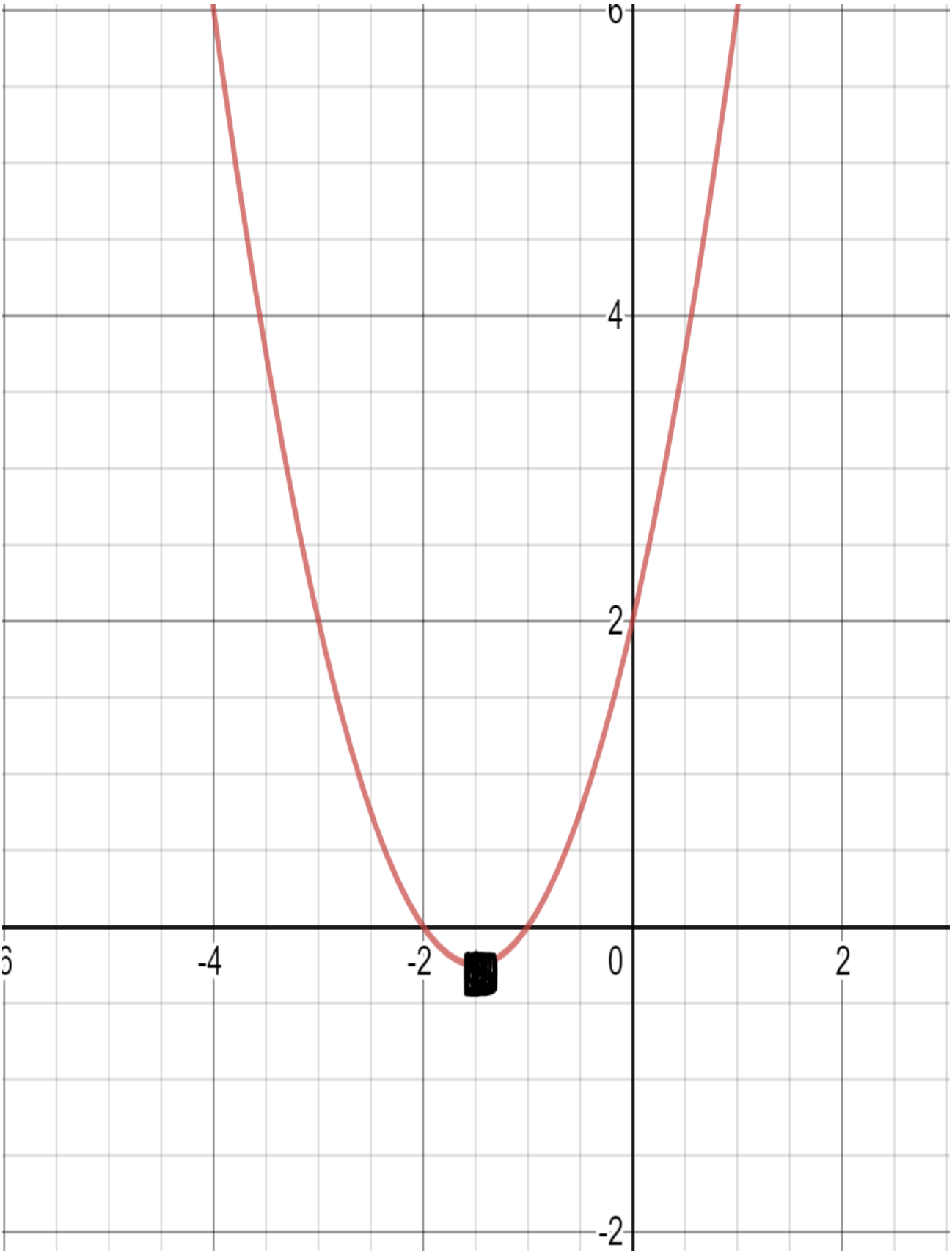
Next is the optimization procedure, or the method that is used to minimize or maximize our cost function with respect to our model parameters. Through this optimization procedure, we are estimating the *model parameters* that make our model perform better.

There are two main forms of optimization procedures:

Closed-Form Optimization

A function can be optimized in *closed-form* if we can find the *exact* minima (or maxima) using a finite number of ‘operations’.

A very simple example only requires high-school calculus.


6
4
2
-2
6
-4
-2
0
2

If you have the function, $J(w) = w^2 + 3w + 2$ (shown above), then you can find the *exact minima of this function* with respect to w by taking the derivative of f(w), and setting it equal to 0 (which are a finite number of operations).

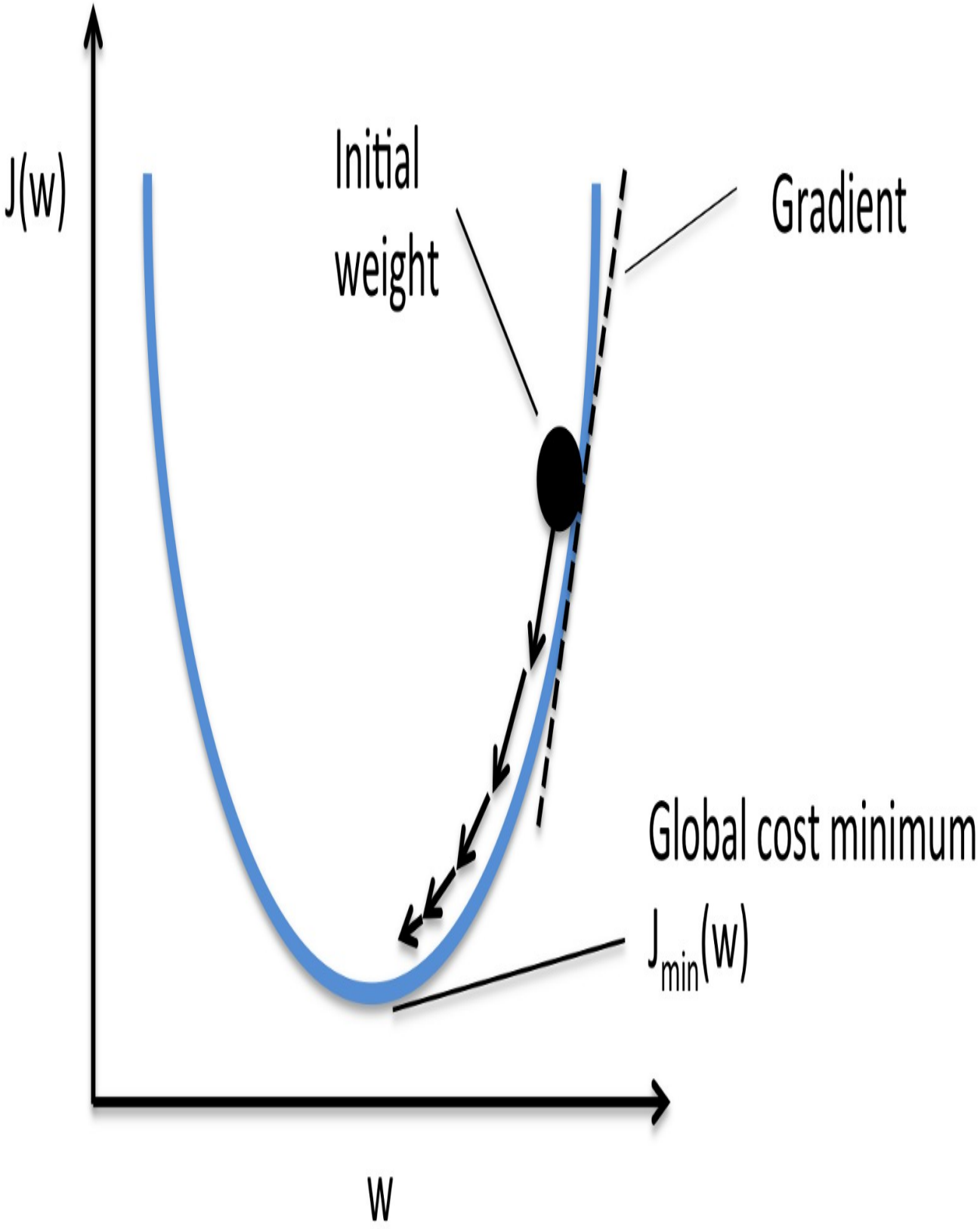
$2w + 3 = 0 \rightarrow w = -3/2$

$f(-3/2) =$ **-1/4**.

Iterative Numerical Optimization

Iterative numerical optimization is a technique that *estimates* the optima.

It is the most common optimization procedure because it often has a lower computational cost than closed-form optimization methods. For this reason, many algorithms will trade 100% accuracy for faster, more efficient estimations of the minima or maxima. Furthermore, many cost functions do not have a closed-form solution!



— Python Machine Learning, 2nd Edition

Using the same example from *closed-form optimization*, we can imagine we are trying to optimize the function $J(w) = w^2 + 3w + 2$. We can imagine choosing a random point on this graph (the model parameters are randomly initialized, so the initial 'prediction' is random, and the initial value of the function is therefore random).

In this case, we can use Stochastic Gradient Descent. See the following articles for more on SGD:

Understanding Gradient Descent

The fundamentals of this critical data science tool

Stochastic Gradient Descent — Clearly Explained !!

Stochastic gradient descent is a very popular and common algorithm used in various Machine Learning algorithms, most

It is best to think of this type of iterative optimization as a ball rolling down a hill/valley, as can be visualized in the image above.

Common Examples

- Stochastic Gradient Descent (SGD) → I.N.O.
- Adam (Adaptive Moment Estimation) → I.N.O.

According to the *Deep Learning* book, "other algorithms such as decision trees and k-means require *special-case optimizers* because their cost functions have flat regions… that are inappropriate for minimization by gradient-based optimizers."

In our *linear regression example*, we could apply SGD to our MSE cost function in order to find the optimal m and b.

Our algorithm would calculate the *gradient of the MSE* with respect to *m and b*, and iteratively update m and b until our model's performance has converged, or until it has reached a threshold of our choosing.

This is analogous to calculating the derivative of our J(w) function shown in Fig 4.1, and moving w in the *opposite direction of the sign of the derivative*, bringing us closer to the minima. (slope is positive, w becomes more negative)

*Note on Backpropagation

Many have heard of the term **backpropagation** in the context of deep learning. A common misconception is that backpropagation itself is what makes the model learn. This is **not** the case. Backpropagation is **not** the optimization procedure.

So where does backpropagation fit into the picture?

Backpropagation is used as a **step** in the optimization procedure of *Stochastic Gradient Descent*. To be more precise, it is the technique used to **estimate the gradients of the cost function with respect to the model parameters.**

5. Conclusion

In this article, we've dissected the machine learning algorithm into common components.

I hope you find comfort in the fact that most machine learning algorithms can be broken down into a common set of components. We can now view 'new' machine learning algorithms as mere *variations* or *combinations* of the 'recipe', as opposed to an entirely new concept.

With that said, don't be afraid to tackle new ML algorithms, and perhaps experiment with your own unique combinations.