NPTEL

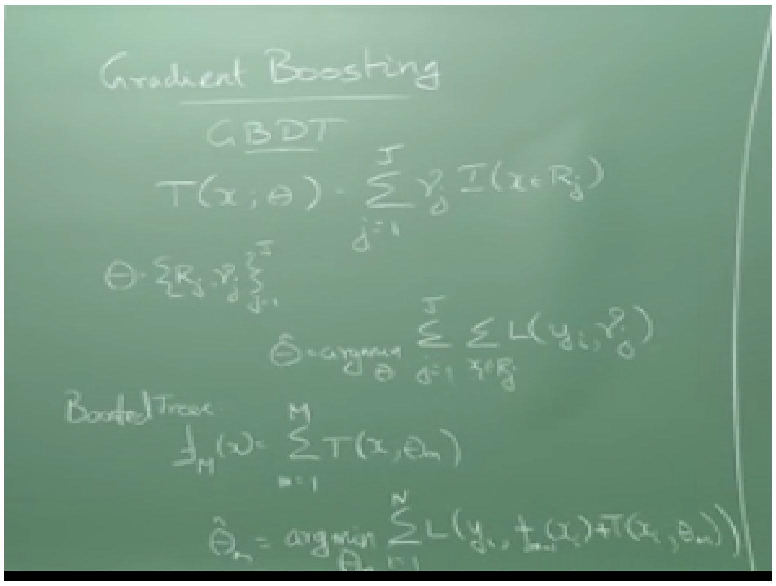
NPTEL ONLINE CERTIFICATION COURSE

Introduction to Machine Learning

Lecture-61
Gradient Boosting

Prof. Balaraman Ravindran
Computer Science and Engineering
Indian Institute of Technology Madras

(Refer Slide Time: 00:16)



Gradient Boosting

GBDT

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$$

$$\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$$

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$$

Boosted Trees

$$f_M(x) = \sum_{m=1}^{M} T(x, \Theta_m)$$

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m))$$

Right, so I want to talk to you about the interesting idea known as gradient boosting. So all of you remember what boosting is about right. What is boosting? What is bagging? Boosting yeah, so boosting is specifically a stage wise process where at every stage you try to boost the classifier from the previous state says that the error is minimized right, error is reduce not necessarily minimize, but the error is reduced right.

So that is a characteristic of boosting, so at every stage you have to look at the errors from the previous stage and you are trying to reduce that right. So we looked at ADABOOST right, one of the most popular boosting algorithms. And then I told you that ADABOOST uses exponential loss and that it is related to the logistic loss, and so you can actually use the logistic loss function and derive a boosting algorithm which is called logit boost right.

But it is very similar properties to ADABOOST, but ADABOOST is more popular especially from analysis point of view and things like this, because it is not really nice properties okay. There is yet another approach to boosting that is gaining a lot of currency recently it is called a gradient boosting, at not well recently would mean in the last decade or so right, compared to ADABOOST which is several decades old right.

So sometimes they even call it gradient boosted decision trees right, gradient boosted decision trees, because you use this specifically in conjunction with trees right. And in fact in many applications now gradient boosted trees are actually getting hard to beat right. And so, we just reintroduced some notations that you might have forgotten right. So I is an identity function which is 1 if X belongs to Rj is 0 otherwise right.

And so, this is summing over all regions so R12 RJ and γj is essentially the output I am going to produce if x lies in RG right, this is the regression tree thinks. So what is my θ here it is all the RJ is the specification of the RJ and the γJ is for each of those region. So that is my θ. And typically we pick some loss function if it is regression it is going to be squared loss and then right.

So I look at the loss incurred when the actual output is YI and the output I am giving you is γJ, so for all data points X that belong to a region RJ the output will be γJ, so this is essentially the loss there and sum this over all regions and this is the rectum just recapping the decision trees for you right. And then we looked at greedy methods for finding RJ right, and given an RJ we knew how to fit γJ right, so given that we looked at some greedy search methods right.

So you can do, you can do boosting with trees also just like you did boosting with other classifiers you can do boosting with trees right. So I have M trees so essentially it is taken some of the output of all the M trees that gives me my boosted tree right, remember that I mean this is not a single tree okay it is now a forest, then I have a correction of trees a collection of trees is a forest right.

So it is actually a forest, so I basically do that and the difference here is can people at the back see this right. So this is essentially when I find the parameters for the m^{th} tree right, so I am going to look at the classifier or the predictor that is formed by the first M-1 trees right. And then I am going to find that tree okay, whose output I will add to this predictor right and you search for computing the loss right.

So for every data point in my training data right I look at the way I look at the output produced by the m-1stage 3, I look at the value that is added by the mth tree so this is the output produced by the mth tree I look at the value added by the mth tree to that right. And then I will compute the loss function okay make sense yeah.

So the basic area is that every point this is just forward stage wise addition that we like whatever we did before introducing boosting right that is exactly that. So now this becomes boosting because I am explicitly trying to figure out what the residual error is from here okay, and trying to adjust for the residual error using my tree the new tree I am learning right. So when will it be the residual error, when it is a regression task and squared error is my metric right.

So and the loss function is squared error right, and I am so trying to solve the regression problem right, then essentially what I will have to do here is take the residual error. So whatever the tree does not explain the m-1 stage tree whatever that does not explained, so that error I will have to explain using this right. So if you think about what we are doing here, so you will first build one tree to predict your output as best as possible right.

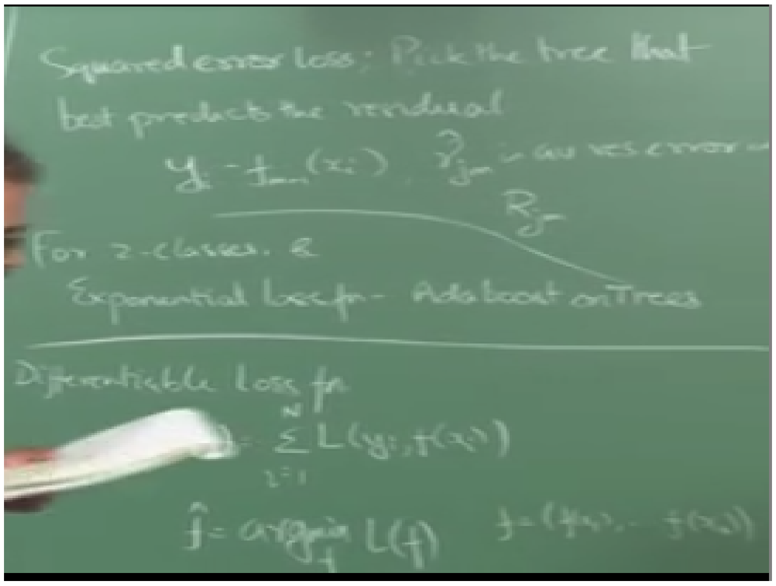
The predictor function as best as possible will build a tree, then what you will do is okay you will take the residual of that, build another tree that predicts the residual as well as possible and add the output to this okay. And then take the combined thing find the residual of that build the third tree which will predict the residual and add it back to this and so on so forth, you just keep doing this right.

So that is essentially what boosting increase means, so you still not come to the gradient boosting part okay. So finally it will look very similar to what I am telling you now, but we have still not come to that part yet right. So as with regular decision tree learning given the RJ's right finding the γJ’s is easy given the RJ’s finding the γJ’s is easy right. But the problem is finding the RJ's in general it becomes a little tricky finding the region's becomes a little tricky, because I have to take into account the other tree's output also right, in general right.

But I am talking about squared error it is very easy, because things nicely decouple right when I am talking about squared error I do not have to worry about FM-1 after I compute the residual right. The residual could have been generated by any classifier any regress righty, it does not have I do not even have to worry about the fact what generated the residual was actually a tree right, I do not have to worry about it, all I need is just the residual.

The residual then becomes any function right, so with squared error boosting becomes just like learning a series of decision trees, nothing special about it. But if you have other kinds of loss functions then we will have to worry about how to accommodate it, but at least in this case of squared error loss okay.

(Refer Slide Time: 10:24)



Squared error loss; Pick the tree that best predicts the residual

$y_i - f_{m-1}(x_i)$, $\hat{\gamma}_{jm}$ = ave res error in R_{jm}

For 2-classes. & Exponential loss fn - Adaboost on Trees

Differentiable loss fn

$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i))$

$\hat{f} = \arg\min_{f} L(f)$ $f = (f(x_1), \dots f(x_N))$

So that is essentially your target function right, and what will be the γhat that you will need just be the average residual error in the Jth region right. Any questions, so in fact there is another case where it becomes simple which is essentially, so for two class problems and exponential loss functions what you think we get, it becomes exactly the same as doing ADABOOST with trees okay, the two class problems.

But, so it turns out there are tricky things here right, so if there is if it is a multi-class problem okay then things do not decouple as nicely. So if you have two class problems and your loss function is exponential loss okay you can show that this is essentially the same the computation that we are trying to do here right that minimization everything that we are trying to do here essentially reduce to the same solution that you get if you did the ADABOOST derivation on decision trees okay.

But these are the two cases where this thing simplifies that for example, if you are trying to use deviance as a loss function then things do not decouple this easily okay. So these are things that you have to keep in mind, so this is one part this is essentially telling you how to do boosting with trees the regular way okay. So let us look at something else now, so I suppose I have some I have some differentiable loss function some loss function which is which I can take the derivative of right.

So if I want to do, so if you want to take a numerical approach to optimizing this kind of a loss function typically what will I end up doing I will start off with some guests for a solution right take the gradient of the loss function with respect to the parameters at that solution point we will number gradient descent right. Then I will compute the gradient and then I will move in the opposite direction of the gradient then I will move a small step in the opposite direction of the gradient go to a new place and compute the gradient again and then move again and so on so forth until I converge to the right answer right.

So if you think about what this is doing this is something like okay take initial solution okay, then I add another solution to it which is essentially the gradient times something then I add something more to it, so I will add something so essentially the solution I am computing finally this is a sequence of additions that they have done on the basic solution I started off with right.

So even though one way of thinking about it is at every point I give you a parameter vector, but the parameter vector itself is composed of a sequence of additions. So I can think of it as first starting off with initial guess for my parameters, then adding something more to it, then adding something more to it, then adding something more to it, and adding something more to it, till I come to the final answer right.

So that is one way of thinking about it, so let us try and write this down a little formally. So now I am going to see what I can do about this f is for the time being ignore the tree constraints I will come back to the trees later, time being let us ignore the tree constraints right. So what I really need is, so just when I am trying to do this in a numerical fashion I am just operating with a single data set right.

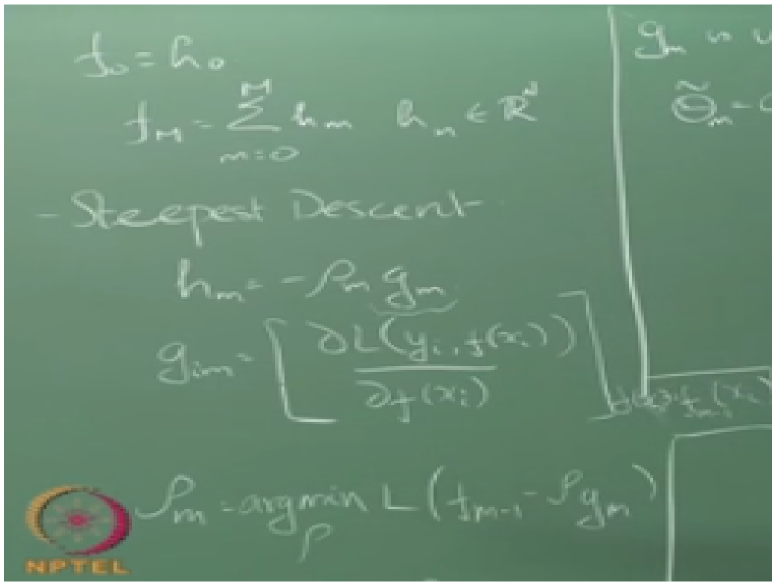
So when I say f what I really have looking at is a point in RN, so what does that mean let f really means okay what is the value of F at X1, what is the value of F at X2, X3, X4 all the way up till Xn so when I impose constraints on F, then I will be restricting the kind of vectors I will see. But in general when I am talking about F in this context I just mean like an n-dimensional vector right. So you can think of it as a point in n dimensional space right.

So typically what you do is you start off with some solution right F0 you start off with some solution let us call it H0 right. So you can think of it like this I start somewhere here that is my F0 right. And then, I compute the gradient and move in the opposite direction right, so I take a small step in this direction so I come here that gives me a new set of parameters right. So this is 1 θ, this is another set of θ and this will give me another F right.

But instead of saying that this will give me another F, so I am going to say that okay this is one F, then I add something to it right, so that gives me the second F. So what I am actually computing in every step is the amount that I add to the previous solution to derive my new solution okay. So I am confabulating θ and F here, so what I have here is θ corresponding to every θ I have here there will be a F right corresponding to every parameter setting I will have that will be output vector F right, when I change θ this values will change right.

So when I am here I have one solution right, so when I want to go to here that essentially means that whatever F vector I have here I will have to change each of those coordinates by some value so that I will end up here right, those values I change the coordinates by it is my H vector right, is it clear what we are doing here.

(Refer Slide Time: 18:58)



$f_0 = h_0$

$f_M = \sum_{m=0}^{M} h_m \quad h_m \in \mathbb{R}^N$

- Steepest Descent

$h_m = -\rho_m \underbrace{g_m}$

$g_{im} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i) = f_{m-1}(x_i)}$

$\rho_m = \arg\min_{\rho} L(f_{m-1} - \rho g_m)$

g_m is

$\tilde{\theta}_m =$

So what is the normal mechanism by which we will do this right, so I have been using the same example so far right, so steepest descent is something that will pop up right even the other ways of doing this optimization right. But steepest descent is the one that we are all familiar with the one that I have been using as an example here so far right. Since I have not chosen any arbitrary parameterizations to form a θ right for the F, I have not chosen any parameterization θ or anything right.

So the parameters of F are the output set each one of the input points see the way I characterize my function F is looking at okay, what will be the value of F at X1, what will be the value of F at X2 and so on so forth right, I do not have any other parameterization for it. So instead of finding your δL/δθ you actually find that I am writing it as δL/δF okay. So F(xi) is essentially the output of F at Xi and what is F here, it is Fm-1, because I am determining the M stage I am looking at the m-1 guess for my function right.

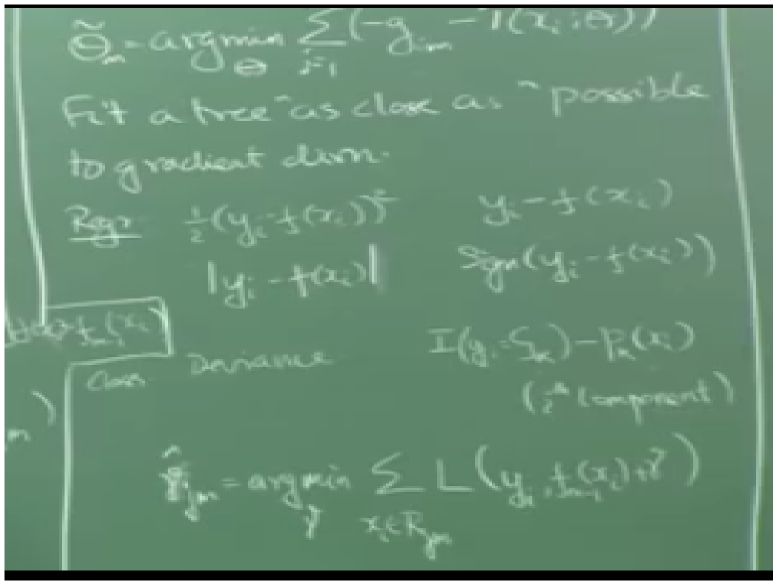
So the steepest descent direction would be saying that okay, so GM is the direction in which you have to move, because that gives me the direction in which the or rather -GM is the direction I have to move, because that gives the direction of steepest descent. And ρm gives me the step size I have to take in that direction how larger step I can take in the direction. So how is flow I am determined you should look very familiar right.

This is exactly how we did the ADABOOST derivation. Now people are think about it really ADABOOST derivation like this yes, we did go back and think about it okay. So very similar not exactly the same thing, but very, very similar the exact the same steps that we did right we first found out which way we have to change it right, and then we found out what the step size should be, and the way we did it was okay, I have already have a classified till m-1 stage okay, what should I do at the M stage.

So as to minimize the error, so this is exactly what we did the idea behind each of the steps are same the mechanics might have been slightly different right. So once I get this then I do okay, right is it clear people are doing so far right. So whatever we are trying to do right, is nothing so there are two different parts here okay, if you people are getting confused so the first part here talked about boosting trees okay, the second part here talks about taking some differentiable loss function and trying to do some kind of a stage wise process on it okay.

Now I just took your normal gradient descent procedure and told you that you can think of it as a stage wise process, I guess like we did with boosting we can think of it this additive model right. So whatever you will learn here right, so now the thing is how we connect up the two will somehow account for that later. I do not want to erase anything from the board because what we are doing right now is connecting the two parts right. So I do not want a raise anything from the board, so you can see both that and this well we are looking at this right.

(Refer Slide Time: 23:48)



$\tilde{\theta}_m = \arg\min_{\theta} \sum_{i=1} (-g_{im} - T(x_i;\theta))$

Fit a tree as close as possible to gradient dirn.

Regr: $\frac{1}{2}(y_i - f(x_i))^2$ $\quad y_i - f(x_i)$

$|y_i - f(x_i)|$ $\quad \text{sgn}(y_i - f(x_i))$

Class Deviance $\quad I(y_i = G_k) - p_k(x_i)$

(i^{th} component)

$\hat{\gamma}_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$

So far so GM, GM is some kind of an unconstrained maximal descent direction I do not have any constraints on f for anything else right what is the maximal descent direction and essentially get GM. So now what we are going to do is to say that hey, all of this is nice, but I would really like some parametric forms for what I am doing right otherwise things become too complicated. So what I am going to do is I am going to fit a tree.

So what I want to know is GM right, so if you think about it, so I really need to compute GM and instead of doing this in this arbitrary unconstrained form I am going to build a tree that approximates GM as closely as possible okay. So you should note something here what is it you should not, so for all this while I have been very carefully writing L for the loss function well I am trying to keep this as generic as possible right.

But here I wrote a squared error loss function, because it does not matter what is the problem that I am trying to solve, it does not matter what this loss function is okay, because what I am trying to solve now is trying to approximate a vector I trying to approximate a direction right by a tree essentially I am always solving a regression problem here right. So if you think about it GIM is going to be some kind of a vector right GIM will be some kind of a vector all I am trying to do is predict the value of that vector component right.

So I am just doing regression regardless of whether my original problem was a classification problem or a regression problem or what not, I could use any loss function here and I could use any loss function here this is the crucial difference you should appreciate right, for the actual or actually solving the problem that I could be using a different loss function here okay. But when I am building the MX stage decision tree all I will be doing is regression, because all I need to predict is what is that particular gradient descent direction for that input value Xi right.

So this is what I am going to do, how am I going to go about doing this. Well it depends on what this loss function is okay, so is this loss function, so if the problem I am solving is a regression, this is what I am solving, this is the squared error loss function okay, this is not this okay, this is that. So if this L is squared error okay, then what do I get here is essentially what is this, give this is GM essentially the gradient of the loss with respect F(xi) right, the gradient of this with respect to F(xi) is essentially the residual Yi-F(xi) right.

So this is basically –GF if you would think correct. So now what happens if I am doing regression with squared error loss function and I am trying to do this gradient boosting right. So I am trying to build a new tree that predicts the direction of the gradient what do I end up doing, I end up predicting this is the residual right. So I end up predicting the residual and here what we said, if you are doing the squared error loss pick the tree that best predicts the residual.

So that was derived from just the basics of boosting regular boosting. Well here I am talking about a technique that can do boosting on trees regardless of what is the underlying loss function right, but it does boosting using trees okay. So that is a cool thing about gradient boosting right, so you always are solving a regression problem as far as the tree is concerned, and solving regulation problems using trees is very easy right, your solving regression problems using tree is always regardless of this loss function.

If you remember when we are deriving this boosting update I said four squared error and for two class exponential loss the boosting form is easy right. But now if you busy doing the gradient formulation of it okay regardless of what you are doing with the loss function you can still do boosting with trees. So that is why gradient boosting decision trees have become very popular now, because you can do all kinds of cool stuff with it.

So what about this right, and suppose you are doing classification let us take deviance as the loss function right, we will remember deviance, we looked at deviance multiple times right, and turns out that. So what is this, if the ith class is GK, then it will be 1 minus that the data point Xi is in class K, the probability of data point XK belonging to class K. So it will be 1 minus that, and if the actual class is not K right, then it will be minus probability of Xi belonging to class K okay.

So this is like the ith component of it, so ith component of –gi okay. So again what I have to do, I have to take this expression plug it in here and do regression again, I will take this expression plug it in here and do regression right. So all you need to do is figure out what is the derivative of your loss function with respect here F(xi) right, and then once you find out the derivative you just do the regression with respect to that for each stage in your decision tree okay. So what will be my γJ this is γJ, such γ, γJM, what will be γJM? So earlier we said it will be just the average residual error in RJM right.

So in this case it is going to be so once I have found out what the actual regions are right, once I have found out what the region. So this is going to give me the regions right, so once I find out what the actual regions are I will do the following. So what have we done here okay. So earlier when we are finding out the, when we are building decision trees right, so the way we found out the regions right.

If you remember the way we found out the regions was we postulated a split point right, and then for that split point we figure out what is the best γ in both half of the tree right, and then we took a value for that, and then we kept looking at all the possible split points right, the splitting variables and split points for all the possible combinations and for each one of them we evaluated what the resulting residual error would be right.

And based on that we pick the split point, so here we are doing something different when we are splitting picking the split point right, we are going to pick the split point such that the residual error in predicting the gradient is minimized right. I am only predicting the gradient here, the residual error in predicting the gradient is minimized. But when I finally decide what is the output that I am going to give in each of the final regions right.

So in the regular decision tree building by the time I come to that point I would have already solved the optimization problem right. So I know what is the solution that I have to give and it is the one same thing which I use for splitting criteria. But in this case when I finally give the outputs I am going to look at the loss function right of data points that fall in that region look at what is already existing that is Fm-1 of Xi right.

And look at what I need to add to bring up the output, so that the loss is minimized. So whatever is the loss function, so when I am doing the loss function here I will no longer be using the squared loss I will be using one of these, I mean this is quite loss I could be using the absolute gloss or I could be using deviance or whatever is the measurement that I want right, I will use this loss function the loss function I use there, I will use that here in order to figure out the outputs okay.

So you let this sink in a little bit, so it is actually a pretty cool idea right. So I have somehow come up with a mechanism where I can use decision trees in a very powerful way, because finding regions while I do regression is very easy with decision trees, because we know how to use squared loss and there are lots of tricks and optimizations that you can do when you are searching through with square loss right.

Now we looked at some of them, but there are many other things that you could do so what I am going to do is for all my tree growing part right I am just going to use the regression trick right, and I finally have to give an output at that point I will use whatever is the true loss function I want right, that is why it is called gradient boosting. So I use the gradient at every point to boost my performance right.

So the way I fit my, so if we think about it this might not necessarily be the most ideal way of doing things why is that right. So if I want to predict γhat JM that truly minimizes forget about the RJ's right, that truly minimizes the loss function I might want to actually split the space in a different way right. But I amusing the gradient information to split the space, then whatever splits I get whatever regions I get by using the gradient information I am using the same regions in order to reduce the error also right.

It is fine as long as I am unconstrained right, as soon as you put in the constraints of trees it is not entirely clear that the tree that you want for getting the representing the best γhat is the tree that you want for representing the best GM it is a very, very settle point you have to think about it a little bit. But more often than not it turns out to be fine okay, but there is really no guarantee that the best three for predicting GM is the best three for representing your γ's okay.

That is two slightly different things, but still it turns out to be fine right. So all of this discussion collapses if you move to L being the residual, I mean the squared error right, L is squared error everything looks the same you already solve it here is a very easy thing to do. But what I want to point out is, it essentially the same squared error mechanism you can build boosted trees for any loss function that you want, it can be regression, it can be classification.

So whatever it is you can build a decision tree. So one thing that you have to be careful about is that you do not over fit the things any time right. So you have to be careful about not over fitting the data, because you are only working with the n data points always right, you can build a very complex tree that will try to over fit the gradient for just the training data right.

So that is not a good idea, so try to keep your tree down complexity of your tree down. In fact quite often people choose the size of the tree a priori. And then, you might actually end up adding more trees than necessary because he chose too small a tree, but at least you will avoid over fitting right. And so that is it for gradient boosting.

IIT Madras Production

Funded by
Department of Higher Education
Ministry of Human Resource Development
Government of India

www.nptel.ac.in

Copyrights Reserved