Tree algorithms

Tateyama Kaoru

February 13, 2025

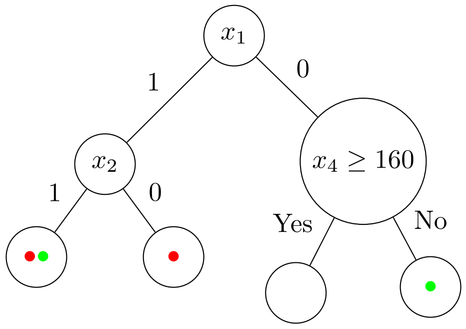
1 Decision tree

1.1 build tree

A decision tree contains root,leaves,nodes and edges.root and leaves are nodes themselves and nodes are used to make decisions with condictions expressed in the nodes.If a data agrees with the condiction then it goes to the left child node of the current node(going to left or right is by convention) otherwise it goes to the right child node.A concise example of how decision tree works is to use it decide given features wether its label is 0 or 1.The following is a table gathering the features and labels of datas

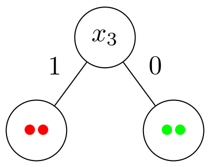
x_1	x_2	x_3	x_4	y	*color*
0	0	0	125	0	🟢
1	1	1	180	1	🔴
1	1	0	210	0	🟢
1	0	1	167	1	🔴

(Note that some of the features are binary.)Now we build a tree to diagonalize if a new data is labelled by 1.



(green dot represents y being 0 and red dot represents y being 1)We look at this tree and find that we can decide the label of a data by going through all the nodes.In the buttom left leaf,it turns out to be one red dot and one green dot which is called impure while its pairwise node only contains one red dot which is called pure.(the bottom right leaf is also pure as it only contains one green dot.)We say that pure leaves perform better than impure nodes as they give decisions without ambiguity.

This is not the only way to build a tree for the given data,we can also construct another tree



This tree clearly does a better job than the previous one as all its leaves are pure.But this tree may have the problem of high variance as it only take x_3 into consideration.

If there are numerous ways to construct trees,to build a better performaning tree,we need to calculate Gini impurity for each node to decide what feature and condiction to put in a node.We define Gini impurity as follow

$$Gini \quad impurity = 1 - \sum_i (ratio \quad of \quad label \quad i)^2$$

where i goes through all catetgories of labels in the considered node.In the previous to examples we only have two categories of labels.In the first decision tree,impurity of bottom left leaf is

$$1 - (\frac{1}{2})^2 - (\frac{1}{2})^2 = 0.5$$

while the bottom right leaf has a 0 Gini impurity.Now we introduce the idea of weighted Gini impurity to describe how well a node performs.The definition of weighted Gini impurity goes

$$Weighted\ Gini\ impurity = \sum_{i=1,2} sample\ ratio\ of\ childnode\ i\ * \\ Gini\ impurity\ of\ childnode\ i$$

In the first tree,the weighted Gini impurty for node x_2 is

$$\frac{2}{3} * (1 - (\frac{1}{2})^2 - (1 - \frac{1}{2})^2) + \frac{1}{3} * (1 - 1^2 - 0^2) = \frac{1}{3}$$

With Gini impurity,the strategy we build a decision tree is

(i)for each feature we take it as a node,we calculate its weighted Gini impurity with respect to all possible condictions and pick out the condiction with smallest Gini impurity as the node condiction.

(ii)Going through all the feature that are preprocessed by step(i),pick out the feature with smallest weighted Gini impurity and choose it to be root.

(iii)repeat (i) with the filtered data by root in childnodes and repeat the first part of (ii) to pick out the most capbable features to be childnodes.

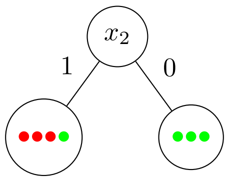
Consider a new data in a table

x_1	x_2	x_3	y	*color*
1	1	7	0	●
1	0	12	0	●
0	1	18	1	●
0	1	35	1	●
1	1	38	1	●
1	0	50	0	●
0	0	83	0	●

We are going to build a tree to decide wether a data should be labelled as 1.As the stratrgy suggests,we first go through all features with all possible conditions and find that

$$\begin{aligned}&(1) feature\ x_1\ with\ condiction\ 0\ or\ 1\ gets\ a\ smallset\\ &\quad weighted\ Gini\ impurity\ of\ 0.405\\ &(2) feature\ x_2\ with\ condiction\ 0\ or\ 1\ gets\ a\ smallset\\ &\quad weighted\ Gini\ impurity\ of\ 0.214\\ &(3) feature\ x_3\ with\ condiction\ x_3 \leq 15\ gets\ a\ smallset\\ &\quad weighted\ Gini\ impurity\ of\ 0.343\end{aligned}$$

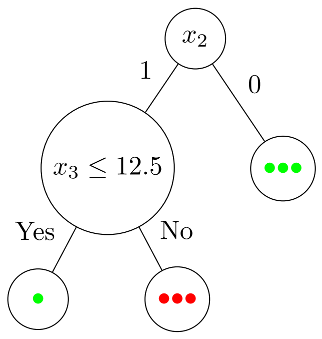
Therefore,we choose feature x_2 with condiction 0 or 1 as root of the tree.Now we get a root



Note that the right childnode is pure so we don't need further update.The left childnode is impure so we need to extend it.To decide which feature we need to use,we continue the strategy.According to the filtered data,one can discover that

$$\begin{aligned}&(1) feature\ x_1\ with\ condiction\ 0\ or\ 1\ gets\ a\ smallset\\ &\quad weighted\ Gini\ impurity\ of\ 0.205\\ &(2) feature\ x_3\ with\ condiction\ x_3 \leq 12.5\ gets\ a\ smallset\\ &\quad weighted\ Gini\ impurity\ of\ 0\end{aligned}$$

And now we can claim our tree has been built and it should look like the following



And now we have built a decision tree that performs as best as it can.If now comes a new data that reads

x_1	x_2	x_3
1	1	15

Then according to the decision tree it first goes left and goes right to be classified as red dot,namely labelled as 1.

Sometimes,we may not build a tree whose leaves are all pure,in this case we say these impure leaves predict the probablity of data being labelled as 1(or 0 if one may prefer).

So far,we have only built tree for classification problem,what if we're dealing with a prediction problem(where we name the tree as regression tree)?It is almost the same precedure when dealing with prediction problem except we have to obey new rules:

(i)replace Gini impurity with MSE(still the smaller the better) and replace weighted Gini impurity with weight MSE.

(ii)we predict new value by taking average of values living in the leaf.

1.2 prune tree

One can easily see that trees tend to have high variance.To address the problem of high variance,we need to prune some of leaves from a tree.We can either pre-pruning or post-pruning.Pre-pruning is easy to impliment,all we have to do is to limit the depth of the tree or set lower bound for number of samples in each leaf or node before hand.Post-pruning is a bit more complex where we'll adpot Cost-Complexity Pruning.For the sake of convenience,we start with regression tree.

We need to define Tree Score in the first place

$$Tree\ Score = SSR\ +\ \alpha * T$$

where SSR is the the sum of Sum of Square Residuals of each leaf in the tree,T counts the number of leaves and α is some constant needs to be determined.We state that we'll pick out the pruned tree with lowest Tree Score.

To find α,we need to do Cross-Validation.We first construct a regression tree with all of the data.And we start with $\alpha = 0$,then increase α until pruning leaves will give us a tree with lower Tree Score.We continue the process until we encounter a tree with only one node(the root) and record these α threasholds.Now we have a sequence of α.

With the α sequence,we can start off Cross-Validation.First look at one specific train-test process.We first construct a sequence of trees using training set,and each tree in the sequence minimizes the Tree Score given each α poping out from the α sequence we've just found.Afterwards,we use these pruned trees to work on test set,pick out the one with smallest SSR and record the α that generates this tree.Completing k-fold Cross-Validation we'll have k such α's and we'll take the average of them to be the α we're seeking for.

Now turn to the decision tree,all we have to do is to replace SSR with Gini impurity so we can prune a decision tree.

2 Random forest bagging

To address the problem of high variance,we can instead use Bootstrap-Aggregation(bagging) technique or say random forest.To impliment this algorithm,we first boostrap from the full data set.for each data selected,we only randomly pick out parts of the features to build a tree(let's call it random tree).And now we have a bunch of randomly generated trees forming a random forest.

The way we use this forest to predict is that we let new data go through all the trees in the forest and they generate a lot of predictions.For classification tree,we count the number of each predicted labels from the forest and understand their ratios taking up the total number of predictions as probalilities.For regression tree,we average over all predictions from all the trees in the forest to get the prediction.

The following is an example of how classification random forest works.

(i)We get a full data set

id	x_0	x_1	x_2	x_3	x_4	y
0	4.3	4.9	4.1	4.7	5.5	0
1	3.9	6.1	5.9	5.5	5.9	0
2	2.7	4.8	4.1	5.0	5.6	0
3	6.6	4.4	4.5	3.9	5.9	1
4	6.5	2.9	4.7	4.6	6.1	1
5	2.7	6.7	4.2	5.3	4.8	1

(ii)boostrap from full data set and randomly assign each member with parts of the features

id
2
0
2
4
5
5

x_0, x_1

id
2
1
3
1
4
4

x_2, x_3

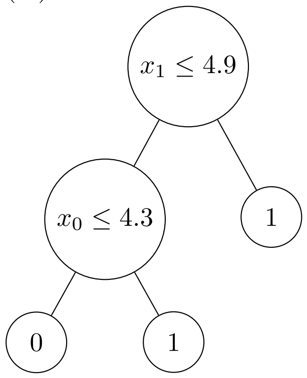
id
4
1
3
0
0
2

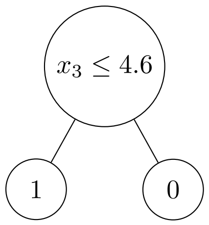
x_2, x_4

id
3
3
2
5
1
2

x_1, x_3

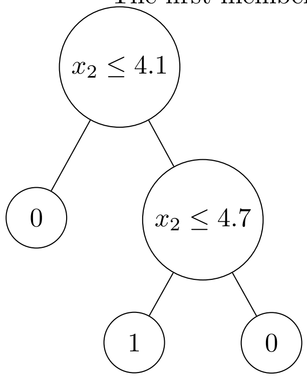
(iii)buil tree for each member

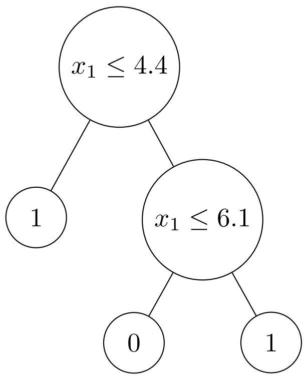




The second member

The first member





The third member

The fourth member

(iv)make prediction for a new data
Now we have a new data that reads

x_0	x_1	x_2	x_3	x_4
5.3	4.2	4.7	5.1	5.4

The random forest gives out predictions with 3 one's and 1 zero.So we believe this new data should be labelled as 1.

3 XGboost

Another way to alleviate high variance is to apply XGboost.

3.1 Regression part

Before going futher,we need to define a quantity called Similarity Score

$$Similarity\ Score = \frac{SR,S}{Number\ of\ Residuals + \lambda}$$
$$with$$
$$SR,S = (Sum\ of\ Residuals)^2$$

where λ is aregularization parameter.And the residual we define here is the difference between the observed and predicted value. For a regression problem,our zeroth prediction would be anything but by default is 0.5.Now we update our prediction values by building a tree using Gain that is generated from while extending a node.We state that

$$Gain = sum\ of\ Similarity\ Scores\ of\ extended\ childnodes - Similarity\ Score\ of\ current\ node$$

Just like how we did in building a tree using Tree Score,we go through exactly the same strategy except we now reckon the higher Gain indicates the better spliting.After we build the tree,we may do some pruning to reduce high variance.

Now say we've finished building a tree and start to work with it.We define output value for each leaf to be

$$Output\ Value = \frac{Sum\ of\ Residuals}{Number\ of\ Residuals + \lambda}$$

With this definition,we can use the tree to update our predictions.The first iteration of updata goes like

$$prediction = zeroth\ prediction + \eta * (output\ value\ generated\ by\ data\ going\ through\ the\ tree)$$

where η is the learning rate,which is set to be 0.3 by default.

After we make new predictions,we also have new residuals.So we can build a new tree based on the new residuals,repeat the previous process to update the predictions.To sum up,we have the update rule

$$y_{i+1}^{(j)} = y_i^{(j)} + \eta * ([output\ value]^{(j)}\ generated\ by\ data\ going\ through\ the\ i^{th}\ tree)$$

A concrete example goes as follow

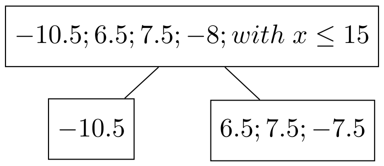
(i)we have four collected data points:(12.5,-10),(17.5,7),(28.5,8),(31.5,-7.5)

(ii)zeroth prediction is 0.5,so the residuals are -10.5,6.5,7.5,-8,for the sake of convenience,we also set λ=0.

(iii)the root of the tree contains 4 datas with similarity score calculated from the residuals is 4.

$$\boxed{\text{-10.5;6.5;7.5;-8}}$$

(iv)we branch out from the root and find that setting the root with condiction $x \leq 15$ where the Gain is 120.33.



Actually,we compute the Gain by first compute the childnodes' Similarity Scores.

$$Left_{Similarity\ Score} = \frac{(-10.5)^2}{1+0} = 110.25$$

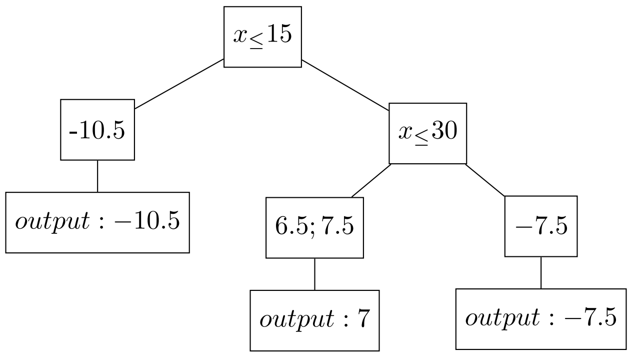
and

$$Right_{Similarity\ Score} = \frac{(6.5+7.5-7.5)^2}{3+0} = 14.08$$

So the Gain is

$$Gain = Left_{Similarity\ Score} + Right_{Similarity\ Score} - Fathernode_{Similarity\ Score} = 120.33$$

(v)Continue the process until we finish the whole tree with output value



(vi)we can now update the predictions using this tree.By the update rule,we have

$$\begin{aligned}
x_1 = 12.5 &\ :\ 0.5 \to 0.5 + 0.3 * (-10.5) = -10 \\
x_2 = 17.5 &\ :\ 0.5 \to 0.5 + 0.3 * 7 = 2.6 \\
x_3 = 28.5 &\ :\ 0.5 \to 0.5 + 0.3 * 7 = 2.6 \\
x_4 = 31.5 &\ :\ 0.5 \to 0.5 + 0.3 * (-7.5) = -1.75
\end{aligned}$$

One can see that the new predictions are closer to the observed values than the old predicitons.

(vii)With new predictions,we can calculate the new residuals,and they turn out to be 0,4.4,5.4,-5.75.According to these new residuals,we can update the decision tree as well.(Note that we now start with the following root)

$$\boxed{0;4.4;5.4;-5.75}$$

3.2 Classification part

With the fundament of Regression part,classification will be simplier.All that we need to do is to change the branching out standard,ouput values of leaves and update rule.Similarity Score now becomes

$$SimScr = \frac{(\sum Residuals)^2}{\sum[Previous\ Probability_i \times (1 - Previous\ Probability_i)] + \lambda}$$

By saying $Previous\ Probability_i$ we refers that the last-time prediction of i^{th} sample(We know that the prediction in classification problem is understood as probability).

Output Value becomes

$$O.V. = \frac{\sum Residuals}{\sum[Previous\ Probability_i \times (1 - Previous\ Probability_i)] + \lambda}$$

However,the output values now cannot be simlpily understood as probability but logrithm of the odds(i.e. log(odds)).The reason for this is that we wish to plug the output value into sigmoid function to become probability,so the output value has better be logrithm of the odds.Our default zeroth prediction which should've been $probability = 0.5$ is now $log(odds) = log(1) = 0$.

With these results,the update rule now reads

$$[log(odds)]^{(i)} \to [log(odds)]^{(i)} + \eta * [Output\ Value]^{(i)}\ from\ the\ current\ tree$$

3.3 Mathematical details

We strike to find out the mathematical principle behide XGboost like the reason we choose Similarity Score in that way.

3.3.1 Regression

While building a regression tree we want to maximize every spliting of nodes by minimizing the cost function L in the current node

$$L = \frac{1}{2}\sum_{i=1}^{n}(p_i - y^{(i)})^2 + \frac{1}{2}\lambda O^2$$

where p is the prediction,O is Output Value,λ is a regularization parameter and n counts the number of samples in the current node.Note that the prediction is equal to the last prediction added up with Output Value,that is,$p_i = p_{i0}+O$.Now we expend L at $p0$ to second order,so we have

$$L \approx L(p_0) + \frac{\partial L}{\partial p}\Big|_{p=p_0} O + \frac{1}{2}\frac{\partial^2 L}{\partial^2 p}\Big|_{p=p_0} O^2 + \frac{1}{2}\lambda O^2$$

where we've already substitute $p - p_0 = O$.We can explictly find out that

$$\begin{aligned}\frac{\partial L}{\partial p}\Big|_{p=p_0} &= \sum_{i=1}^{n}(p_{i0} - y^{(i)}) = -\ Sum\ of\ Residuals \\ \frac{\partial^2 L}{\partial^2 p}\Big|_{p=p_0} &= n = Number\ of\ Residuals\end{aligned}$$

The cost function reaches its minimum at

$$\begin{aligned}O =& - \frac{\frac{\partial L}{\partial p}\Big|_{p=p_0}}{\frac{\partial^2 L}{\partial^2 p}\Big|_{p=p_0} + \lambda} \\ =& \frac{Sum\ of\ Residuals}{Number\ of\ Residuals + \lambda}\end{aligned}$$

Which is identical to the previous statement.And the minimum of cost function is

$$\begin{aligned}L_{min} =& L(p_0) - \frac{1}{2}\frac{(\frac{\partial L}{\partial p}\Big|_{p=p_0})^2}{\frac{\partial^2 L}{\partial^2 p}\Big|_{p=p_0} + \lambda} \\ =& \ - \frac{(Sum\ of\ Residuals)^2}{Number\ of\ Residuals + \lambda}\end{aligned}$$

where we've dropped out irrelevant parts.Now we define Similarity Score=$-L$ So minimizing L is same with maximizing Similarity Score and this conclusion coincides with previous assertion.

3.3.2 Classification

Classification part is almost identical to regression part except that we need to change the cost function and now the cost function is found to be

$$L = \sum_{i=1}^{n} -(y^{(i)}log(p_i) + (1 - y^{(i)})log(1 - p_i)) + \frac{1}{2}\lambda O^2$$

By convention,we'd like to use $log(odds)$ as variable,so we substitute that

$$p = sigmoid(log(odds)) = \frac{1}{1 + exp(-log(odds))}$$

and expand L at $[log(odds)]_0$ to seconde order using relationship $log(odds)_i = [log(odds)]_{i0} + O$.Therefore,we get

$$\begin{aligned}\frac{\partial L}{\partial log(odds)}\Big|_{log(odds)=[log(odds)]_0} &= \sum_{i=1}^{n}(\frac{1}{1+e^{-[log(odds)]_{i0}}} - y^{(i)}) \\ &= \sum_{i=1}^{n}(p_{i0} - y^{(i)}) \\ &= -\ Sum\ of\ Residuals\end{aligned}$$

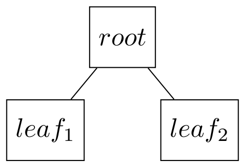
$$\begin{aligned}\frac{\partial^2 L}{\partial^2 log(odds)}\Big|_{log(odds)=[log(odds)]_0} &= \sum_{i=1}^{n}(\frac{1}{1+e^{-[log(odds)]_{i0}}}(1 - \frac{1}{1+e^{-[log(odds)]_{i0}}})) \\ &= \sum_{i=1}^{n}(p_{i0}(1-p_{i0}) + \lambda) \\ &= Previous\ Probability * (1 - Previous\ Probability) + \lambda\end{aligned}$$

Same as derivation in regression part,we can find out the output value and Similarity Score are

$$O = \frac{Sum\ of\ Residuals}{Previous\ Probability * (1 - Previous\ Probability) + \lambda}$$
$$SimScr = \frac{(Sum\ of\ Residuals)^2}{Previous\ Probability * (1 - Previous\ Probability) + \lambda}$$

4 Adaboost

Another common way to improve decision tree is to impliment adaboost.We now use stumps to vote(not like the in case of random forest where we use trees to vote) and each stump has different weight on its vote.In adaboost,order of creating stumps matters not like the case in random frost.(Note:stump is a short tree with only two leaves.)



above is a stump

4.1 classification

The following demenstrates how a bunch of stumps are created (i)assign each sample with sample weight and initialize each of them as

$$\frac{1}{Total\ Number\ of\ Samples}$$

Now we start to find the stump that best classifies the samples according to Gini index.Then we calculate the total error of this stump

$$Totoal\ error = sum\ of\ weights\ over\ all\ misclassified\ samples$$

Also,for this stump,we define amount of say that will play a role in voting

$$Amount\ of\ say = \frac{1}{2}log(\frac{1 - Total\ Error}{Total\ Error})$$

(ii)Modify the sample weights.According to the stump we picked in step (i),We define

$$New\ Misclassified\ Sample\ Weight = Sample\ Weight \times exp(Amount\ of\ Say)$$
$$New\ Correctly\ classified\ Sample\ Weight = Sample\ Weight \times exp(-Amount\ of\ Say)$$

Then we normalize the new sample weights.(By doing these,for the well performing stump(with positive amount of say),we increase the misclassified sample weights and decrese the correctly classified sample weights,on the other hand for the terribly performing stump(with negative amount of say),we increase the the correctly classified sample weights and decrese the misclassified sample weights.Therefore in next stage we stimulate the stumps the better classify the data(or say with higher accuracy).)

(iii)Duplicate new sample sets the same size according to the new sample weights.Take new weights as distribution(like cdf) and pick random number drawn from $U(0,1)$ to determine which sample will go to the new sample set.

For instance,we have a sample set with sample weights

id	*weight*
0	0.07
1	0.07
2	0.07
3	0.58
4	0.07
5	0.07
6	0.07

Say we generate a number 0.03,then we choose sample id 0 as $0.03 \leq 0.07$ or we generate a number 0.18,then we choose sample id 2 as $0.18 > 0.07 + 0.07$ and $0.18 \leq 0.07 + 0.07 + 0.07$.

(iv)use the new generated samples to go through step (i) to step (iii) and record the stumps along with their amount of says we pick each round and now we have a bunch of stumps.

(v)Calculate sums of amount of say over stumps with same output and pick out the output category with max sum of amount of say as prediction for new data.

4.2 regression

To impliment adaboost in regression tree,we only need to update some details from classification tree.

(i)Initialization of sample weights is still unchanged.

(ii)find a best tree according to MSE(note that we are not only confined to stump,instead,we can use trees)

(iii)for the tree we pick define sample error for each sample

$$L_i = |y^{(i)} - p^{(i)}|$$

where $p^{(i)}$ is prediction of i^{th} sample by the tree.Find the max error L_{max} and define relative error $\tilde{L}_i$for each sample

$$\tilde{L}_i = \frac{L_i}{L_{max}}$$

so that we can define the weighted error $\bar{L}$

$$\bar{L} = \sum_{i=1}^{m} \tilde{L}_i \times sample\ weight\ of\ i^{th} sample$$

The amount of say of each tree will now be replaced by weight w

$$w = \frac{\bar{L}}{1 - \bar{L}}$$

(iv)The update rule for i^{th} sample weight will be

$$New\ Sample\ Weight = Sample\ Weight \times w^{1-\tilde{L}_i}$$

Then normalize the sample weight as usual.

(v)The prediction will be

$$\sum_{s=1}^{n} w_s T_s(x)$$

where w_s is the weight of $tree_s$ and $T_s(x)$ is the output of $tree_s$

5 Gradient boost

Updating