Linear and Logistic Regression, Decision trees

Swapnoneel Kayal

18 May-24 May 2021

1 Linear Regression

1.1 Introduction and Basic Notations

Linear regression is a supervised machine learning regression algorithm which, given a training dataset along with the corresponding continuous valued labels, determines the straight line function which best fits the training data. Thus, it assumes a linear function for the task and optimizes the function's parameters to fit the training data.

NOTATION

m	the number of instances in the dataset
$\boldsymbol{x^{(i)}}$	the vector of all feature values (excluding the label) of the i^{th} instance in the dataset
$y^{(i)}$	the label (desired/expected output value) of the i^{th} instance in the dataset
$\boldsymbol{x_j}$	the vector of values from all data instances corresponding to the j^{th} feature in the dataset
$x_j^{(i)}$	the value of the j^{th} feature in the i^{th} instance of the dataset
$\boldsymbol{X}$	the entire dataset (excluding labels)
$\boldsymbol{y}$	the vector of all labels for the dataset
θ_j	the parameter of the regression model which corresponds to the j^{th} feature in the dataset
$\boldsymbol{\theta}$	the vector of all parameters of the linear regression model
$h_\theta\left(\boldsymbol{x^{(i)}}\right)$	the predicted label (output value) for the i^{th} training instance

---

1.2 Hypothesis Function

In linear regression tasks, the hypothesis function for an input data instance $\boldsymbol{x}$ having n features and parameter vector $\boldsymbol{\theta}$ is given by:

$$h_\theta(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

Here, x_0 is always equal to 1, and θ_0 is called the *bias term*. It can be thought of as the intercept parameter. Therefore, any linear regression task is effectively a problem of analyzing the given training dataset and figuring out the best possible parameter vector which achieves the maximum accuracy among all possible parameter vectors. This is where the cost function comes in.

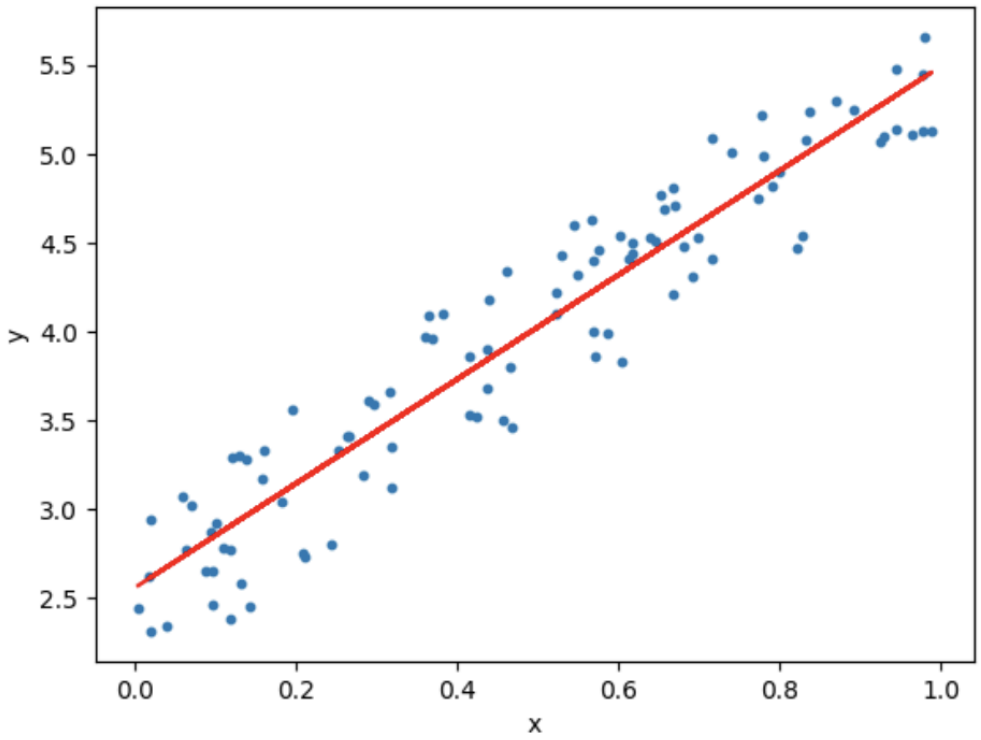


Figure 1: Linear regression

1.3 Cost Function

In a machine learning algorithm, the cost function is used to determine the most accurate set of parameters which fits the training dataset. It provides a measure of the inaccuracy of the machine learning model, and the best possible parameters are determined by minimizing the cost function, using various methods. In linear regression tasks, the most commonly used cost function is:

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(\boldsymbol{x^{(i)}} \right) - y^{(i)} \right)^2 = \frac{1}{2m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^T \boldsymbol{x^{(i)}} - y^{(i)} \right)^2$$

The linear regression model attempts to optimize this cost function, i.e., it calculates the parameters that minimize it. Several methods are available for this optimization, such as batch gradient descent, stochastic gradient descent, mini-batch gradient descent, and the method of normal equations.

1.4 Gradient Descent

The method of gradient descent first assumes a totally random parameter vector, and then moves towards the optimal parameters step by step. During each such step, the gradient of the cost function at the current parameter vector is computed, and a multiple of the gradient is subtracted from the parameter vector, so as to always keep moving along the direction of fastest descent. Note that we have to be careful with this method, since gradient descent moves towards a local minimum, and may never reach the global minimum. The mathematical formulation for each step of gradient descent is as follows:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} - \alpha \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\boldsymbol{\theta}) \end{pmatrix} = \boldsymbol{\theta} - \frac{\alpha}{m} \begin{pmatrix} \sum_{i=1}^{m} \left(h_\theta \left(\boldsymbol{x^{(i)}} \right) - y^{(i)} \right) x_0^{(i)} \\ \sum_{i=1}^{m} \left(h_\theta \left(\boldsymbol{x^{(i)}} \right) - y^{(i)} \right) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} \left(h_\theta \left(\boldsymbol{x^{(i)}} \right) - y^{(i)} \right) x_n^{(i)} \end{pmatrix}$$

$$\implies \boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta \left(\boldsymbol{x^{(i)}} \right) - y^{(i)} \right) \boldsymbol{x^{(i)}} = \boldsymbol{\theta} - \frac{\alpha}{m} \boldsymbol{X}^T \left(\boldsymbol{X\theta} - \boldsymbol{y} \right)$$

This formula is used for a method called batch gradient descent, where the entire dataset is considered in every single iteration, thus slowing down the training. To speed up the process, we can use stochastic gradient descent (one randomly selected instance in every iteration) or mini-batch gradient descent (a randomly selected subset of the dataset in every iteration).

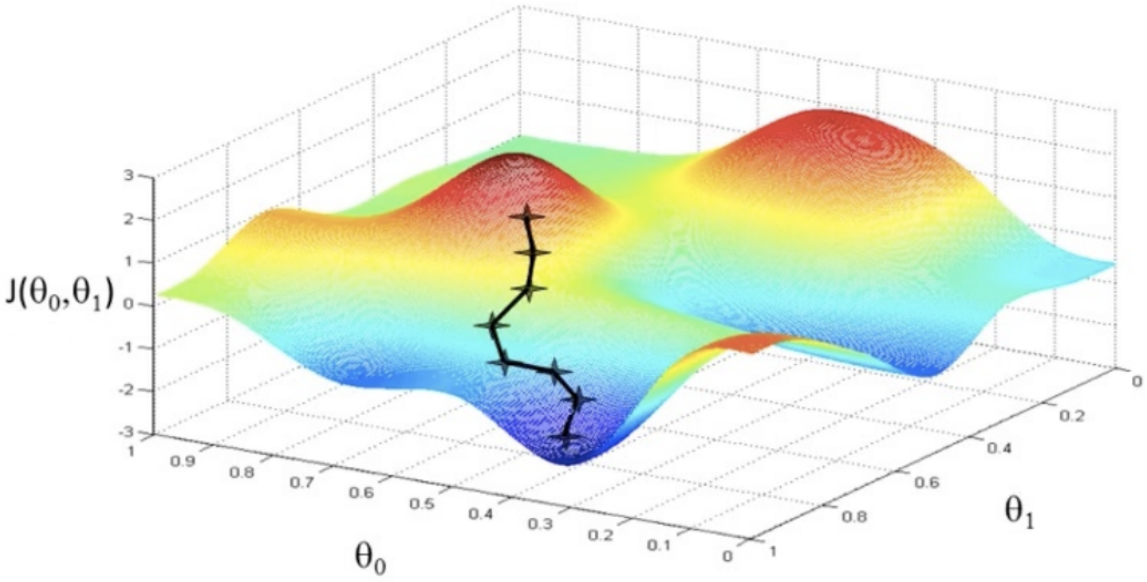


Figure 2: Movement towards lower cost function values using **gradient descent**. For the sake of making visualization possible, this figure considers a dataset consisting of only two features, the corresponding parameters being θ_0 and θ_1.

1.5 Normal Equations

The normal equations method gives us a direct formula to calculate the optimized vector of parameters:

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^T \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

Although this gives us a direct method to obtain the answer in just one step, there is one major demerit. The matrix $\boldsymbol{X}^T \boldsymbol{X}$ is a square matrix having order equal to the number of instances in the dataset, and this makes inverse calculation computationally very expensive for larger datasets. The complexity of matrix inversion algorithms for an $n \times n$ matrix range from nearly $O(n^{2.37})$ to $O(n^3)$, and this large complexity is the main drawback for this method. Thus, gradient descent is preferred for datasets with more than $10,000$ data instances, whereas the normal equations method may be used for smaller datasets.

1.6 Polynomial Regression

Although polynomial regression is different from linear regression, a polynomial regression task can be reduced to a linear regression task by simply adding more features, formed by multiplication of powers of the existing features (according to the degree of polynomial regression), and then applying ordinary linear regression with all the features. Some datasets are better fit by polynomial functions than linear functions, and thus, a higher degree polynomial is often a better solution for the problem, as compared to a linear best fit.

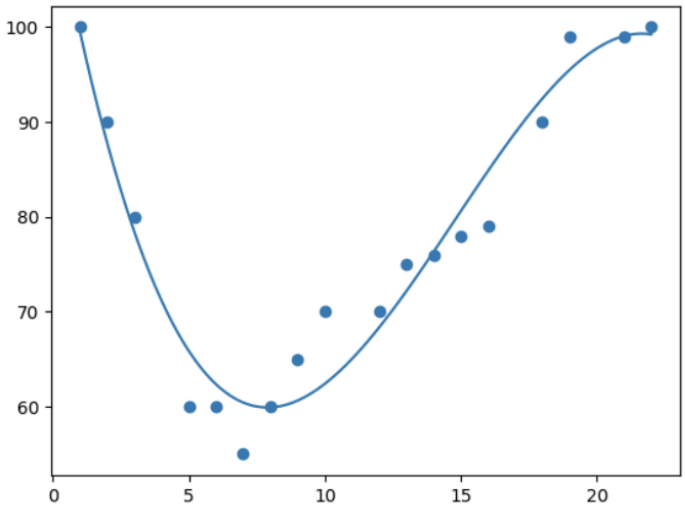


Figure 3: Polynomial regression

2 Logistic Regression

2.1 Introduction

Logistic regression is (quite confusingly, since it has the word *regression*) a classification algorithm, under supervised learning. Given a training dataset along with the corresponding discrete valued labels, it determines a function to fit the dataset. The working principles of this algorithm are somewhat similar to those of linear regression. As for class representation, in a binary classification task, there is one scalar label for each data instance, which contains either 0 or 1, whereas in multiclass classification, the label is expressed as a one-hot encoded vector, wherein the index numbering equal to the correct class number has value 1 and all other indices have value 0. We discuss binary classification first.

2.2 Hypothesis Function

The hypothesis function in this case forces the weighted sum of feature values into the interval $[0, 1]$. This number represents the predicted probability of that instance having a label of 1. The hypothesis function is usually written as:

$$h_\theta(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^T \boldsymbol{x}\right)$$

$$\text{where} \quad g(z) = \frac{1}{1 + e^{-z}}$$

$$\therefore \quad h_\theta(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}$$

Using this probability vector, we predict the label as:

$$y = \begin{cases} 1 & h_\theta(\boldsymbol{x}) \geq 0.5 \\ 0 & h_\theta(\boldsymbol{x}) < 0.5 \end{cases}$$

The above function $g(z)$ is called the sigmoid function. Also, the threshold here is 0.5, but in some special cases, where one out of precision or recall (later) is more important than the other, the threshold value can be changed to obtain better results.

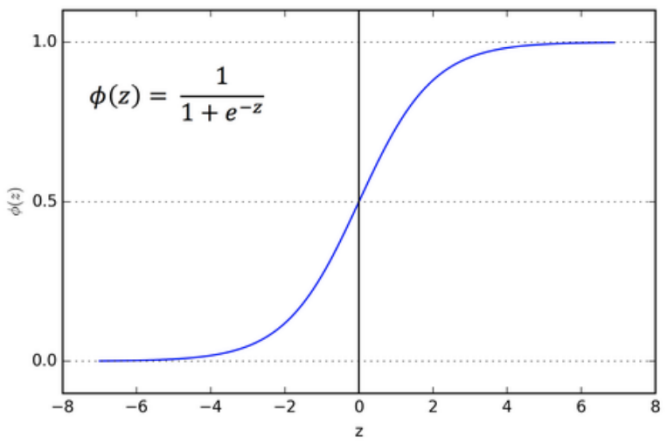


Figure 4: Sigmoid function

2.3 Decision Boundary

It is clear from the graph of the sigmoid function, that it is an increasing function, and it takes a value of 0.5 at $z = 0$. Thus, we predict label:

$$y = \begin{cases} 1 \text{ if } \boldsymbol{\theta}^T \boldsymbol{x} \geq 0 \\ 0 \text{ if } \boldsymbol{\theta}^T \boldsymbol{x} < 0 \end{cases}$$

This allows us to think of the classification in terms of a decision boundary. This boundary has the equation:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = 0$$

The region where the above quantity is positive is the label 1 region, whereas the remaining region is the label 0 region.

2.4 Cost Function

In logistic regression, the cross-entropy cost function is used:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log h_\theta \left(\boldsymbol{x^{(i)}} \right) + \left(1 - y^{(i)} \right) \log \left(1 - h_\theta \left(\boldsymbol{x^{(i)}} \right) \right) \right]$$

It is clear that the above function adds larger errors for larger deviations in the probability from the true labels. As in linear regression, the cost function is minimized using gradient descent (batch/stochastic/mini-batch). There is no normal equations method here.

2.5 Gradient Descent

As in linear regression, gradient descent is implemented by subtracting a multiple of the gradient of the cost function from the parameter vector at each step:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} - \alpha \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\boldsymbol{\theta}) \end{pmatrix} = \boldsymbol{\theta} - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta \left(\boldsymbol{x^{(i)}} \right) - y^{(i)} \right) \boldsymbol{x^{(i)}}$$

As can be seen, the gradient expression turns out to be somewhat similar to that in linear regression.

2.6 Multiclass Classification

For multiclass classification, the one-vs-rest algorithm is used. In this method, the probability that an instance belongs to class i is calculated for each class i, by creating several classifiers (number of classifiers equals the number of classes) and then training each classifier separately. For label prediction, the class label with the maximum probability is returned as the final prediction.

3 Decision trees

3.1 Introduction

A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface.
Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new nodes.
A general algorithm for a decision tree can be described as follows:

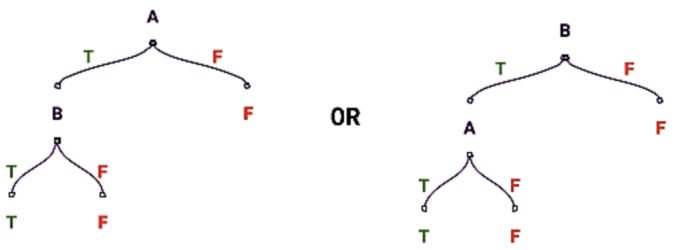
- Pick the best attribute/feature. The best attribute is one which best splits or separates the data.
- Ask the relevant question.
- Follow the answer path.
- Go to step 1 until you arrive to the answer.

The best split is one which separates two different labels into two sets.

3.2 Expressiveness of decision trees

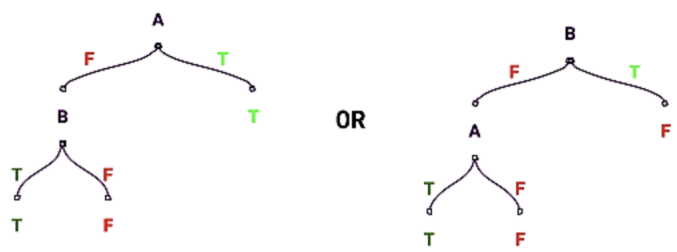
Decision trees can represent any boolean function of the input attributes. Let's use decision trees to perform the function of three boolean gates AND, OR and XOR.
Boolean Function: **AND**

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T



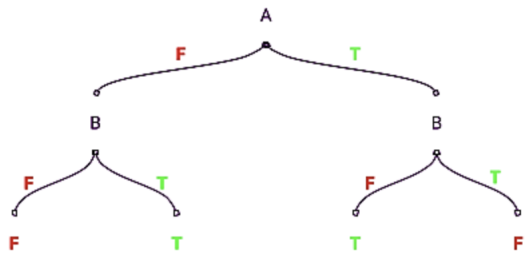
Boolean Function: **OR**

A	B	A OR B
F	F	F
F	T	T
T	F	T
T	T	T

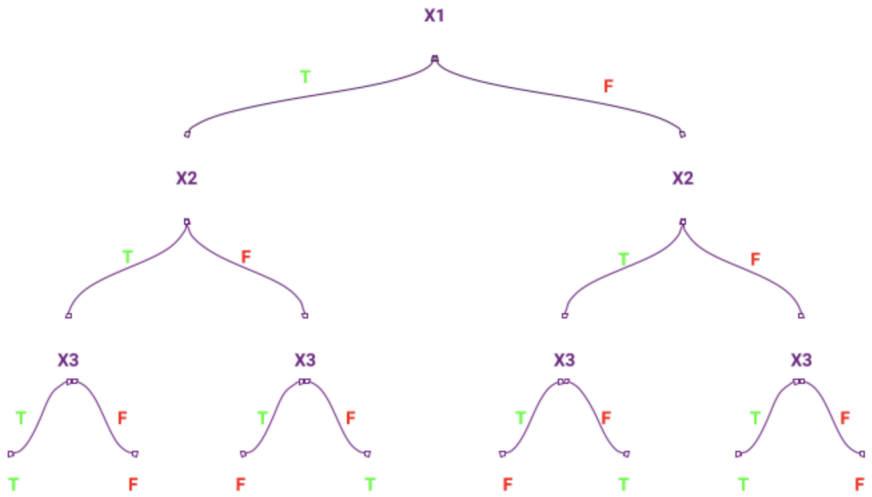


Boolean Function: **XOR**

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F



Let's produce a decision tree performing XOR functionality using 3 attributes:



In the decision tree, shown above, for three attributes there are 7 nodes in the tree, i.e., for $n = 3$, number of nodes $= 2^3 - 1$. Similarly, if we have n attributes, there are 2^n nodes (approx.) in the decision tree. So, the tree requires exponential number of nodes in the worst case.
We can represent boolean operations using decision trees. But, what other kind of functions can we represent and if we search over the various possible decision trees to find the right one, how many decision trees do we have to worry about. Let's answer this question by finding out the possible number of decision trees we can generate given N different attributes (assuming the attributes are boolean). Since a truth table can be transformed into a decision tree, we will form a truth table of N attributes as input.

X1	X2	X3	....	XN	OUTPUT
T	T	T	...	T	
T	T	T	...	F	
...	...	...	...	...	
...	...	...	...	...	
...	...	...	...	...	
F	F	F	...	F	

The above truth table has 2^n rows (i.e. the number of nodes in the decision tree), which represents the possible combinations of the input attributes, and since each node can a hold a binary value, the number of ways to fill the values in the decision tree is 2^{2^n}. Thus, the space of decision trees, i.e, the hypothesis space of the decision tree is very expressive because there are a lot of different functions it can represent. But, it also means one needs to have a clever way to search the best tree among them.

3.3 Decision tree boundary

Decision trees divide the feature space into axis-parallel rectangles or hyperplanes. Let's demonstrate this with help of an example. Let's consider a simple AND operation on two variables(Refer to the table given below for clarity). Assume X and Y to be the coordinates on the x and y axes, respectively, and plot the possible values of X and Y (as seen the table below). The given figures below represent the formation of the decision boundary as each decision is taken. We can see that as each decision is made, the feature space gets divided into smaller rectangles and more data points get correctly classified.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1