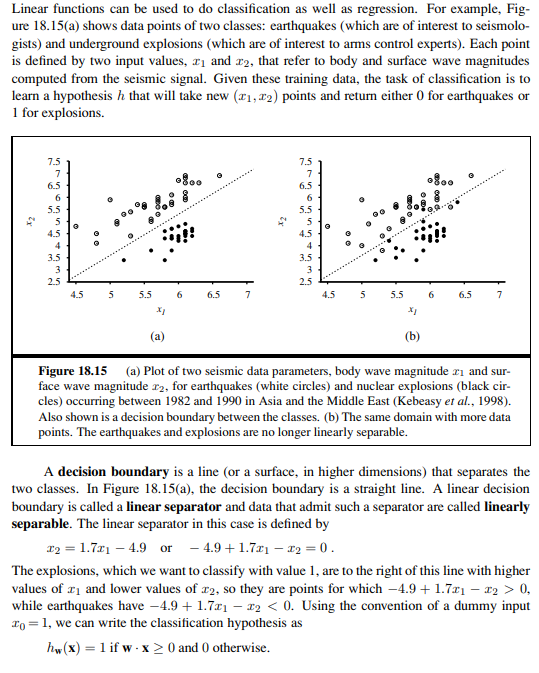
# Linear classification

## Hard Threshold



In other words, given that a set of data points can be divided into two categories through a straight line then we can categorise them. Thus, we can perform a binary classification.

Given that we have a set of data points described by n-tuples (tuples containing n elements) (m1,m2,…, mn) and these data points can be divided into two categories through a hyper-plane (in the case of 2-tuples we obtain a line) then let this hyperplane be equal to :

xn = w0 + w1x1 + … + wn-1xn-1

which we can convert into

w0 + w1x1 + … + wn-1xn-1 + wnxn = 0

Therefore, when the value of w0 + w1x1 + … + wn-1xn-1 + wnxn ≥ 0 output 1 otherwise 0 where 1/0 are the two classifications.

Therefore, in order to build a line classifier all we have to do is to come up with the following function:

h**w**(**x**) = Threshold (**w** · **x**)

where

threshold n | n ≥ 0 = 1

| otherwise = 0

The threshold function can be plotted in the following way:



In order to find out the weight vector **w** so that to minimise the loss and so increase the accuracy of the classification we need the following update rule which is called the **PERCEPTRON LEARNING RULE.**

The perceptron learning rule is applied one example at a time in a similar way as stochastic gradient descent. Indeed, we can say that in order to find the best vector **w** we actually use stochastic gradient descent.

The algorithm that we use is the following:

**w** = select a random weight vector to start

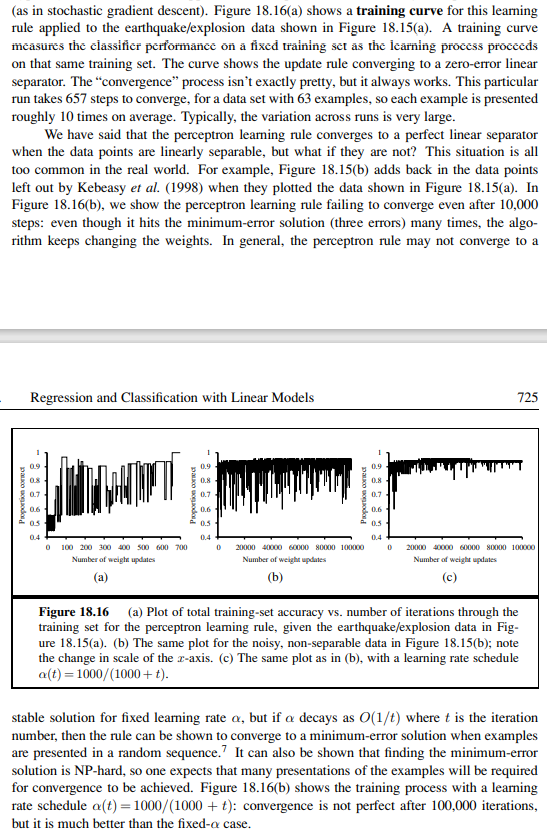
loop until convergence //not guaranteed to converge if no linearly classifiable

loop for each data point (**xn, yn**) in the data set //xn = vector data point, yn = classification

for reach wi in **w**

wi = wi + α (yn – h**w**(**x**n)) xni

Typically the learning rule is applied one example at a time, choosing examples at random (as in stochastic gradient descent).



Therefore, the perceptron learning rule is not guaranteed to converge when the data cannot be linearly classified. It has been shown that the α factor should decay of a 1/t factor where t = iteration number in order to guarantee that we converge to the minimum error solution given that we iterate over the examples in a random way when updating the weights.

The problem of this hard threshold is that if the data points are linearly separable then it is guaranteed to converge and so find the weights **w** so that we classify the data points with 100% accuracy. It is possible to spot this convergence when the weight vector **w** is not updated when performing stochastic gradient descent for all data points.

However, it is not easy to spot when the vector **w** has converged to the best possible vector weight when the data is not linearly classifiable as this kind of hard threshold function is not differentiable and it is discontinuous and thus we cannot find the precise value for **w**.

All this makes learning with the perceptron rule a very unpredictable adventure as we cannot find for what values of **w** we minimise the loss function of this hard threshold as it is not differentiable and it is discontinuous.

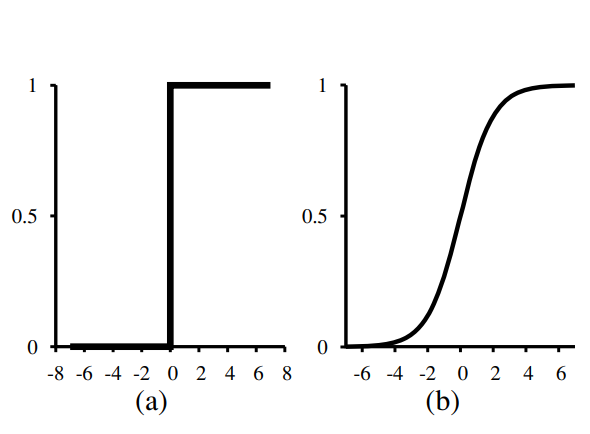
Furthermore, the linear classifier always announces a completely confident prediction of 1 or 0, even for examples that are very close to the boundary; in many situations, we really need more gradated predictions.

## Soft Threshold (Sigmoid function)

The sigmoid function, also called soft threshold, improves on the defect of the Hard threshold function.

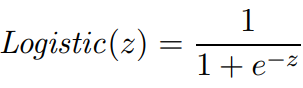
Indeed, instead of outputting either 0 or 1, it outputs a continuous range of values within 0 and 1 excluded. Therefore, the value that outputs the sigmoid function can be interpreted as a probability that a given data point belongs to the category 1. Therefore, a high output value like 0.9 tells us that a given data point has a probability of 0.9 to be in category 1. On the other hand, a low output value like 0.1 tells us that a given data point has a probability of 0.1 to belong to the category 1 and consequently a probability of 0.9 to belong to the category 0.

The sigmoid function achieves this because it a smoother version of the hard threshold function.



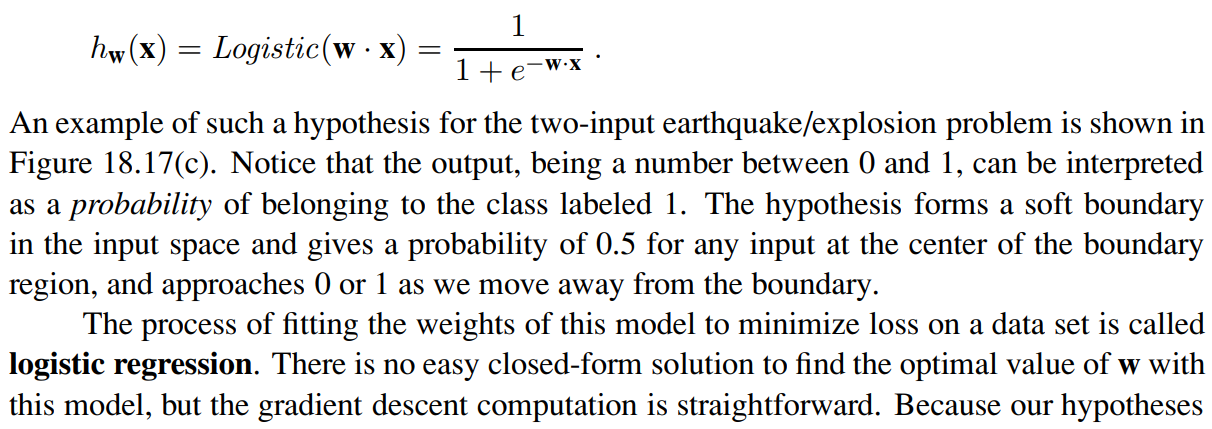
The function (a) is the hard-threshold function while the function (b) is the sigmoid function.

The sigmoid function is also called logistic function and its definition is the following:

A very important improvement of the sigmoid function over the hard threshold is that it is a continuous and differentiable function.

Given that we have introduced the sigmoid function then we can create a linear classifies in the same way as we have done with the hard-threshold.

We are going to call the linear classifies h**w**(**x**).

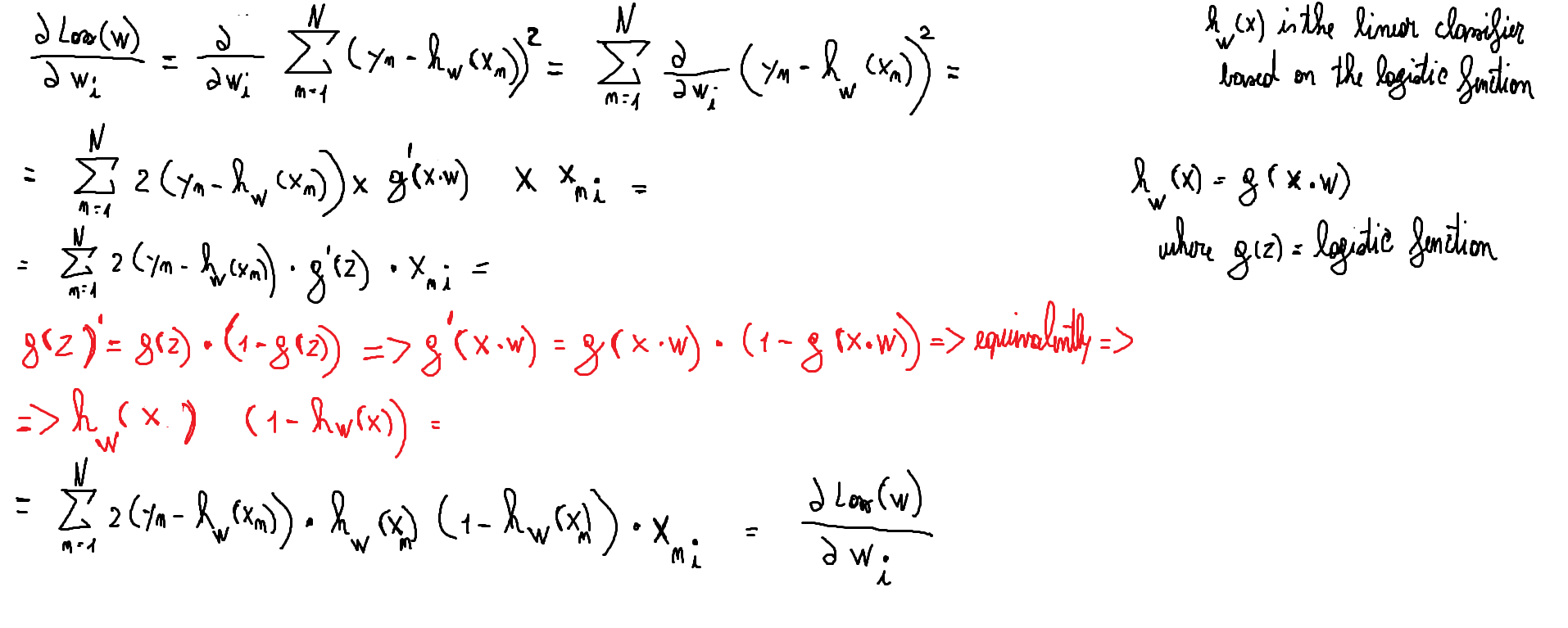


This linear classifier given a certain data point will output the probability that a certain data point belongs to 1. Given that **w**·**x** represents the line separating the two categories then it follows that when a point is on the line then the probability of that point of belonging to 1 must be intuitively 0.5 and this is what occurs with the linear classification based on the logistic function above as when a point lies on the line then -**w**·**x = 0**  and so the value that the logistic function will output is 0.5.

Given that the logistic function is differentiable and continuous and that the linear function **w**·**x** is also differentiable (as it is just a hyperplane) then we can train the model so that to select the best weights **w** by using gradient descent on the loss function. The loss function that we are going to use is the classic average squared loss function that we have used in Linear Regression.

As in linear regression we can use both BATCH GRADIENT DESCENT or STOCHASTIC GRADIENT DESCENT.

Now we are going to see how to find the partial derivative with respect to a weight parameter in the average squared loss function of the linear classifier based on the logistic function.



Given that we already know how to compute both batch gradient descent and stochastic gradient descent then we just need to place in those formulas the above definition of the partial derivative with respect to a weight parameter of the average loss function of the linear classifier based on the sigmoid function.

Given all the advantages that we have outlined so far we can state that logistic regression has become one of the most popular classification techniques for problems in medicine, marketing and survey analysis, credit scoring, public health, and other applications.

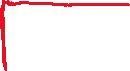
## Another Hard threshold

We have seen so far as hard threshold the function Threshold (n) = 0 if n < 0 or 1 if n ≥ 0. It turns out that there exists another hard threshold function that is called sign which is defined in the following way:

sign n | n ≥ 0 = 1

|otherwise = -1

Its graphic representation is the following:



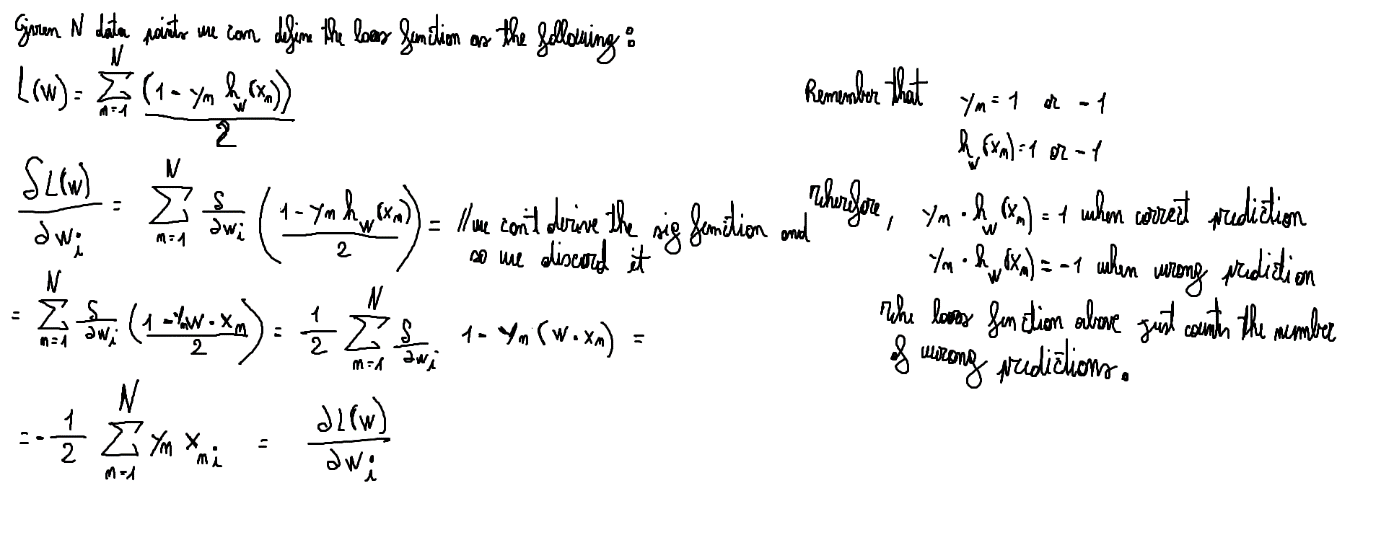
Therefore, we can create a linear classification function h**w**(**x**) = sign (**w**·**x**).

In order to train the function h**w**(**x**) we are going to use the following learning rule:

wi = wi - αxi

This rule can be used in a stochastic gradient descent setting so that to update the weights data point by data point. It is important to notice that this learning rule will always update a weight for a given data point even though that data point is correctly classified. Thus, when we perform stochastic gradient descent for learning a linear classifier involving the sign function then we must firstly check if that data point is at the moment correctly classified. If it is correctly classified then we skip it otherwise we update the weights.

However, we could also use batch gradient descent as we are going to derive the loss function which we will see how to minimise.



The trick in the derivation was to discard the sig function from the definition of h**w**(**x**) in the loss function. We obtain an equivalent minimisation problem because the vector **w** that minimises L(**w**) defined without the sig function will also minimise the real loss function of the function h**w**(**x**).

This follows because if we minimise L(**w**) defined without the sig function then we are trying to make the linear function **w**·**x** output 1 when the real output is 1 and output -1 when the real value is -1. Therefore, we are making the algorithm learn.

As regards the stochastic gradient descent formula outlines in the previous page it omits the value yn. This is ok as yn is just -1 or +1 and so it represents just a constant which we can take into account when choosing the value α for each data point. The value α should take into account also the value -1/2. This is important because otherwise in our gradient descent algorithm we would go towards the maxima instead of the minima.