Machine Learning

Lecture 7

Recall how we do linear regression and fit a linear model to a dataset $\mathcal{D} = (X, \mathbf{y})$ where $X = \{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N\}$ is a set of datavectors i.e. points in some space of d-tuples $\mathbb{R}^d$, so each $\underline{x}_i$ is of the form $\underline{x}_i = (x_i^{(1)}, x_i^{(2)}, x_i^{(3)}, \ldots, x_i^{(d)})$ The set $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$ is the set of responses i.e. for each data vector $\underline{x}_i$ we have a response y_i .

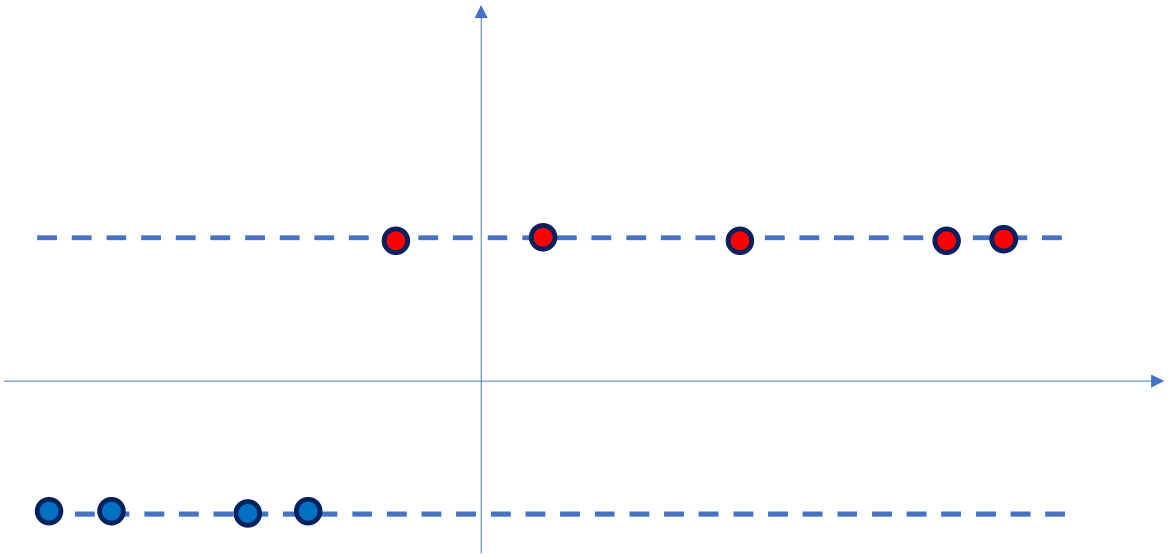
The idea is to find numbers $\beta_0, \beta_1, \beta_2, \ldots, \beta_d$ such that

$$y_i \sim \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)}$$

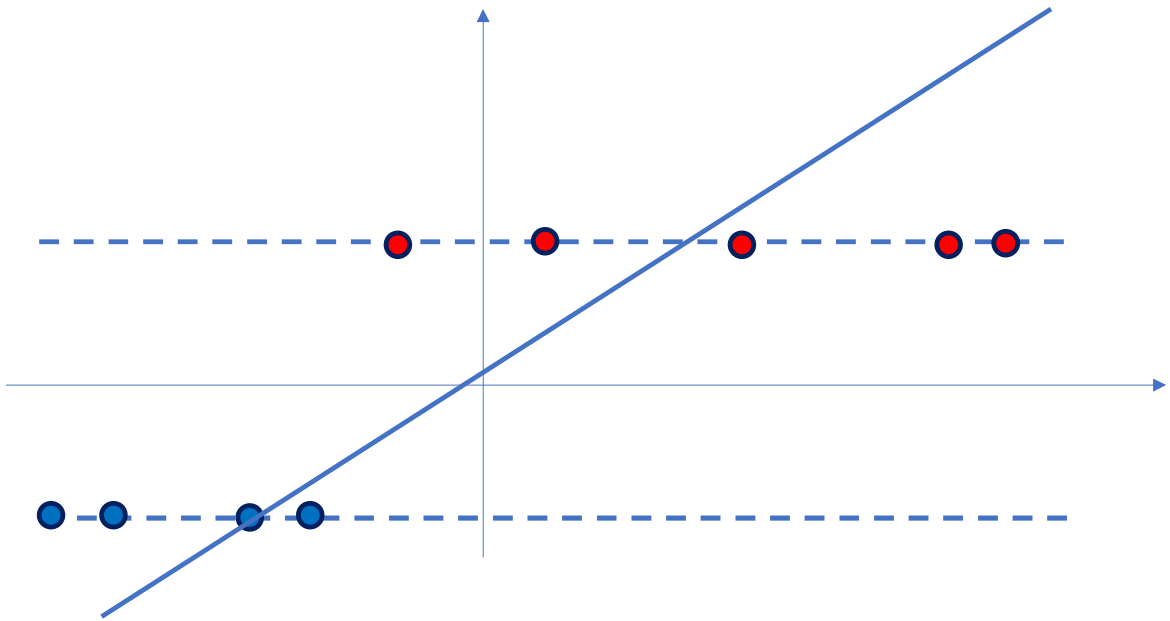
for all $i = 1, 2, 3, \ldots, N$

In the regression problem the y‘s can take any value.

Suppose now that the y‘s can only take the values +1 or -1



It is clearly not possible to fit a line through these points with any kind of precision



We will discuss the *Logistic Regression Model.* Even though the name says regression it is in fact a classification model.

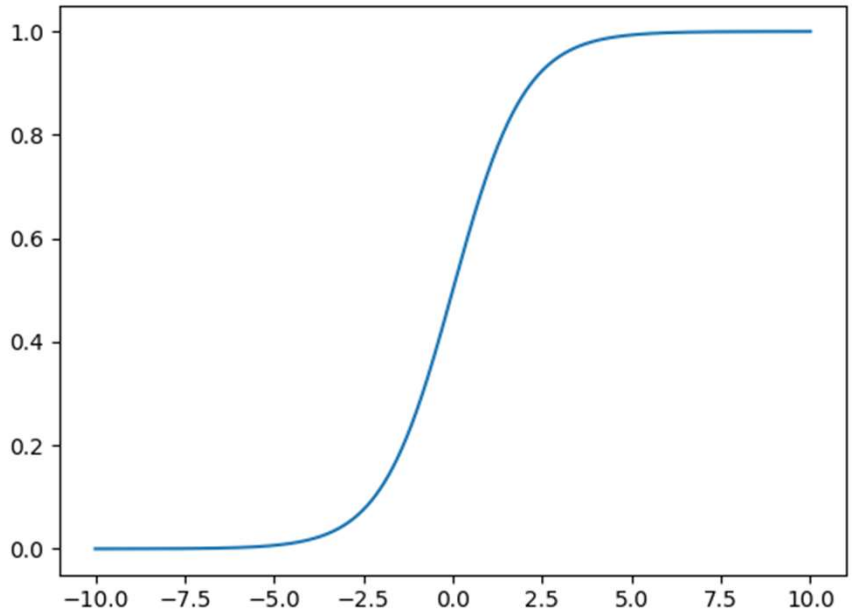
The idea is to assign to each data vector a *probability* i.e. a number between 0 and 1. If this probability is > ½ we associate the label +1 and otherwise -1.

To assign a probability we shall use the logistic function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

```
1  def sigma(z):
2      return 1./(1.+np.exp(-z))
```

executed in 5ms, finished 11:26:13 2019-05-11



So we need to associate to each data vector $\underline{x}$, a real number z .

Compute the value of the logistic function $\sigma(z)$ and check whether this value is > ½ or < ½.

The way we associate the number z is in fact similar to the linear regression, namely we want to find numbers $\beta_0, \beta_1, \beta_2, \ldots, \beta_d$ such that

$$z = \beta_0 + \beta_1 x^{(1)} + \beta_2 x^{(2)} + \cdots + \beta_d x^{(d)}$$

Remark that whether the data vector is classified as +1 or -1 is determined by the sign of z , if: $z > 0$, $\sigma(z) > 1/2$ so the label is +1

The logistic function at a point z defines a probability distribution over the set $\{-1, +1\}$ by

$$P(+1) = \sigma(z), P(-1) = 1 - \sigma(z)$$

Remark that

$$1 - \sigma(z) = 1 - \frac{1}{1 + \exp(-z)} = \frac{1 + \exp(-z) - 1}{1 + \exp(-z)} = \frac{1}{\exp(z) + 1} = \sigma(-z)$$

So

$$P(-1) = \sigma(-z)$$

The idea of logistic regression is to determine the coefficients $\beta_0, \beta_1, \beta_2, \ldots, \beta_d$ so that for all the data points $\underline{x_i}$ with label $y_i = \pm 1$ the probability distribution over $\{-1, +1\}$ defined by

$$\sigma(z = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)})$$

is 'close' to the true distribution

$$Q(y_i) = 1, Q(-y_i) = 0$$

where we measure 'closeness' by the KL divergence

$$D_{KL}(Q||P) = -\mathbb{E}_Q(\log \frac{P}{Q})$$

In our case this is

$$-Q(1)\log\frac{P(1)}{Q(1)} - Q(-1)\log\frac{P(-1)}{Q(-1)} = -Q(1)\log\frac{\sigma(z)}{Q(1)} - Q(-1)\log\frac{\sigma(-z)}{Q(-1)}$$

In case $y = +1$ this is $-\log\sigma(z)$ and if $y = -1$ it is $-\log\sigma(-z)$ and so in any case

$$D_{KL}(Q||P) = -\log\sigma(yz)$$

It follows that we want to find the coefficients $\beta_0, \beta_1, \ldots, \beta_d$ that minimizes

$$\mathcal{L} = -\frac{1}{N}\sum \log\sigma(y_i(\beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)}))$$

To minimize the loss function we can again use gradient descent i.e. we start with some $\underline{\beta} = (\beta_0, \beta_1, \beta_2, \ldots, \beta_d)$ and then we iterate the sequence

$$\underline{\beta}_{new} = \underline{\beta}_{old} - \alpha \nabla_{\underline{\beta}} \mathcal{L}(\underline{\beta}_{old})$$

So we need to compute the gradient i.e. the partial derivatives

$$\frac{\partial}{\partial \beta_k} \mathcal{L} = -\frac{1}{N} \sum_{i=1,2,\ldots,N} \frac{\partial}{\partial \beta_k} \log \sigma(y_i(\beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_k x_i^{(k)} + \cdots + \beta_d x_i^{(d)}))$$

Now

$$\log \sigma(z) = -\log(1 + \exp(-z))$$

and using the chain rule we get

$$\frac{\partial}{\partial \beta_k} \log \sigma(y_i z_i) = \boxed{\frac{\partial}{\partial z_i}(-\log(1 + \exp(-y_i z_i))} \frac{\partial}{\partial \beta_k} z_i$$

The first factor is (again using the chain rule)

$$\begin{aligned} & -\frac{1}{(1 + \exp(-y_i z_i))} \frac{\partial}{\partial z_i}(1 + \exp(-y_i z_i)) \\ &= -\frac{1}{(1 + \exp(-y_i z_i))} \exp(-y_i z_i)(-y_i) \\ &= \boxed{y_i \left(\frac{\exp(-y_i z_i)}{1 + \exp(-y_i z_i)} \right)} \end{aligned}$$

The second factor is much simpler

$$z_i = \beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_i^{(d)}$$

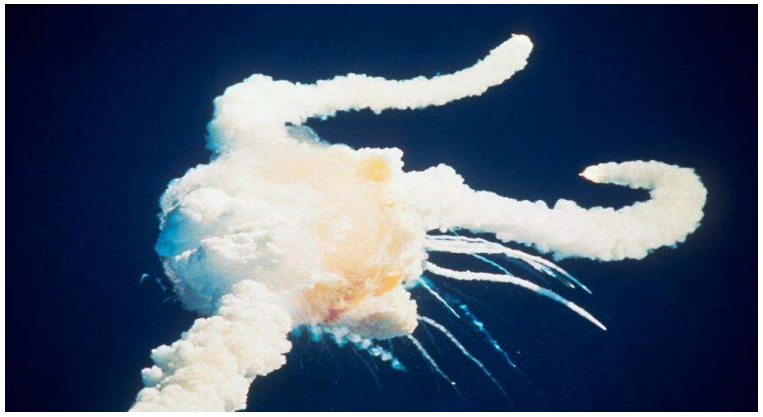
so

$$\frac{\partial}{\partial \beta_k} z_i = \begin{cases} 1 \text{ if } k = 0 \\ x_i^{(k)} \text{ otherwise} \end{cases}$$

finally we get

$$\frac{\partial}{\partial \beta_k} \mathcal{L} = \begin{cases} -\frac{1}{N} \sum_{i=1,2,\ldots,N} y_i \frac{\exp(-y_i z_i)}{1 + \exp(-y_i z_i)} = y_i \sigma(-y_i z_i) \text{ if } k = 0 \\ -\frac{1}{N} \sum_{i=1,2,\ldots,N} y_i \frac{\exp(-y_i z_i)}{1 + \exp(-y_i z_i)} x_i^{(k)} = y_i \sigma(-y_i z_i) x_i^{(k)} \text{ if } k > 0 \end{cases}$$

Challenger Explosion

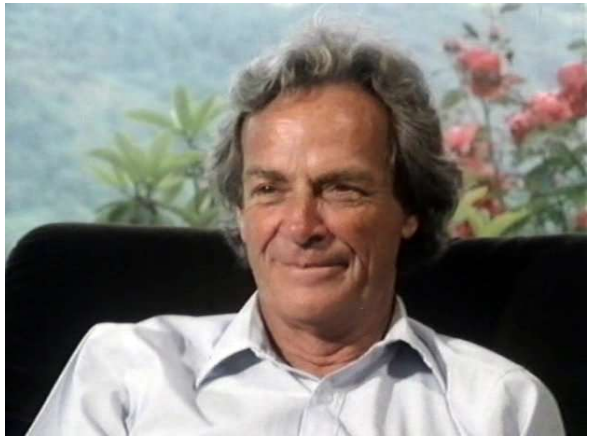


On January 28, 1986 the space shuttle, Challenger, exploded shortly after launch, killing the entire crew. It was later determined that the cause of the disaster was the burn-through of an O-ring seal at a joint in one of the solid-fuel rocket boosters. Of the 24 previous shuttle flights it was determined that in 7 of those flights there was damage to the O-ring seals. In one case the solid fuel booster rockets were not recovered so there are only data for 23 flights. We want to investigate if the temperature at launch time has an influence on whether the O-rings are damaged during the launch.

Table 1: Temperature data from space shuttle launches

Temp	Damage(No = 0,Yes = 1)
66	0
70	1
69	0
68	0
67	0
72	0
73	0
70	0
57	1
63	1
70	1
78	0
67	0
75	0
70	0
81	0
76	0
79	0
75	1
76	0
58	0

During the investigation of the disaster, Richard Feynman, a member of the investigative panel, demonstrated that the O-rings would lose their elasticity under low temperature conditions. The O-rings were designed to expand to completely seal the joints and so the theory was that because of the unusually cold temperatures at launch time of the Challenger mission the O-rings had lost their elasticity and were unable to expand and seal the joints



We are going to use Logistic Regression to investigate whether the data support the theory that cold weather had an impact on damage to the O-ring seals.

We are first going to replace the 0 labels with -1 labels and so we have the dataset

```
import numpy as np
import matplotlib.pyplot as plt
```

```
temp = np.array([66.,70.,69.,68.,67.,72.,73.,70.,57.,
        63.,70.,78.,67.,53.,67.,75.,70.,81.,
        76.,79.,75.,76.,58.])

damage = [-1,1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1]
```

A -1 indicates no damage to the O-rings while a 1 indicates damage

We first write the logistic function and the loss function and the gradient of the loss function

```
1  def sigma(z):
2      return 1./(1.+np.exp(-z))
```

executed in 5ms, finished 11:26:13 2019-05-11

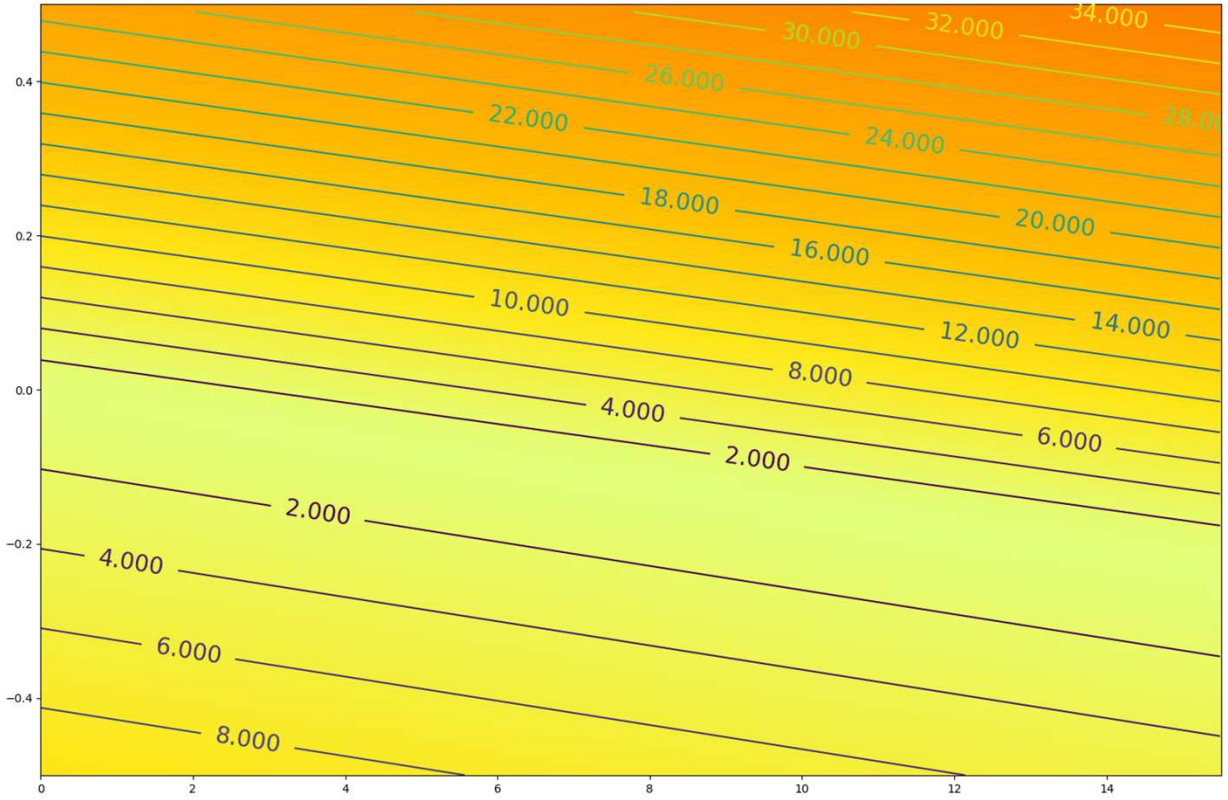
```
1  def loss_function(b0,b1):
2      return np.mean([-np.log(sigma(y*(b0 + b1 * t))) for y,t in zip(damage,temp)])
```

executed in 4ms, finished 11:46:06 2019-05-11

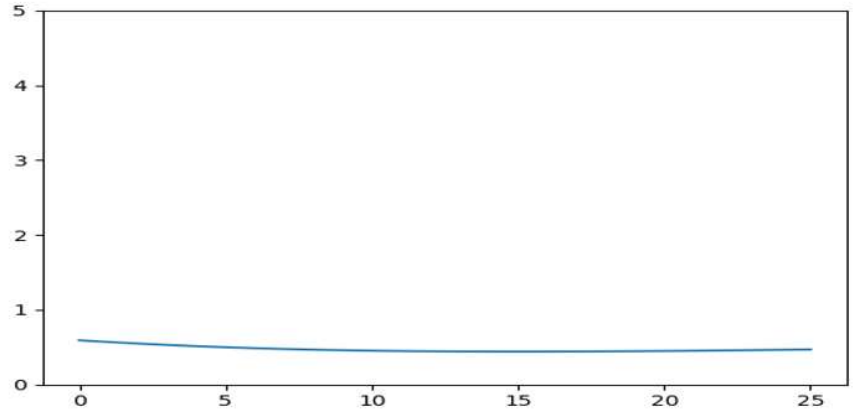
```
1  def gradient(b0,b1):
2      return np.mean([np.array([-y * sigma(-y * (b0 + b1 * t)),
3              -y * sigma(-y * (b0 + b1 * t))*t]) for y,t in zip(damage,temp)],axis=0)
```

executed in 15ms, finished 13:10:41 2019-05-11

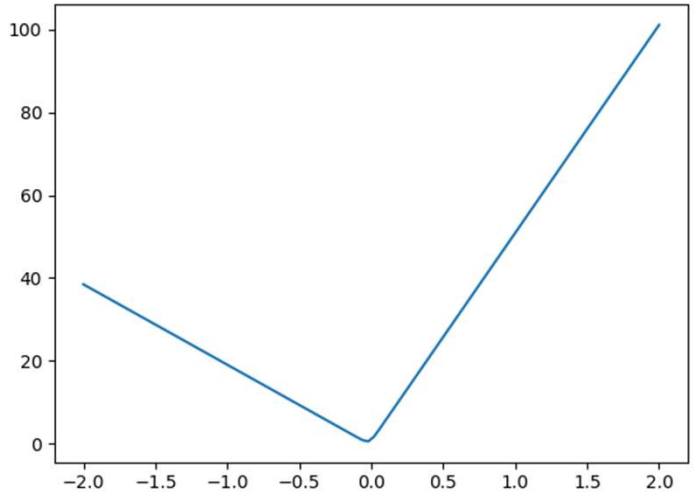
We can visualize the loss function since in this case it is a function of only 2 variables. The minimum is somewhere in the lightest area. We can see that there is a long flat valley so the gradient descent will be very slow when we move along the valley because the gradient will be close to 0



This graph shows the loss function along the bottom of the valley. We see how flat it is so the magnitude of the gradient will be very small = slow convergence



The other cross-section shows how the loss function drops off as we approach the valley



This gives us a problem: if we use a large learning rate to speed up the convergence, the second coordinate will not converge because it will jump from one side of the valley to the other

Learning rate = 0.01, no convergence

```
B = np.array([0,0.])
for _ in range(1000):
    B = B - 0.01 * gradient(B[0],B[1])
    print(B)
```

executed in 481ms, finished 12:40:12 2019-05-12

```
[ 0.24694358 -0.20314444]
[ 0.24998703 -0.00923311]
[ 0.24899799 -0.09478147]
[0.25201923 0.09770133]
[ 0.2450739  -0.40332013]
[ 0.24811738 -0.20940708]
[ 0.25116084 -0.01549522]
[ 0.25115638 -0.03203237]
[0.25296357 0.07760455]
```

Learning rate = 0.001, very slow convergence after 100,000 steps not close to the minimum

```
B = np.array([0,0.])
for _ in range(100000):
    B = B - 0.001 * gradient(B[0],B[1])
    print(B)
```

executed in 50.0s, finished 12:42:15 2019-05-12

```
[ 2.14228488 -0.04428776]
[ 2.14230417 -0.04428804]
[ 2.14232347 -0.04428832]
[ 2.14234276 -0.04428859]
[ 2.14236205 -0.04428887]
[ 2.14238134 -0.04428915]
[ 2.14240064 -0.04428943]
[ 2.14241993 -0.0442897 ]
[ 2.14243922 -0.04428998]
```

There are many ways to construct sequences converging to a minimum of the loss function. The gradient descent sequence can be very slow as the example shows. One of the fastest converging methods is Newton's method. While it constructs a sequence that converges fast it is also somewhat more complicated.

The gradient descent sequence can be derived from approximating a function with a linear function

$$f(x + \Delta x) \sim f(x) + \nabla f(x) \cdot \Delta x$$

Here we view x as being fixed and we view both sides as functions of Δx

In Newton's method we use the quadratic approximation to f :

$$f(x+\Delta x) \sim f(x) + \nabla f(x) \cdot \Delta x + \frac{1}{2}\Delta x^T \nabla^2 f(x) \Delta x$$

where $\nabla^2 f(x)$ is the matrix of double derivatives, this matrix is called the Hessian

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x^{(1)^2}} & \frac{\partial^2 f(x)}{\partial x^{(1)} \partial x^{(2)}} & \cdots & \frac{\partial^2 f(x)}{\partial x^{(1)} \partial x^{(d)}} \\ \frac{\partial^2 f(x)}{\partial x^{(2)} \partial x^{(1)}} & \frac{\partial^2 f(x)}{\partial x^{(2)^2}} & \cdots & \frac{\partial^2 f(x)}{\partial x^{(2)} \partial x^{(d)}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f(x)}{\partial x^{(1)} \partial x^{(d)}} & \frac{\partial^2 f(x)}{\partial x^{(2)} \partial x^{(d)}} & \cdots & \frac{\partial^2 f(x)}{\partial x^{(d)^2}} \end{pmatrix}$$

The idea is that for small Δx , the quadratic function is a good approximation to f and so the minimum of the quadratic function will approximate the minimum of f

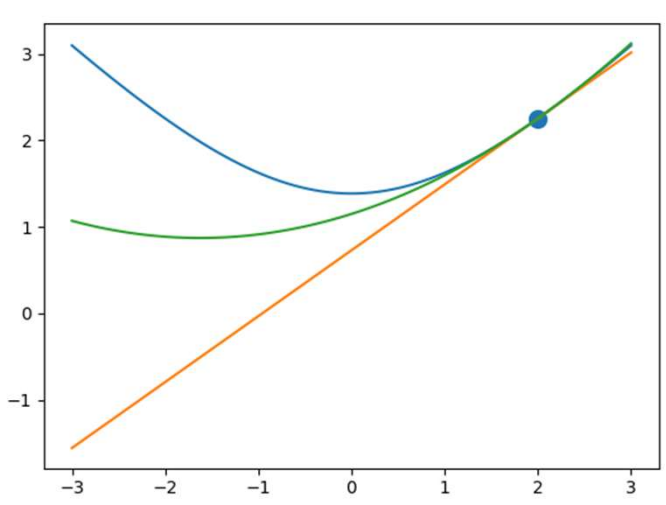
Taking derivative with respect to Δx and setting the derivative equal to 0, we get

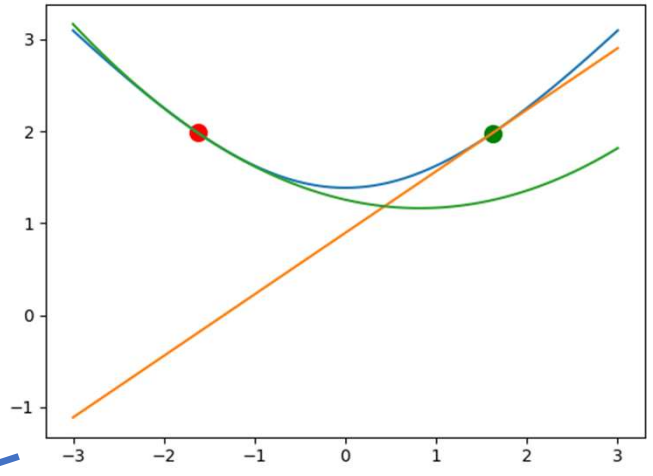
$$\nabla f(x) + \nabla^2 f(x)\Delta x = 0$$

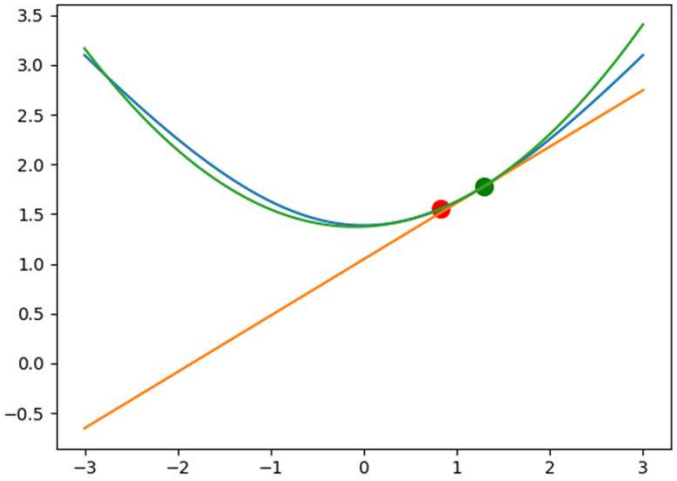
$$\Delta x = -\nabla^2 f(x)^{-1}\nabla f(x)$$

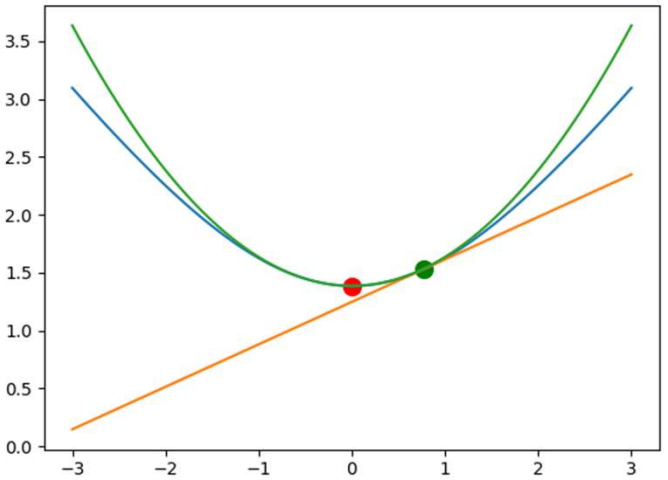
so

This is the increment in the Newton sequence


3
2
1
0
−1
−3
−2
−1
0
1
2
3


3
2
1
0
−1
−3
−2
−1
0
1
2
3


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−3
−2
−1
0
1
2
3

```
1  def Hessian(b0,b1):
2
3      return np.array([[[sigma(y * (b0 + b1 *t)) * sigma(-y * (b0 + b1 *t)),
4                         sigma(y * (b0 + b1 *t))* sigma(-y * (b0 + b1 *t)) *t],
5                        [sigma(y * (b0 + b1 *t))* sigma(-y * (b0 + b1 *t)) *t,
6                         sigma(y * (b0 + b1 *t))* sigma(-y * (b0 + b1 *t)) *t**2]] for
7                       y,t in zip(damage,temp)]).mean(axis=0)
```

executed in 31ms, finished 13:34:58 2019-05-11

```
1  B = np.array([0.,0.])
2  for _ in range(10):
3      B = B - np.linalg.inv(Hessian(B[0],B[1])).dot(gradient(B[0],B[1]))
4      print(B)
```

executed in 31ms, finished 13:43:34 2019-05-11

```
[ 9.61904762 -0.14952381]
[13.65573791 -0.21124698]
[14.93828914 -0.2306001 ]
[15.04229114 -0.23215369]
[15.04290163 -0.23216274]
[15.04290165 -0.23216274]
[15.04290165 -0.23216274]
```

Convergence in 6 iterations

At launch time of the Challenger mission the temperature was 31F.

Using the minimum of the loss function we found we can compute the prediction of the model

```
1  sigma(15.04290165 -0.23216274 * 31)
```

executed in 18ms, finished 13:10:49 2019-05-12

```
0.9996087829369722
```

So the model predicts with almost certainty that there would be damage to the O-ring seals

The other concept we need is *regularization* more specifically *L2 - regularization*

It is a way to put bounds on the size of the parameters when minimizing a loss function. It is a good tool to avoid overfitting.

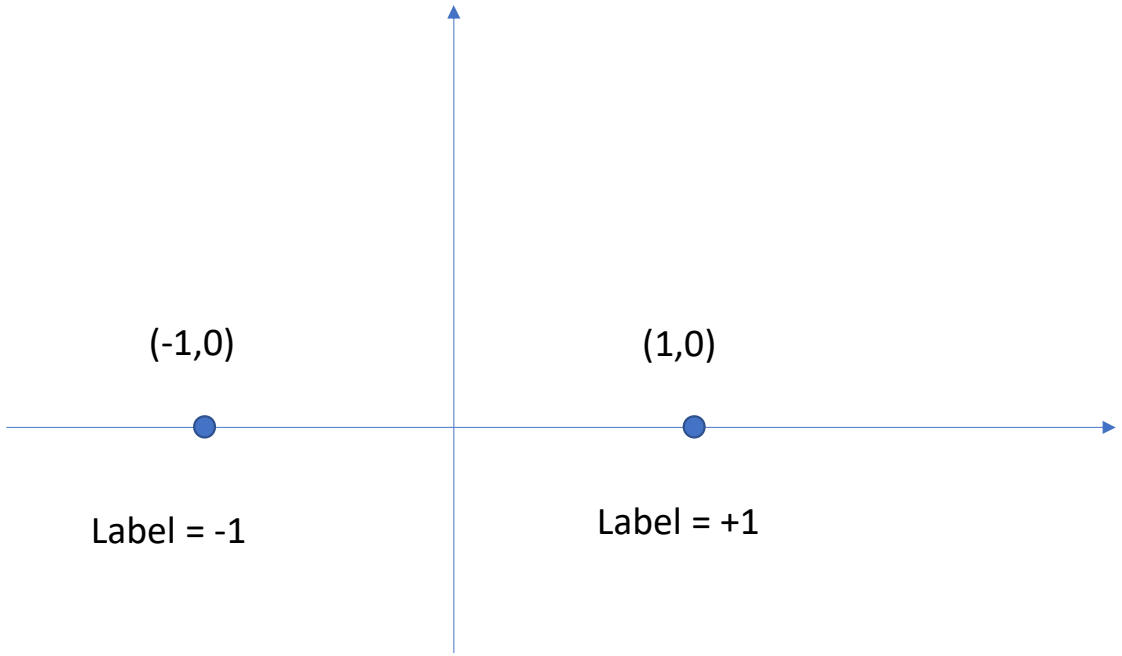
It works by adding a term to the loss function and then minimizing this new function, for instance for regression the regularized loss function becomes

$$\frac{1}{N}\sum\left(y_i - (\beta_0 + \beta_1 x_i^{(1)} + \beta_2 x_i^{(2)} + \cdots + \beta_d x_d^{(d)})\right)^2 + \lambda(\beta_1^2 + \beta_2^2 + \cdots + \beta_d^2)$$

The parameter λ is called the *regularization parameter,* the larger the regularization parameter, the more the minimization algorithm will focus on the size of the parameters at the expense of fitting to the data.

A regularized loss function will always have a minimum, this is not necessarily the case without regularization

Consider the simplest possible example



The loss function is

$$-\log(\sigma(-(\beta_0 - \beta_1)) - \log(\sigma(\beta_0 + \beta_1))$$

and the partials are

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = \frac{\exp(\beta_0 - \beta_1)}{1 + \exp(\beta_0 - \beta_1))} + \frac{-\exp(-(\beta_0 + \beta_1))}{1 + \exp(-(\beta_0 + \beta_1))}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_1} = \frac{-\exp(\beta_0 - \beta_1)}{1 + \exp(\beta_0 - \beta_1))} + \frac{-\exp(-(\beta_0 + \beta_1))}{1 + \exp(-(\beta_0 + \beta_1))}$$

$$\frac{\partial \mathcal{L}}{\partial \beta_2} = 0$$

Setting these expressions equal to 0 we get equations that have no solutions

Now the regularized loss function is

$$-\log(\sigma(-(\beta_0-\beta_1))-\log(\sigma(\beta_0+\beta_1))+\lambda(\beta_1^2+\beta_2^2)$$

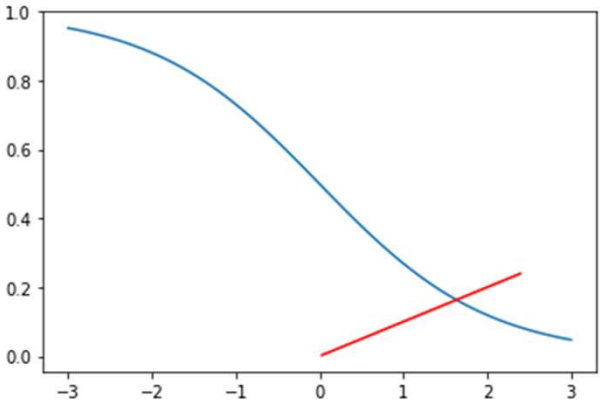
with partials

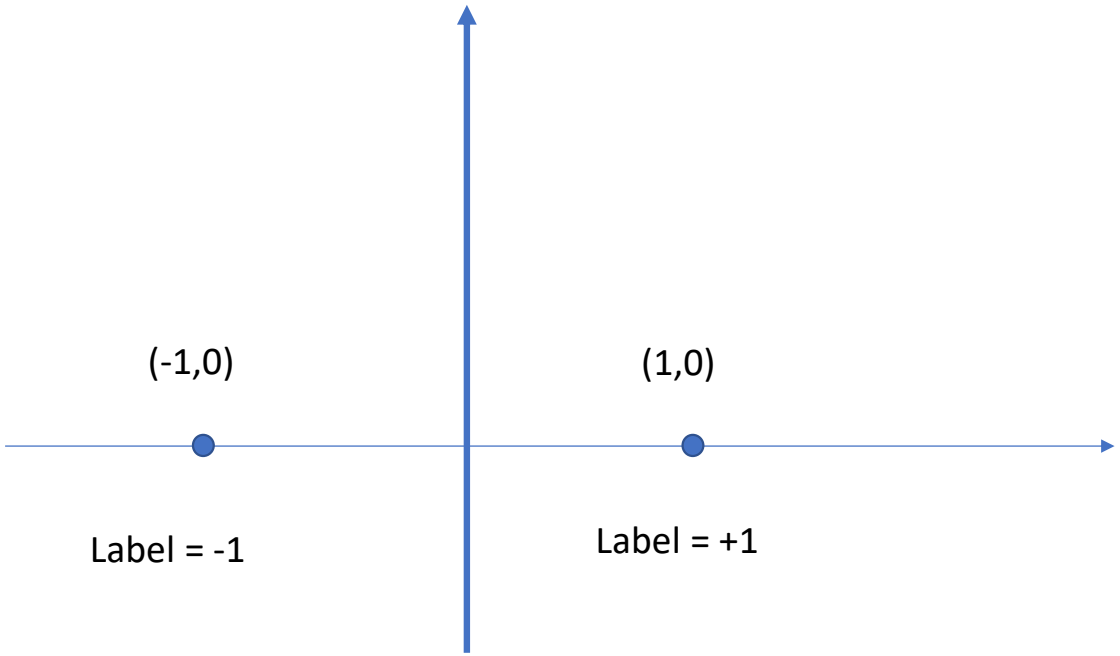
$$\frac{\partial\mathcal{L}}{\partial\beta_0}=\frac{\exp(\beta_0-\beta_1)}{1+\exp(\beta_0-\beta_1))}+\frac{-\exp(-(\beta_0+\beta_1))}{1+\exp(-(\beta_0+\beta_1))}$$

$$\frac{\partial\mathcal{L}}{\partial\beta_1}=\frac{-\exp(\beta_0-\beta_1)}{1+\exp(\beta_0-\beta_1))}+\frac{-\exp(-(\beta_0+\beta_1))}{1+\exp(-(\beta_0+\beta_1))}+2\lambda\beta_1$$

$$\frac{\partial\mathcal{L}}{\partial\beta_2}=2\lambda\beta_2$$

These equations are satisfied with $\beta_0 = \beta_2 = 0$ and $\lambda\beta_1 - \sigma(-\beta_1) = 0$ we get the classification function $\sigma(x^{(1)})$ so any point $(x^{(1)}, x^{(2)})$ is classified by $sign(x^{(1)})$





Next we are going to discuss the boosting algorithm that currently is the most popular, Gradient Boosting

Consider a data set $\{(y_{j_1}, \underline{x}_1), (y_{j_2}, \underline{x}_2), \ldots, (y_{j_N}, \underline{x}_N)\}$ and a classifier f. Let $\mathcal{L}$ be a loss function and the value (the loss) on the classifier f and the data set, so

$$\mathcal{L}(\{(y_{j_i}, \underline{x}_i)\}, f) = -\frac{1}{N}\sum_i \ell(y_{j_i}, f(\underline{x}_i))$$

We want to consider a new classifier of the form $f + g$ such that

$$\mathcal{L}(\{(y_{j_i}, \underline{x}_i)\}, f + g) \leq \mathcal{L}(\{(y_{j_i}, \underline{x}_i)\}, f)$$

i.e. the new classifier improves the loss.

Let $z_i = f(\underline{x}_i)$ so we can view the loss function as a function of $z_1, z_2, \ldots, z_N$. The partial derivatives with respect to the z's are given by

$$\frac{\partial \mathcal{L}(\{(y_{j_i}, \underline{x}_i)\}, z_1, z_2, \ldots, z_n)}{\partial z_i} = -\frac{1}{N}\frac{\partial \ell(y_{j_i}, z_i)}{\partial z_i}$$

As an example consider the logistic loss function for a single data point

$$-\log p(y|\underline{x}) = \ell(y, f(\underline{x})) = \log\left(\frac{1}{1+\exp(-yf(\underline{x}))}\right) = \log\left(\frac{1}{1+\exp(-yz)}\right) = -\log\left(1+\exp(-yz)\right)$$

When we were doing logistic regression, f was of the form

$$f(\underline{x}) = \beta_0 + \beta_1 x^{(1)} + \beta_2 x^{(2)} + \cdots + \beta_d x^{(d)} = z$$

If $p(y=1|\underline{x}) = \dfrac{1}{1+\exp(-z)}$ and so

$$p(y=-1|\underline{x}) = 1 - \frac{1}{1+\exp(-z)} = \frac{1}{1+\exp(z)}$$

In any case $p(y|\underline{x}) = \dfrac{1}{1+\exp(-yz)}$

The derivative w.r.t. z is $y\dfrac{\exp(-yz)}{1+\exp(-yz)} = \begin{cases} 1-p(y=1|z), y=1 \\ -p(y=1|z), y=-1 \end{cases}$

We shall use as our example a binary classification problem, where the classifier is of the form $\sigma(f(\underline{x}))$ and the loss function $\ell(y_{j_i}, f(\underline{x}_i)) = -\log(\sigma(y_{j_i} f(\underline{x}_i))$

Thus the gradient

$$\nabla_{\underline{z}} \mathcal{L}(\underline{z}) = \frac{1}{N} \begin{pmatrix} y_{m_1} \dfrac{1}{1+\exp(y_{m_1} z_1)} \\ y_{m_2} \dfrac{1}{1+\exp(y_{m_2} z_2)} \\ \vdots \\ y_{m_N} \dfrac{1}{1+\exp(y_{m_N} z_N)} \end{pmatrix}$$

The Hessian is a diagonal matrix since all the mixed derivatives $\dfrac{\partial^2}{\partial z_i \partial z_j} \mathcal{L} = 0$ for $i \neq j$ and the diagonal terms are given by

$$\frac{\partial}{\partial z_i}(y_{m_i} \frac{1}{1+\exp(y_{m_i} z_i)}) = y_{m_i} \left(-\frac{y_{m_i} \exp(y_{m_i} z_i)}{(1+\exp(y_{m_i} z_i))^2} \right) = -\frac{1}{1+\exp(y_{m_i} z_i)} \frac{1}{1+\exp(-y_{m_i} z_i)}$$

Thus we get the Newton step

$$
\begin{aligned}
&\Delta \\
&\quad = -(\nabla^2 \mathcal{L})^{-1} \nabla \mathcal{L} \\
&\quad = \begin{pmatrix} (1+\exp(y_{m_1} z_1))(1+\exp(-y_{m_1} z_1)) & 0 & \dots \\ 0 & (1+\exp(y_{m_2} z_2)(1+\exp(-y_{m_2} z_2)) & \dots \\ \vdots & \vdots & \vdots \\ 0 & 0 & \dots \end{pmatrix} \begin{pmatrix} y_{m_1} \dfrac{1}{1+\exp(y_{m_1} z_1)} \\ y_{m_1} \dfrac{1}{1+\exp(y_{m_2} z_2)} \\ \vdots \\ y_{m_N} \dfrac{1}{1+\exp(y_{m_N} z_N)} \end{pmatrix} \\
&\quad = \begin{pmatrix} y_{m_1}(1+\exp(-y_{m_1} z_1)) \\ \vdots \\ y_{m_N}(1+\exp(-y_{m_N} z_N)) \end{pmatrix}
\end{aligned}
$$

So if we can find a function g such that $g(\underline{x}_i) = y_{m_i}(1+\exp(-y_{m_i} f(\underline{x}_i))$ the classifier $f+g$ will improve the fit.

Remark that the general case is quite similar to this because the Hessian is always diagonal.

The way we are going to find g is to use a *Regression Tree*

The general algorithm for fitting a tree to a function is quite similar to the process for fitting a decision tree.

Say we are given a data set $\{(y_i, \underline{x}_i)\}_{i=1,2,\ldots,N}$ where the y_i's are real number. To fit a regression tree we use an iterative process. Assume we are at a node and we have to split the node in some way. Let $\{(y_j, \underline{x}_j\}$ be the points in the node. We then seek the coordinate x_k and the split value s and values c_1 and c_2 to minimize

$$\sum_{x_k \leq s} (c_1 - y_j)^2 + \sum_{x_k > s} (c_2 - y_j)^2$$

If we split at x_k, say $x_k < s$, and let $n_1 = \#\{\underline{x}_i \text{ with } \underline{x}_{ik} < s\}$ and $n_2 = \#\{\underline{x}_i \text{ with } \underline{x}_{ik} \geq s\}$ then we get the minimal impurity by choosing $c_1 = \dfrac{1}{n_2} \sum_{x_k < s} y_j$ and $c_2 = \dfrac{1}{n_1} \sum_{x_k \geq s} y_j$. This is easy to see by differentiating with respect to c_1 and c_2 and setting the derivatives $= 0$

The gradient boosting algorithm then proceeds as follows:

Start with f_0, a constant function where the value a is chosen to minimize the loss function

$$\mathcal{L}(\{y_{m_i}, \underline{x}_i\}, a)$$

Compute the Newton step which is an N-dimensional vector and fit a regression tree T_0 to the values of the Newton step.

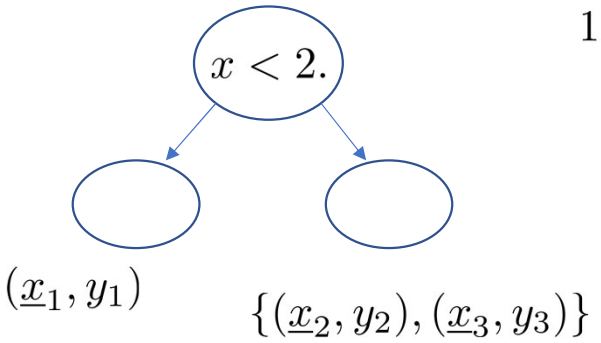
The updated classifier is then $f_1 = a + T_0$. We then continue this way until we are satisfied with the fit. After M steps our classifier is a sum of tree functions

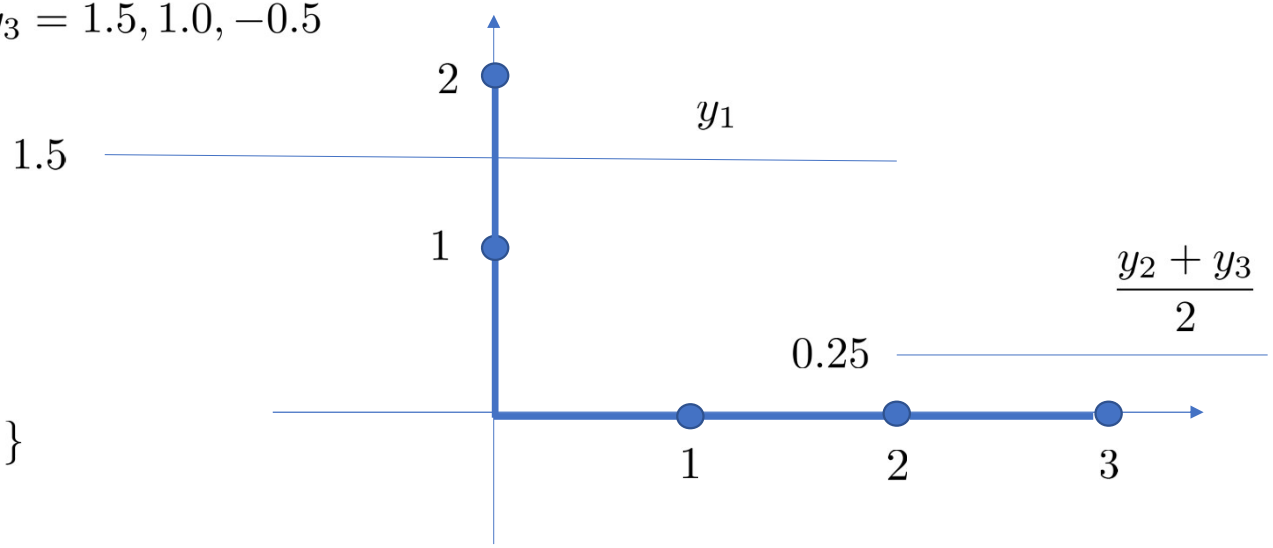
$$a + T_0 + T_1 + T_2 + \cdots + T_M$$

Remark that this is a piecewise constant function.

The function associated to a tree T is computed on a point $\underline{x}$ by sending the point through the tree until we land at a terminal node. The value of the function is then the average of the y_i's for the data points in the node. Remark that this is a piecewise constant function, it can only take values that are averages of the $\{y_j\}$ that are already in the data set.

For instance consider data $\{(1.0, 1.5), (2.0, 1.0), (3.0, -0.5)\}$ i.e. $\underline{x}_1, \underline{x}_2, \underline{x}_3 = 1.0, 2.0, 3.0$ and $y_1, y_2, y_3 = 1.5, 1.0, -0.5$





Now let's apply this to our toy example. We use the logistic model i.e. our model for the conditional probability is $p(y=1|\underline{x}) = \dfrac{1}{1+\exp(-f(\underline{x}))}$

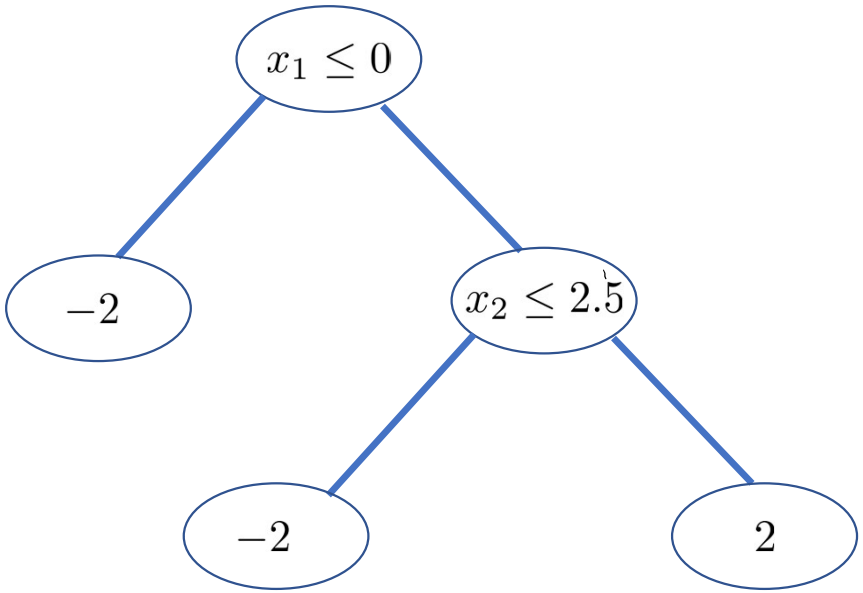
The loss function is

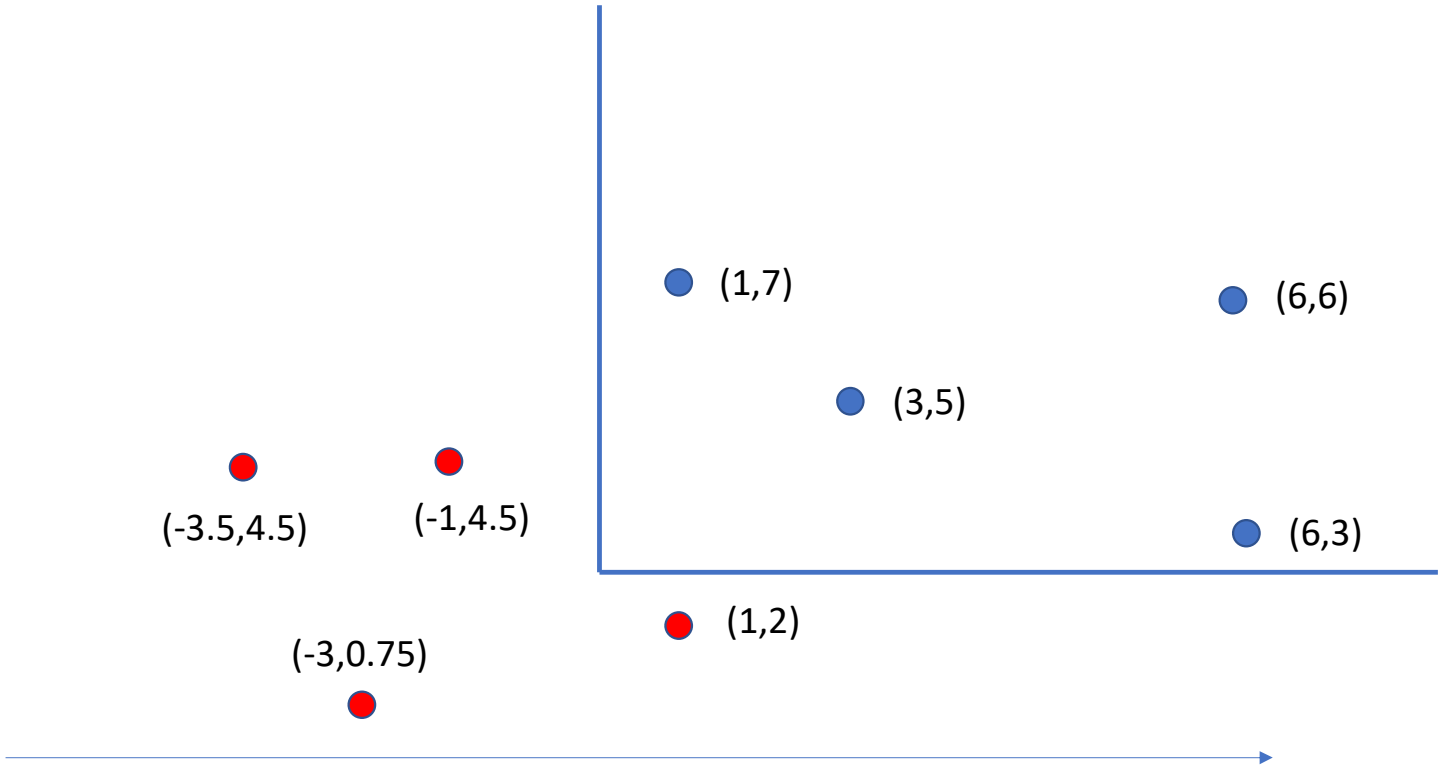
$$\mathcal{L}(f) = -\frac{1}{8}\sum \log \frac{1}{1+\exp(-y_i f(\underline{x}_i))}$$

The constant function that minimizes the loss function is $f_0 = 0$. Thus from our formula the Newton step is given by $\begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ -2 \\ -2 \\ -2 \\ -2 \end{pmatrix}$

Let T_1 be the tree with the split at $x_1 = 0$ and if $x_1 > 0$ we split at $x_2 = 2.5$.

Fitting this to the Newton step gives $T_1(\underline{x}) = \begin{cases} -2 \text{ if } x_1 \leq 0 \\ -2 \text{ if } x_1 > 0, x_2 \leq 2.5 \\ 2 \text{ if } x_1 > 0, x_2 > 2.5 \end{cases}$




(1,7)
(6,6)
(3,5)
(-3.5,4.5)
(-1,4.5)
(6,3)
(1,2)
(-3,0.75)

In general assume we have computed $F_{m-1} = f_0 + f_1 + \cdots + f_{m-1}$ where the f_is are functions defined by trees. Let $\mathcal{L}(\{(y_{j_i}, \underline{x}_i\}, F_{m-1})$ be the loss function. We want to find the f_m that most improves the loss function

$$\mathcal{L}(\{(y_{j_i}, \underline{x}_i\}, F_{m-1} + \underline{f_m})$$

We use the quadratic Taylor expansion in the parameters $z_i = F_{m-1}(\underline{x}_i)$, which is very simple since the loss function is of the form

$$\mathcal{L}(\{(y_{j_i}, \underline{x}_i\}, F_{m-1}) = -\frac{1}{N}\sum_i \ell((y_{j_i}, \underline{x}_i), F_{m_i}(\underline{x}_i)) = -\frac{1}{N}\sum_i \ell((y_{j_i}, \underline{x}_i), z_i)$$

so we can just expand the individual terms

$$\ell((y_{j_i}, \underline{x}_i), z_i + f_m(\underline{x}_i)) \sim \ell((y_{j_i}, \underline{x}_i), z_i) + \frac{d}{dz_i}\ell((y_{j_i}, \underline{x}_i), z_i) f_m(\underline{x}_i) + \frac{1}{2}\frac{d^2}{dz_i^2}\ell((y_{j_i}, \underline{x}_i), z_i) f_m(\underline{x}_i)^2$$

Let $g_i = \dfrac{d}{dz_i}\ell((y_{j_i}, \underline{x}_i), z_i)$, $h_i = \dfrac{d^2}{dz_i^2}\ell((y_{j_i}, \underline{x}_i), z_i)$ and $w_i = f_m(\underline{x}_i)$ so we have

$$\ell((y_{j_i}, \underline{x}_i), z_i + w_i) \equiv \ell((y_{j_i}, \underline{x}_i), z_i) + g_i w_i + \frac{1}{2} h_i w_i^2$$

The quadratic function has minimum when $w_i = -\frac{g_i}{h_i}$ and the minimum value is $\ell((y_{j_i}, \underline{x}_i), z_i) - \frac{1}{2}\frac{g_i^2}{h_i}$

This suggests that we should try to fit a tree $\mathcal{T}_m$ to the w_is i.e. such that $\mathcal{T}_m(\underline{x}_i) \sim w_i$ for $i = 1, 2, \ldots, N$

But we also want to include regularization terms to limit the size of the w_is and the size of the tree. If $|\mathcal{T}|$ denotes the size of the tree i.e. the number of leaves we regularize by minimizing the regularized loss function

$$\mathcal{L}(\{(y_{j_i}, \underline{x}_i)\}, z_i) + \sum_i g_i w_i + \frac{1}{2}\sum_i h_i w_i^2 + \gamma|T| + \frac{1}{2}\lambda\sum_i w_i^2$$

where γ and λ are hyper-parameters.

This gives

$$w_i = -\frac{g_i}{h_i + \lambda}$$

Let's assume the w_is are given by a tree $\mathcal{T}$, i.e. each w_i is the value at some leaf of $\mathcal{T}$. Let $\mathcal{N}$ denote a leaf in the tree. Let $\{\underline{x}_{1_\mathcal{N}}, \underline{x}_{2_\mathcal{N}}, \ldots, \underline{x}_{r_\mathcal{N}}\}$ be the data that under the decision rules of the tree end up in leaf $\mathcal{N}$.

We can write the regularized loss function

$$\mathcal{L}(\{(y_{j_i}, \underline{x}_i)\}, F_{m-1}) + f_m)$$

$$\simeq \sum_{\mathcal{N}} \left(\mathcal{L}^{(\mathcal{N})}(\{(y_i, \underline{x}_{i_\mathcal{N}})\}, F_{m-1}) + \sum_{i_\mathcal{N}} g_{i_\mathcal{N}} w_\mathcal{N} + \frac{1}{2} \sum_{i_\mathcal{N}} h_{i_\mathcal{N}} w_\mathcal{N}^2 + \frac{\lambda}{2} \sum_{i_\mathcal{N}} w_\mathcal{N}^2 \right) + \gamma |\mathcal{T}|$$

$$= \sum_{\mathcal{N}} \left(\mathcal{L}^{(\mathcal{N})}(\{(y_i, \underline{x}_{i_\mathcal{N}})\}, F_{m-1}) + \sum_{i_\mathcal{N}} g_{i_\mathcal{N}} w_\mathcal{N} + \frac{1}{2} \sum_{i_\mathcal{N}} (h_{i_\mathcal{N}} + \lambda) w_\mathcal{N}^2 \right) + \gamma |\mathcal{T}|$$

We look at the part of the loss function coming from the data in the leaf $\mathcal{N}$

The value of $w_{\mathcal{N}}$ which produces the greatest improvement in the loss function is

$$w^*_{\mathcal{N}} = -\frac{\sum\limits_{i_{\mathcal{N}}} g_{i_{\mathcal{N}}}}{\sum\limits_{i_{\mathcal{N}}} (h_{i_{\mathcal{N}}} + \lambda)}$$

The optimal value of $\sum\limits_i f_m(\underline{x}_i)$ is $-\frac{1}{2}\sum\limits_{\mathcal{N}} \left(\frac{\left(\sum\limits_{i_{\mathcal{N}}} g_{i_{\mathcal{N}}}\right)^2}{\sum\limits_{i_{\mathcal{N}}} (h_{i_{\mathcal{N}}} + \lambda)} \right)$ so the improved value of the loss function we get by adding the function f_m defined by the tree $\mathcal{T}$ with values at the nodes the $w^*_{\mathcal{N}}$, is

$$\mathcal{L}(\{(y_{j_i} \underline{x}_i)\}, F_{m-1}) - \frac{1}{2}\sum_{\mathcal{N}} \left(\frac{\left(\sum\limits_{i_{\mathcal{N}}} g_{i_{\mathcal{N}}}\right)^2}{\sum\limits_{i_{\mathcal{N}}} (h_{i_{\mathcal{N}}} + \lambda)} \right) + \gamma |\mathcal{T}|$$

If we split a leaf $\mathcal{N}$ into two new leaves $\mathcal{N}_L$ and $\mathcal{N}_R$ and compute the change in the loss function we get

$$-\frac{1}{2}\frac{\left(\sum\limits_{i_{\mathcal{N}_L}} g_{i_{\mathcal{N}_L}}\right)^2}{\sum\limits_{i_{\mathcal{N}_L}}(h_{i_{\mathcal{N}_L}}+\lambda)} - \frac{1}{2}\frac{\left(\sum\limits_{i_{\mathcal{N}_R}} g_{i_{\mathcal{N}_R}}\right)^2}{\sum\limits_{i_{\mathcal{N}_R}}(h_{i_{\mathcal{N}_R}}+\lambda)} + \frac{1}{2}\frac{\left(\sum\limits_{i_{\mathcal{N}}} g_{i_{\mathcal{N}}}\right)^2}{\sum\limits_{i_{\mathcal{N}}}(h_{i_{\mathcal{N}}}+\lambda)} + \gamma$$

Remark the extra γ we get since the new tree now has one more leaf.

This formula is used to find the best split at a given leaf. We want to find the split that gives us the best improvement in the loss function. Depending on the size of γ it may well happen that there is no split that improves the loss function in which case we keep the leaf as it is.

We are going to use the lightgbm package instead of the sklearn GradientBoosting class. Lightgbm is both faster and has a lot more features such as the regularization we just have discussed.

https://lightgbm.readthedocs.io/en/latest/Python-Intro.html#setting-parameters

```
(base) C:\Users\niels>conda install lightgbm
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\niels\Anaconda3

  added / updated specs:
    - lightgbm


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-4.6.14               |           py37_0         2.1 MB  conda-forge
    lightgbm-2.2.3             |   py37h6538335_0         528 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         2.6 MB

The following packages will be UPDATED:

  conda                                     4.6.11-py37_0 --> 4.6.14-py37_0
  lightgbm                            2.2.2-py37h6538335_0 --> 2.2.3-py37h6538335_0
```

```
1  gb_clf = lgb.LGBMClassifier(objective='multi_class',n_classifiers=400,
2                              n_estimators=400,reg_alpha=0.4,reg_lambda=0.5,learning_rate=0.01)
```