Simple Linear Regression: Part 1

Jay Lee

Machine Learning Engineer, Deeping Source Inc.

CodeSeoul MLA
January 14th, 2022

Overview

Disclaimer

I am not an expert nor do I claim to be an expert in this field. I have not written or published a book on machine learning. These contents are written merely from personal experiences and self-study.

With that aside, hope that people get something out of today's talk.

What is Machine Learning?

Machine learning is a field of algorithms that are capable of learning from a dataset by solving an optimization problem. It generally involves finding a set of parameters $\Theta_* = \{\theta_1, \theta_2, ..., \theta_m\}$ that best explains the target dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$

Some examples of machine learning tasks involve learning:

- A mapping function. E.g. classification, regression. Learn $f(x) \mapsto y$
- Representations (learning the features to perform a specific task. E.G. Deep Learning)

What is supervised learning?

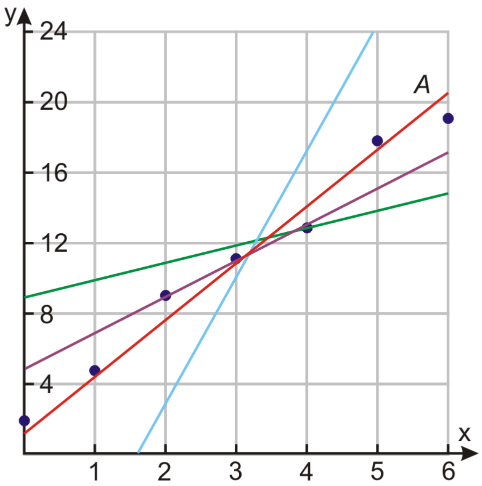
Formal problem definition: Given a labeled dataset $X = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, learn a mapping function $F(x) \mapsto \hat{y}$.

Main idea: learn from a **labeled dataset**. Examples of supervised learning tasks include.

- Classification. E.g. given a dataset of cats a dog, train a model to accurately predict whether a given image is a cat or a dog
- Regression. E.g. Given a dataset of consisting of square footing and pricing of houses LA, learn to predict house prices in LA.

What is regression in machine learning?

Regression involves predicting continuous values (think of floating point integers).



Why Linear Regression

- It is simple
  - Great way to obtain an intuitive understanding of how machine learning works.
  - Simple models are easy to understand, interpret and visualize
- It is fast!
  - $O(1)$ for inference $(y = \theta_1 x + \theta_0)$.
  - Linear models can be fit quickly to check whether the independent variable(s) - x has a linear relationship with dependent variable - y.

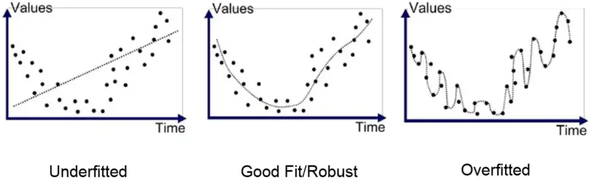
Occam's razor

In plain English, given a set of theories and explanations, the simplest one should be preferred.

For example, if we can solve a problem using a linear model vs a highly parameterized neural network, the linear model should be preferred.

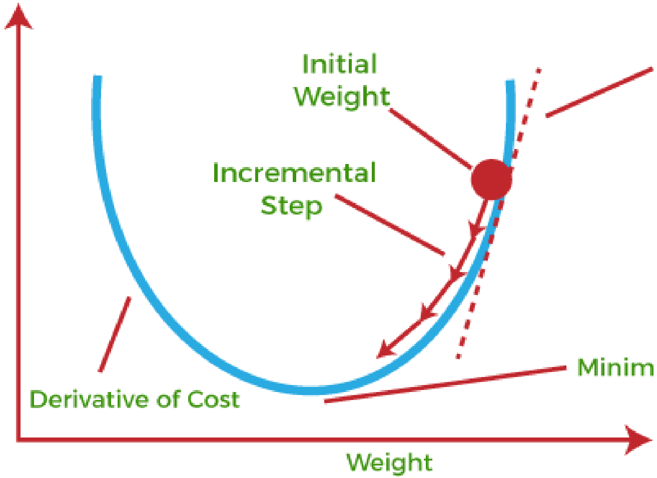
General rule of thumb.

- If two models have the same performance on the validation and test set, prefer the simpler one.
- Simpler models generalize better



Gradient Descent

An optimization algorithm for finding the local minima of a differentiable loss function.



Gradient Descent Part II

Think of a large ball rolling down the hill. The algorithm will find the bottom of the hill that you are currently on. However, if you are on a large mountain with many hills, there is no guarantee that you will reach the bottom.

Sample youtube videos of Gradient Descent (GD) animation:

- GD on linear regression
- Another GD animation

Gradient Descent Part III

Great! Now we know the intuition behind how it works, what is the actual equation behind the optimization?

$$\theta = \theta - \eta \frac{\delta L}{\delta \theta} \tag{1}$$

Notations:

- $\theta =$ parameter
- $\eta =$ Learning Rate
- $\frac{\delta L}{\delta \theta} =$ Derivative of the loss function W.R.T the parameter

Learning rate

In gradient descent, we saw a ball rolling down a slope / hill. At each step the ball moves. In practice, how much should the ball move by? The amount by which the ball moves is dictated by the **learning rate**.

These arbitrary values that are not "learnable", but are used to control the learning process are called **hyper-parameters** in machine learning. Learning rate is one of the most important hyper-parameters to consider during training:

- A large learning rate will result in faster convergence. However, towards the bottom of the slope, we might overshoot the target, resulting in a "pendulum" effect.
- Setting a small learning rate may result in more stable training. However, since update steps are smaller, it will take longer to train the model.

What is a Loss Function

A loss function is a differentiable function that provides the model with feedback on how well it is doing.
For a loss function to be effective, it must be

- Differentiable: we need to be able to calculate the derivative to perform gradient descent.
- In line with the end user's objective(s).
  - How many "outliers" exist in the training set? Do we want to penalize outliers?
  - What kind of errors do we want our model to minimize? Do we want to heavily penalize false positives?
  - What does the distribution of our dataset look like?
  - What kind of patterns do we want our model to learn?

Mean Squared Error

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

```
np.mean(np.square(y - y_pred))
```

Loss function for minimizing the linear regression models. Is also used for minimizing the L2 distance for training Autoencoders in Deep Learning. Key features:

- Penalizes high loss due to square components, but laxes on lower loss (intuitively, squared numbers < 1 results in smaller numbers)
- MSE is more sensitive to outliers, since loss on outliers will be much higher. This can be a good thing if we want to clearly catch outliers in the dataset.

Mean Absolute Error

$$MAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y_i}|}{N} \tag{3}$$

```
np.mean(np.abs(y - y_pred))
```

Mean absolute error is calculated as the sum of absolute errors divided by the number of samples.
Key features:

- Treats large errors and small errors equally.
- More interpretable because the error value is in the same scale as the target we are aiming to predict

Notations

- N: Size of training set or batch
- x_i: The ith data point in the training set $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$
- $\hat{y}_i$: Prediction made on the ith training data point: x_i
- θ_0: The bias
- θ_1: Weight of corresponding feature

Derivatives

Going back to high-school calculus, the derivative of a function represents how much a function's output will change W.R.T its inputs.

- If we are trying to predict the height of a person, it can explain how a change in the age value will impact the final output.
- Remember, our model is $\hat{y} = x\theta_1 + x\theta_0$
- To find the minima, we need to find the rate of change W.R.T each of our parameters: θ_1 and θ_0 by taking partial derivatives and setting it to 0.

Deriving MSE Loss: Part One

Our loss function is defined as follows:

$$Error = \frac{\Sigma_{i=1}^{N}(y_i - \hat{y}_i)^2}{2N} = \frac{\Sigma_{i=1}^{N}(y_i - \theta_0 - \theta_1 x_i)^2}{2N} \tag{4}$$

Calculate partial derivatives W.R.T θ_0 and set it to zero to get value of the minima:

$$\begin{aligned} \frac{\delta L}{\delta \theta_0} &= \frac{\Sigma_{i=1}^{N}(y_i - \theta_0 - \theta_1 x_i) * (-1)}{N} = 0 \\ 0 &= \frac{\Sigma_{i=1}^{N}(\theta_0 + \theta_1 x_i - y_i)}{N} \\ 0 &= \frac{\Sigma_{i=1}^{N}(\hat{y}_i - y_i)}{N} = -\frac{\Sigma_{i=1}^{N}(y_i - \hat{y}_i)}{N} \end{aligned} \tag{5}$$

Deriving MSE Loss Part Two

Find partial derivative of the loss W.R.T parameter θ_1

$$\begin{aligned}
\frac{\delta L}{\delta \theta_1} &= \frac{\Sigma_{i=1}^{N}(y_i - \theta_0 - \theta_1 x_i)(-x_i)}{N} = 0 \\
0 &= x_i \frac{\Sigma_{i=1}^{N}(\theta_0 + \theta_1 x_i - y_i)}{N} \\
0 &= x_i \frac{\Sigma_{i=1}^{N}(\hat{y}_i - y_i)}{N} = -\frac{\Sigma_{i=1}^{N} x_i (y_i - \hat{y}_i)}{N}
\end{aligned} \tag{6}$$

Implementing Linear Regression

Great! we went through all the theory. You should have enough information to implement Linear Regression. We will be implementing it from scratch.

Things to know in numpy

- $np.sum$: Calculate the sum of a vector / matrix
- $np.mean$: Calculate the mean of a vector / matrix
- $np.dot$: Dot product of two arrays / tensors

GitHub Link: `https://github.com/CodeSeoul/machine-learning/`