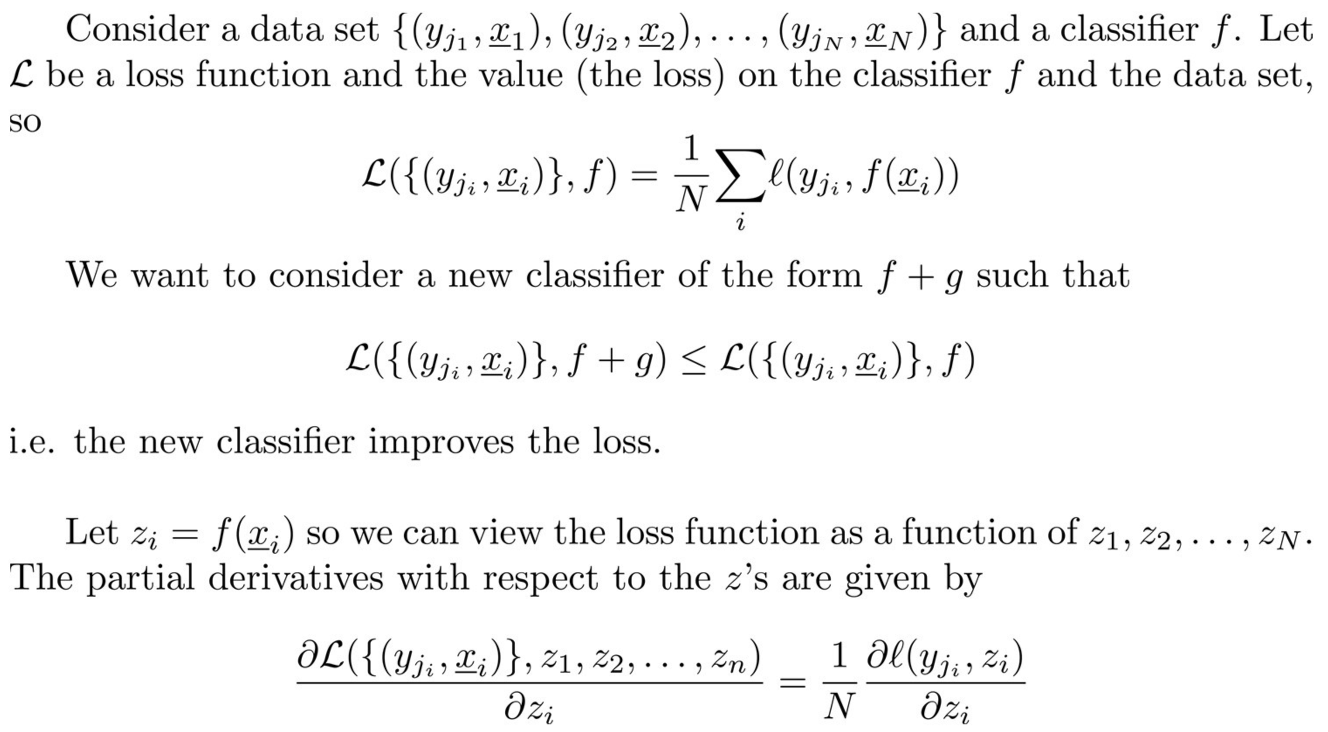
Gradient Boosting

Lecture 11

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

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

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

So we have

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

The derivative of $-\log p(y|\underline{x}) = -\log \sigma(yz)$ w.r.t. z is

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

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

$$\ell(y_{j_i}, f(\underline{x}_i)) = -\log \sigma(y_{j_i} z_i)$$

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

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

Thus we get the Newton step

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

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 $n_1 = \#\{\underline{x}_i$ with $\underline{x}_{ik} < s\}$ and $n_2 = \#\{\underline{x}_i$ with $\underline{x}_{ik} \geq s\}$ then we get the minimal impurity by choosing $c_1 = \dfrac{1}{n_2} \sum\limits_{x_k < s} y_j$ and $c_2 = \dfrac{1}{n_1} \sum\limits_{x_k \geq s} y_j$. This is easy to see by differentiating with respect to c_1 and c_2 and putting 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.

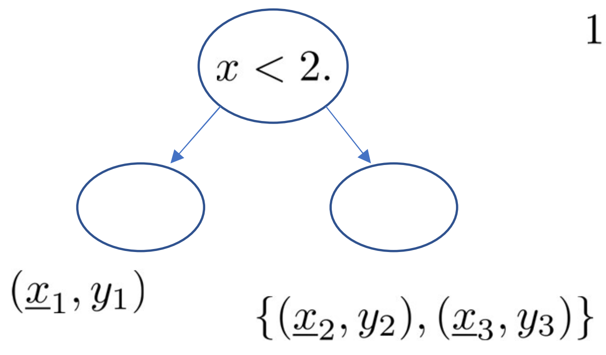
The updated classifier is then $f_1 = a + T_0$. We then continue this way until we are satisfied with the fit. After M steps our classifier is a sum of tree functions

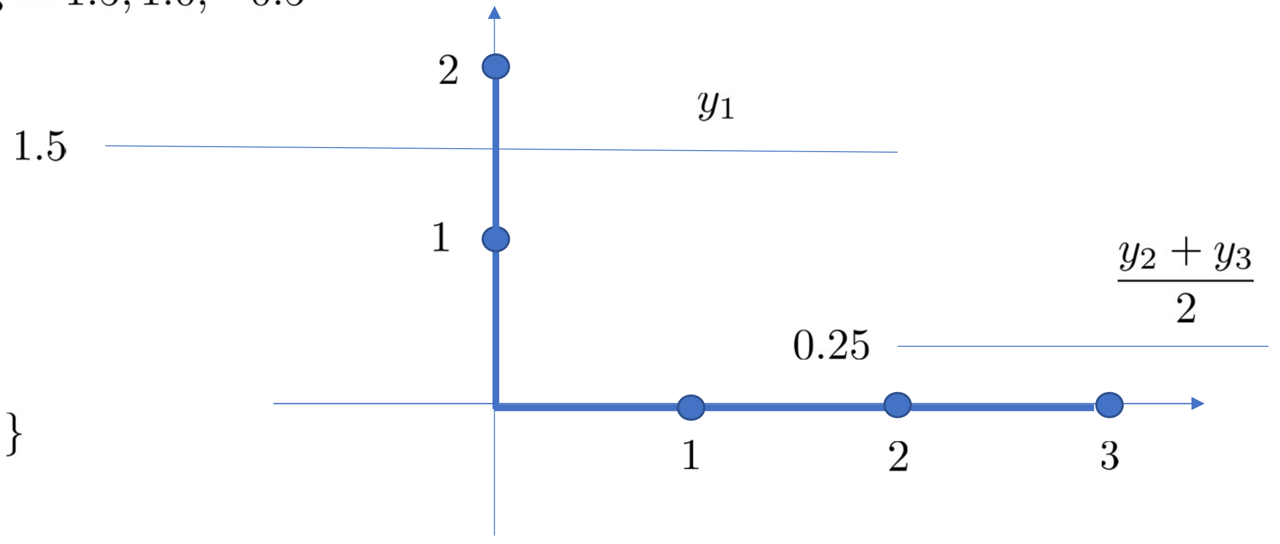
$$a + T_0 + T_1 + T_2 + \cdots + T_M$$

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 here the y_i's are real numbers not ± 1). 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 $y_1, y_2, y_3 = 1.5, 1.0, -0.5$



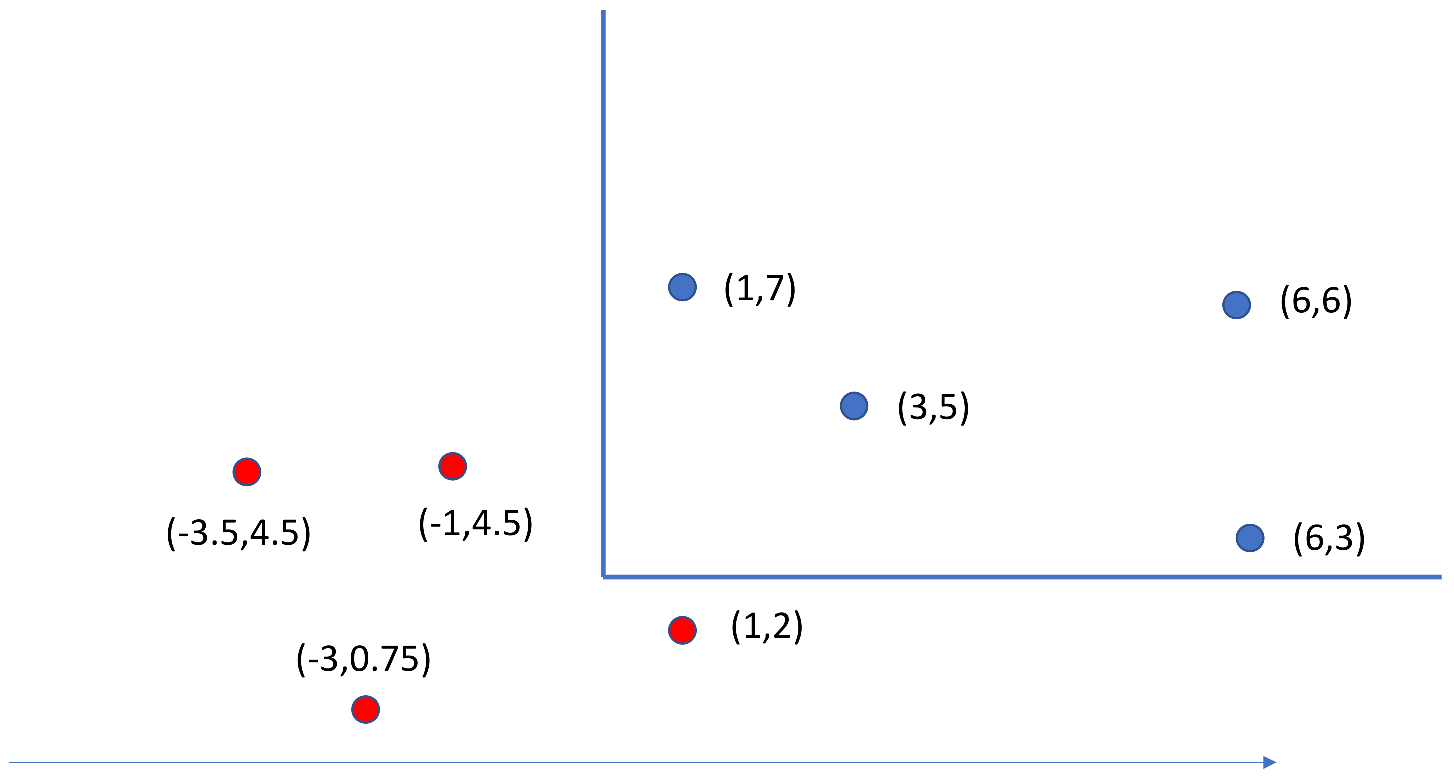


Now let's apply this to our toy example. We use the logistic model i.e. our model for the conditional probability is $p(y = 1|\underline{x}) = \dfrac{1}{1 + \exp(-f(\underline{x}))}$

The loss function is

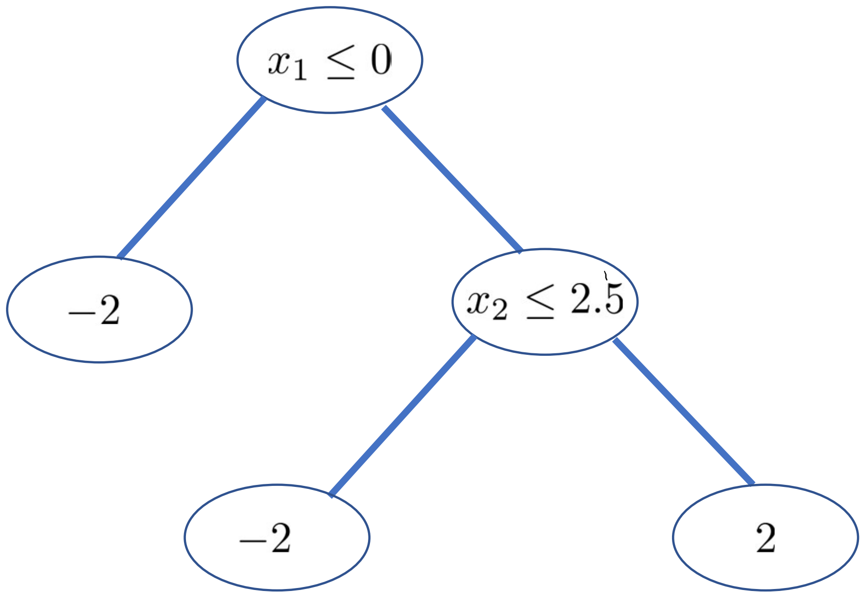
$$\mathcal{L}(f) = -\frac{1}{8} \sum \log \frac{1}{1 + \exp(-y_i f(\underline{x}_i))}$$

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


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

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 $T_1(\underline{x}) = \begin{cases} -2 \text{ if } x_1 \leq 0 \\ -2 \text{ if } x_1 > 0, x_2 \leq 2.5 \\ 2 \text{ if } x_1 > 0, x_2 > 2.5 \end{cases}$



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

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

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

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

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

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

$$\begin{aligned}
&\mathcal{L}(\{(y_{j_i}, \underline{x}_i)\}, F_{m-1} + f_m) \\
&\simeq \sum_{\mathcal{N}} \left(\mathcal{L}^{(\mathcal{N})}(\{(y_i, \underline{x}_{i_\mathcal{N}})\}, F_{m-1}) + \sum_{i_\mathcal{N}} g_{i_\mathcal{N}} w_\mathcal{N} + \frac{1}{2} \sum_{i_\mathcal{N}} h_{i_\mathcal{N}} w_\mathcal{N}^2 + \frac{\lambda}{2} \sum_{i_\mathcal{N}} w_\mathcal{N}^2 \right) + \gamma |\mathcal{T}| \\
&= \sum_{\mathcal{N}} \left(\mathcal{L}^{(\mathcal{N})}(\{(y_i, \underline{x}_{i_\mathcal{N}})\}, F_{m-1}) + \sum_{i_\mathcal{N}} g_{i_\mathcal{N}} w_\mathcal{N} + \frac{1}{2} \sum_{i_\mathcal{N}} (h_{i_\mathcal{N}} + \lambda) w_\mathcal{N}^2 \right) + \gamma |\mathcal{T}|
\end{aligned}$$

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

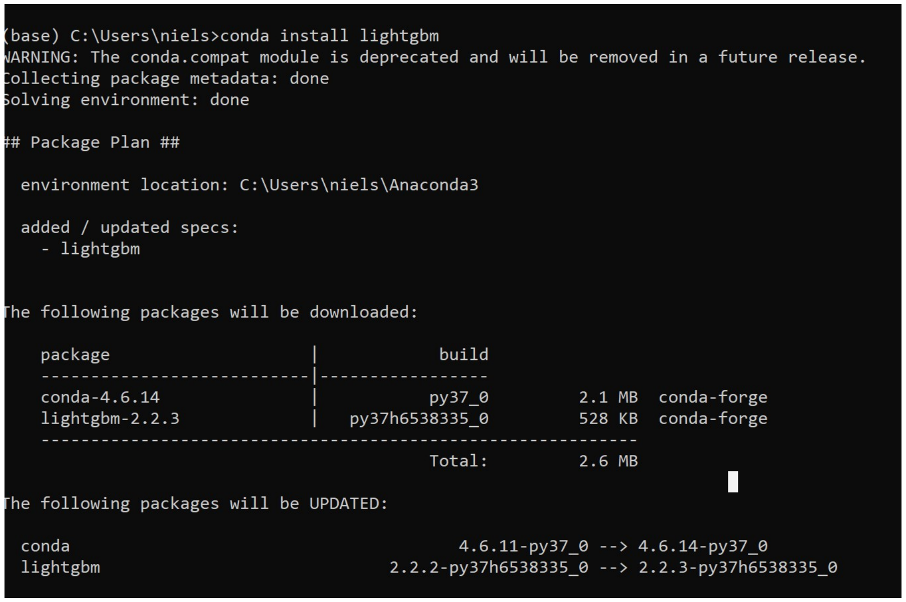
$$-\frac{1}{2}\frac{\left(\sum\limits_{i_{\mathcal{N}_L}} g_{i_{\mathcal{N}_L}}\right)^2}{\sum\limits_{i_{\mathcal{N}_L}}(h_{i_{\mathcal{N}_L}}+\lambda)} - \frac{1}{2}\frac{\left(\sum\limits_{i_{\mathcal{N}_R}} g_{i_{\mathcal{N}_R}}\right)^2}{\sum\limits_{i_{\mathcal{N}_R}}(h_{i_{\mathcal{N}_R}}+\lambda)}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

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

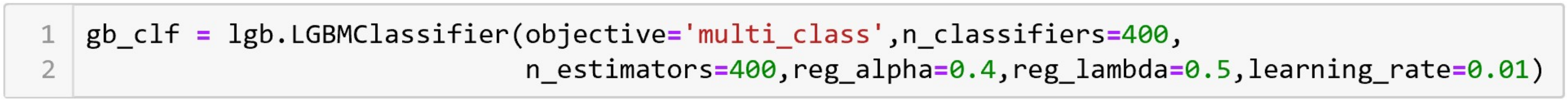
This formula is used to find the best split at a given leaf. We want to find the split that gives us the best improvement in the loss function. Depending on the size of γ it may well happen that there is no split that improves the loss function in which case we keep the leaf as it is.

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
ARNING: The conda.compat module is deprecated and will be removed in a future release.
ollecting package metadata: done
olving environment: done

## Package Plan ##

  environment location: C:\Users\niels\Anaconda3

  added / updated specs:
    - lightgbm


he 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

he 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
```



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