Machine Learning - Week 1

ZVENC

December 2024

1 Introduction

There are many problems that have been solved using rule-based algorithms, such as finding the shortest path between two points, sorting lists of items, etc. There are problems that cannot be solved this way because they are simply too complex; this is a problem area where machine learning excels. In simple terms, machine learning is a field concerned with teaching computers how to learn to do certain tasks. Machine learning is heavily relevant today, being used in various fields such as healthcare, scientific research, social media, manufacturing, etc.

1.1 Types of Machine Learning Approaches

Machine Learning algorithms can be categorized into two main types:

1. Supervised Learning
2. Unsupervised Learning

While there are other types of machine learning, these two are the most relevant at this stage.

1.2 Supervised Learning

This approach to machine learning involves giving the computer a number of examples to learn from in the form of input-output pairs. The output represents what is expected given a particular input. The computer then learns from these input-output pairs and tries to guess the correct output for an input that it has never seen before.

For example let's say you had a plot of house prices against size for about 15 houses. The goal would be to get the computer to estimate the price of a house based on its size, even for houses not explicity represented in the dataset. This is called a *'regression problem'*, and the relationships could either be linear, polynomial, etc.

Another example is breast cancer detection. A model can learn to predict whether a tumor is cancerous based on its size. It can be provided with a plot of patient age against tumor size with the malignant and benign tumors marked on the plot. The model might the attempt to create a boundary to identify malignant tumors. In this example, the model only has to guess between two possible output classes, 'benign' or 'malignant'. This is what's known as a *'classification problem'*. Unlike regression, classification requires the model to predict the correct value from a finite set of classes, not necessarily just two, like in the example. A model could also be designed to detect different classes of malignant tumors, in which case the classes could be 'benign', 'malignant-type-1', and 'malignant-type-2'.

1.3 Unsupervised Learning

With supervised learning, a model learns from data labeled with 'correct answers'. In contrast, unsupervised learning involves providing the model with unlabeled data and tasking it with finding any patterns or structures hidden within the data. Let's say we have another patient age vs tumor size plot except the points aren't labeled. A model might try to group the points into categories or *'clusters'*. This is type of supervised learning algorithm called a *'clustering algorithm'*. Google News employs this algorithm to group related news articles. Achieving this with supervised learning is unfeasible for obvious reasons. Other unsupervised learning algorithms include *'anomaly detection'* and *'dimensionality reduction'*.

1.4 Terminology

1. Training Set: Data used to train the model

2. Input Variable (x): The input provided to the model in the training set. Also called *'feature'* or *'input feature'*.

3. Output Variable (y): The output correct output provided to the model in the training set. Also called the *'target'* variable.

4. Training Example ($x^{(i)}, y^{(i)}$): This is an input output pair from the training set. The superscript 'i' represents the position of the training example in the training set, for instance, in a table.

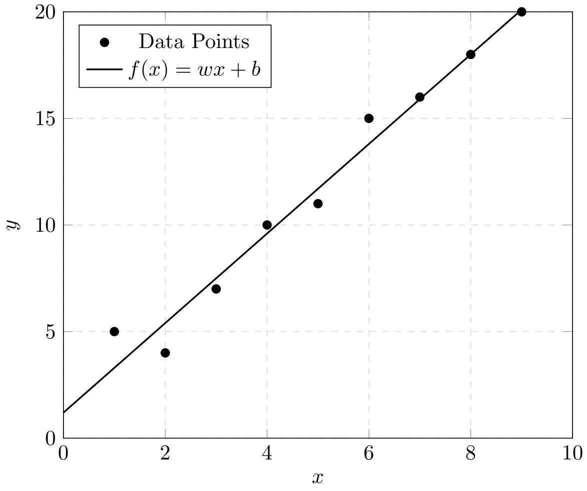
5. Total Number of Training Examples (m): I believe this is self-explanatory.

2 Linear Regression Model

The training set, which contains the input features and output targets, is fed into the supervised learning algorithm. The algorithm then produces a function (historically called a hypothesis) that it then uses to estimate an output for any given input. That function, also called a model, takes in a feature, x, and then produces and estimate, $\hat{y}$. Note the difference between y and $\hat{y}$; y is the target/true value in the training set, while $\hat{y}$ is an estimate of y. A linear regression model takes on the form:

$$f_{w,b}(x) = wx + b$$

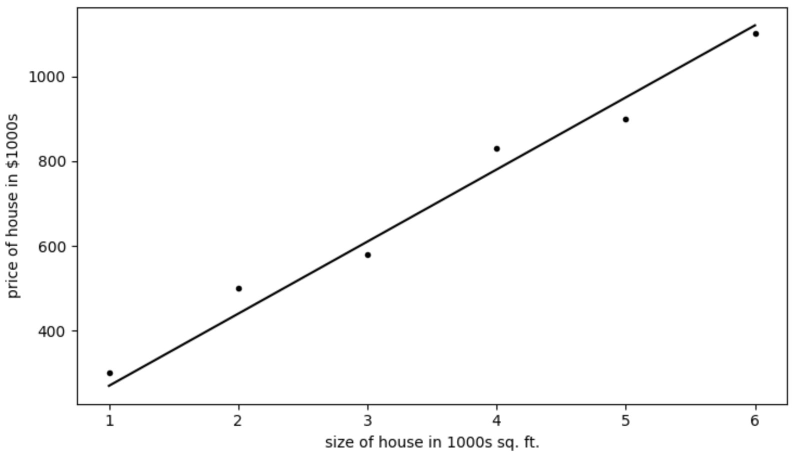
For the sake of convenience, $f_{w,b}(x)$ will be written as $f(x)$. If the features and targets were plot on a graph, the algorithm would generate a best-fit line for the plot which would be represented by that equation.



This is one-variable or *univariate* linear regression, which means there's a single feature for each data point. Using the house price example, it is possible to take other factors into consideration rather than just the size of the house such as number of bedrooms, location, etc.

2.1 Linear Regression Implementation in Python

A linear regression model can be implemented in python using `numpy` and `matplotlib`. The code can be found here. This is a linear regression model for predicting house prices based on size using this training set:



x	y
1.0	300.0
2.0	500.0
3.0	580.0
4.0	830.0
5.0	900.0
6.0	1100.0

House size in $1000ft^2(x)$ and house price in $\$1000(y)$

In the implementation, the features (`x_train`) and targets (`y_train`) are stored in respective `numpy` arrays. The number of training examples is obtained by finding the number of items in the features array, using the array's `.shape` method (it returns a tuple containing the size of the array along each dimension). The `compute_model_output` function creates an array containing the model's estimations of training set's targets, which is used to generate the graph. The weight and bias were initially arbitrarily set but later adjusted to fit the plots on the generated graph. With the accepted weight and bias, the model is used to provide an estimate of the price of a house that is $1200ft^2$, which is $304,000.

3 Cost Function

In order to adjust the parameters of a linear regression model appropriately, it is necessary to have a reliable way to determine the model's performance on the training data. This is acheived through the *cost function.* This function compares the model's predictions to the targets and outputs a number that represents the model's performance. The most commonly used linear regression cost function is called the *Mean Squared Error Cost Function.* Recall that

$$\hat{y} = f_{w,b}(x) \quad \text{and} \quad f_{w,b}(x) = wx + b$$

where $\hat{y}$ is the model's prediction for a given feature. The cost function can then be expressed as

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

where $J(w,b)$ is the cost function, m is the number of training examples, $y^{(i)}$ is a feature, and $\hat{y}^{(i)}$ is a prediction.
The goal is to make the output of the cost function as small as possible

$$\min_{w,b} J(w,b)$$

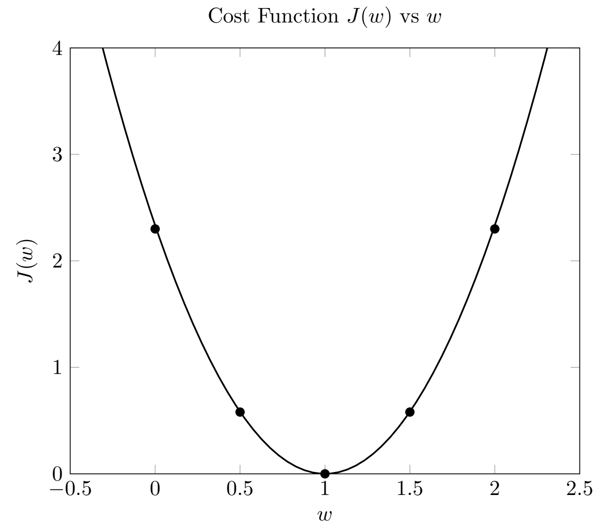
as the size of the output directly reflects the variance between the model's predictions and the target values. Thus, it should be kept to a minimum.

Given $y = x$, if b is set to 0 and J is plot against w for different values of w, observing the resulting parabola might help build some intuition about the behavior of the cost function.
For example, if $w = 1$, $f_w(x) = x$ therefore, $y = f_w(x)$. This implies,

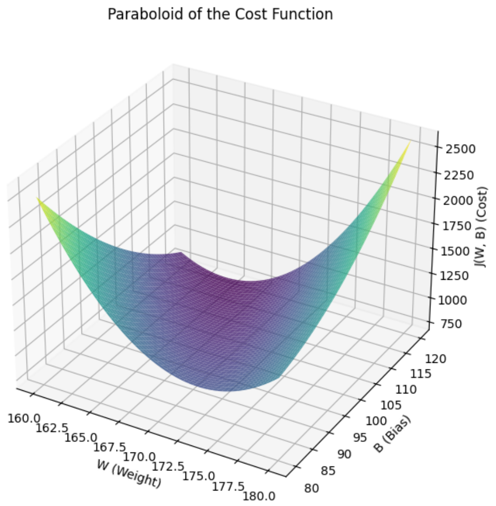
$$\begin{aligned} J(1) &= \frac{1}{2m} \sum_{i=1}^{m} (wx^{(i)} - y^{(i)})^2 \\ J(1) &= \frac{1}{2 \times 3}[(1-1) + (2-2) + (3-3)] \quad = \quad 0 \end{aligned}$$

Repeating this process for more values of w we get:

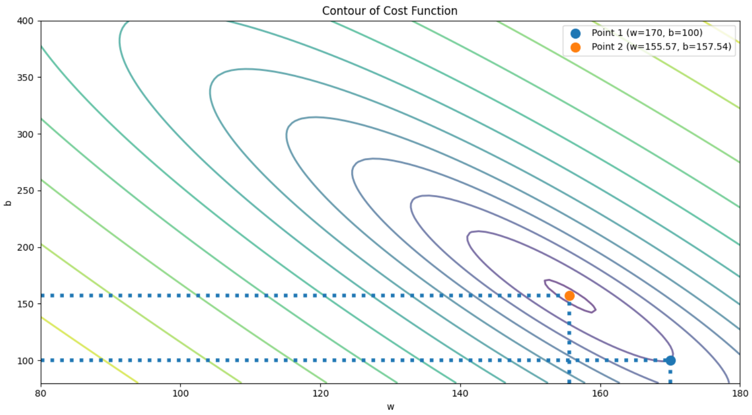
w	$J(w)$
0	≈ 2.3
0.5	≈ 0.58
1.5	≈ 0.58
2	≈ 2.3



The approach taken in the example above was purely demonstrational as the data was prepared specifically for that purpose and would not be helpful when dealing with real-world data. If a different value had been selected for b, the minimum value of the parabola would not have been 0. We would have selected the corresponding value of w while a better value exists although, we had no way of knowing. It is possible that no values of w and b yield a cost function output of zero. This further demonstrates why plotting only 2 variables is not sufficient. The goal remains the same: to minimize the cost function. Values of w and b will be selected to produce the lowest possible cost. But, how are the ideal values determined when a 2-dimensional plot provides a limited understanding of the cost function's behavior? The solution is simply to plot w against b and $J(w, b)$ simultaneously.



The resulting shape is a paraboloid. The minimum value lies at the lowest point of the paraboloid, just as it does with a parabola but this way we have a somewhat complete view of the cost function's behavior. It is still not very convenient to attempt to ascertain what the lowest point is and what the values of w and b are at that point. To address this, the paraboloid can be represented with a contour plot, where the minimum value is the center of the smallest ellipse.



From the contour, we are able to make a reasonably accurate approximation of the ideal parameters for the model. The initial parameter values ($w = 170, b = 100$) are not poor, but they are much less accurate than our contour-based approximations ($w = 155.57, b = 157.54$).

4 Gradient Descent

Gradient descent provides a systematic way of minimizing functions, not just a cost function for linear regression, even for models that have more than two parameters. The gradient descent algorithm takes arbitrarily selected values for w and b, and then changes those values iteratively to reduce $J(w, b)$ until a local minimum is achieved. For a linear regression model, there are no local minima and one global minimum, this means that gradient descent will always reach the minimum. Not all cost functions behave this way. Neural net cost functions for example, are typically non-convex, having many local minima, saddle points and flat regions. Gradient descent is not guaranteed to achieve the global minimum of such functions. The gradient descent function can be represented as:

$$\theta_{new} = \theta_{old} - \alpha \nabla J(\theta_{old})$$

Where:

- θ: The parameters of the model (e.g. w and b)
- α: The learning rate, a hyperparameter that controls the step size (how much the parameters change with each iteration).
- $\nabla J(\theta)$: The gradient of the cost function $J(\theta)$