NPTEL

NPTEL ONLINE CERTIFICATION COURSE

Introduction to Machine Learning

Lecture-60
Boosting

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

Okay, so with one of the most popular and most some sense mind blowing thing with the non-thermal method space right, so boosting in fact the original boosting work original analysis of the boosting work comes from theoretical computer science community not necessarily from an empirical mission learning community right, so therefore they were that they looked at having some oracle that had a probability slightly greater than 0.5 of being correct right, and then they try to see how you can get better and better predictions from somebody who is just above 0.5 okay.

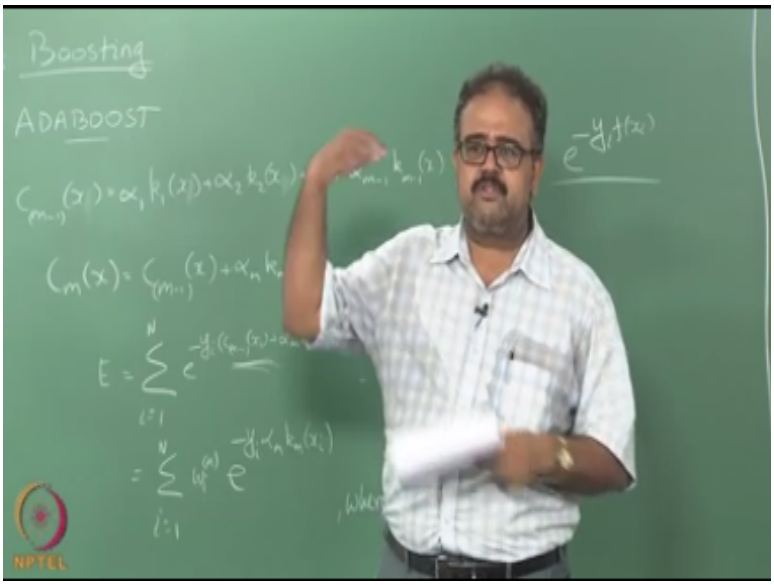
By combining many, many search Oracle's right I can keep improving my accuracy of prediction arbitrary close to 1 right, so that is the amazing part right I start off with each individual predictor as accuracy of 0.5 plus some epsilon just better than random that I can combine a lot of them together and produce something that as accuracy close to 1 right, so this was a very big result that came out earlier and so we are going to look at some kind of simplified version of it, so they remember the goal that distinguishes I mean the main thing that distinguishes boosting from the other methods is that boosting is inherently serial okay.

So boosting is going to build this ensemble classifier in an incremental fashion right, where at each stage I am going to try and explicitly reduce the error produced by the previous stage, so this is something that you have to keep in mind you just cannot write it just cannot come up with some ensemble method and call it a boosting method and I have seen that happen in many papers that I have reviewed people just write something that has multiple classifiers in it so it is boosting, because they have read somewhere that boosting is a very hot area and people papers in boosting get accepted so they come up with any classifier and the any ensemble method and

call it boosting, boosting has this very specific property that at every stage right, you add one more classifier to the existing ensemble right, and this is done in such a fashion as to reduce the error produced by the classifier up till that point okay, makes sense.

Sorry, you get the choice as to what to add next right so that is that you choose it such a way that you minimize the error that they have not at least you reduce the error okay, so not necessarily minimize but you reduce the error of whatever has happened the prediction till that point okay, does it make sense, so that is essentially what boosting is sometimes you can think of it as error boosting, sometimes they call it as error boosting and so on so forth.

(Refer Slide Time: 03:33)



The one very popular and one of the original boosting algorithms is called ADABOOST okay, so let us and it is going to I am going to put up a tutorial for you guys to refer to okay I will use the notation from the tutorial so I will not translate it to the notation in the text book okay, so when you read the textbook you have to do the translation yourself, so that is one of the main problems when you have too many different disciplines contributing to the same field right, machine learning has people from computer vision, people from statistics, people from AI and all other disciplines contributing to it and each one of them brings their own notation to the mix, right.

So it becomes harder to keep track of everything, but currently the I output is necessarily complicating my dress okay, so I am going to denote by C subscript x the (m-1)th stage classifier

okay, that is obtained by basically adding the outputs of all this individual classifies so α_1, α_2 to α_{m-1} right, so I am going to add up these are the weights and k_1 is a classifier that I added in the first stage right, k2 is the classifier added in the second stage and so on so forth and k_{m-1} is a classifier added in the m-1 stage okay.

And then basically I want to produce that okay, yeah, the rest of the class, so there are a couple of things which I should point out here one of the most obvious ways of doing this forget about Erebus one of the most obvious ways of doing this is to say that okay I am going to take this guy right, look at the residual error you know I can think of this as a prediction problem right, and look at the residual error of the predictor right, and then train a classifier km to minimize the residual error, right.

So what will be αm, essentially how to make sure that this whole thing is along the direction of the residual so we talked about this earlier right, when where did we talk about this, forwards stages we ask stage wise or step wise, stage wise, so when we talk about stage wise feature selection we talked about something similar right, so you could think of something along the same lines here instead of thinking of selecting features right, I am just selecting classifiers right.

So I can just take the residual error of c_{m-1} and then use that to train $k_m(x)$ and then add it here right, in fact this can be one, it can be one does not matter because the $k_m(x)$ will actually align itself in the direction of the residual so I can just add it here so it is fine right, so that is a simplest way to do this thing and it is actually a good way to do it if you are doing regression let that make sense and I can take this as they can take the residual error and then train my k_m to actually go in the direction of the residual.

So I can actually do this, so you can get a boosting like algorithm for regression just by training it along the direction of a residual right, but when I am doing classification that is not necessarily the right thing to do so people come up with different kinds of loss functions and then they try to improve the classification, so the loss function we look at is the exponential loss, so people remember the exponential loss, I talked about it when we are doing SVM's is exponential loss okay, $e^{yif(xi)}$.

So we looked at the exponential loss earlier so we will essentially continue with that, so I will sum over all the training points right, so that is the exponential loss for the m^{th} stage classifier

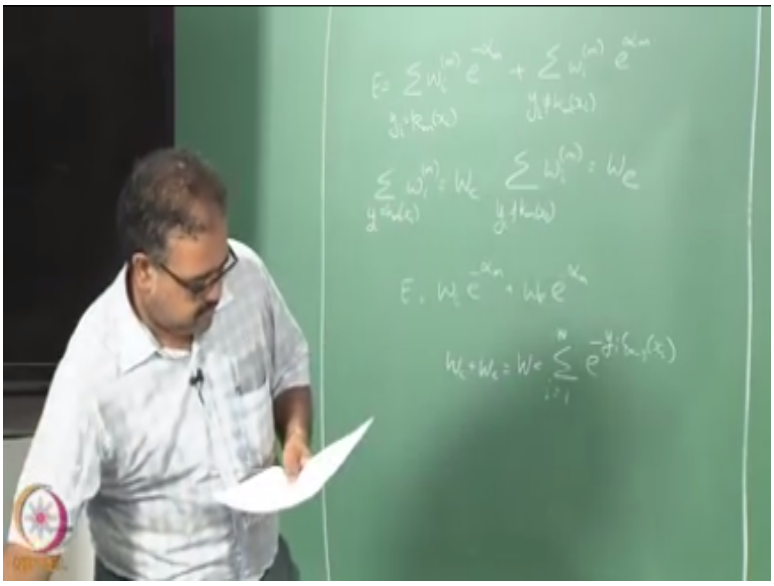
right, people agree with me on that, so that is essentially what I wanted to write right, now expanded the cm and I have written it as this expression in the bracket here. Yet, can people see me at the back I see not me but the I am kind of hard to miss right, that makes sense okay great.

So this thing we already know right, so there is no control we have over that that thing we already know that is given to us all we need to find is α_m and k_m right, so I am going to rewrite this as what is that, that is the last function we are going to be using like that is exponential loss function so for classification we looked at, if you remember we looked at the different loss functions and when we looked at hinge loss right, and I said exponential loss is one of the loss functions and this is how we defined it so essentially I am using that exponential loss function here.

And in fact I mean this was not the way ADABOOST was originally derived okay, ADABOOST was derived in a completely different way and later on about five years after they publish ADABOOST they kind of discovered the connection between this kind of stage ways modeling right, forward are additive stage wise additive modeling and exponential loss they said okay, I can do forward stage wise modeling with an exponential loss function I end up with ADABOOST that connection was discovered five years later.

But now almost always people except in the theory community, in the machine learning community is always introduced like this okay. So where wmi is sorry, m the same thing I wrote here, so the weight of the i^{th} data point at the m^{th} stage right, the weight of the i^{th} data point at the m^{th} stage is essentially $e^{-yicm1(xi)}$ right, so what does this mean what is exactly this expression if you think about it, it is a loss I have incurred on that point x(i) up till the m-1 stage right, that is essentially the right just the loss that I have incurred on the ith data point up till the stage m-1 right.

(Refer Slide Time: 14:03)



Okay, now I am going to break that sum up into two components, so do you think of these two components, no, no, no this is varies say $e\alpha^m$ so when will I get $e^{-\alpha m}$ when I am correctly classified it, when I will get $e^{\alpha m}$ when I am misclassified it so these are all the data points such that right you are the correctly classified data point is all the miss classified data points right, is intuitively you can see where we are going with this, so what is the best classifier that I can find at the m^{th} stage.

Well, obviously the best classifier can find the one for which this $\sum$ is empty the right, and get everything correct classified correctly so that is the best classifier. But now increase the cracks right, remember our classifiers are all weak classifiers and exactly that is a basic assumption we are starting off with right the classifiers are all weak classifiers I can do only slightly better than random, so I have to get nearly half the data points incorrect, right.
So which half should go here which half should come here, and then we can move one data point from to here that here to make it better than half, so which half should go here which half should come here intuitively you tell me, which will incur less penalty, what is small half, it is half man what is small half that will be what clear me, can be more clear as to what is small means, no that is a valid way of interpreting small half tell me, no, no, no, wms right, so all the w's that have a large value should come here because they get $e^{-\alpha}$.

The w's ensemble small value should go there because they get multiplied by $e^{\alpha m}$, so what are w's is a small values the ones that I have correctly classified up till the previous point w is the large value are the ones that I have incorrectly classified up to the previous point, so at the m^{th}

stage what I should be looking at is try to get the data points which I misclassified from the previous stage, try to get them correctly as many as possible right.

So that is essentially the intuition behind ADABOOST, so at every stage what you do is you try to look at the previous stage see which are the data points you misclassified it try to get them correctly in this stage, right and it is okay if you make mistakes on data points that you have correctly classified till the previous stage why is it, okay. And because those classifiers can possibly adjust for it okay, then we will look at how we will actually do this again right.

So I am going to call, so it is all the weights of all the data points I got correct at m^{th} stage right, likewise weight of all the data points I made a mistake on at the m^{th} stage right, so then I can write my es simply as okay, so if you think about it the value of α really does not matter in my choice of k_m right, regardless of the value of α right, regardless of value of αm whatever argument I gave you this no holes right, the idea is to see how much of the weight, the weight you can push to Wc right, and how less of the weight you keep in We.

I mean there is total of weight right, there is some total W right, W is a constant okay, WC +W is a constant, the goal is now to see how much weight you can push into WC as posted, km will be the classifier that rise as to my WC, so how we do this well you can use you can classifier they images that can assign based data point, we discussed very briefly in session the case right, we can assign ways to data points and you can essentially multiply the error that you make on a data point by the corresponding weight, right.

So the error that you make you multiplied by the corresponding weight so that you can use weighted minimize other ways of doing this, so one way people see what I am saying over km right, you see what you are supposed to do to get your k, the k_m is such that maximum weight goes into Wc they are splitting your W into two parts and depending on what data points are making mistakes on right, the data points you do not make mistakes on contributed Wc the data points you make mistakes on contribute of e.

If you want to see how much larger you can Wc and We basic that is the classifier you have to find, before that you use some kind of a payment method, so one way of achieving this is do the following you are saying weights to all the data points now what you do, if you go and sample some of these data points according to their weights, create a new training set by sampling from

this data points are given things according to the weights, so what does this mean points for which the weight is higher you get sample more often into this data sets, points for which the weights are very low I do not even appear in the data set, right.

So the points appears multiple times in the data set then when you are trying to minimize the training error you are likely to get a point correct, so instead of using a directly using a weighted training algorithm people simulate that by sampling from the data weights okay, so what has happened unfortunately because of this I change of tends to compact by bagging and boosting in the minds of people and if you look at some of the data mining text books especially some of the earlier data mining text books exciting and boosting we needed will described in a very similar fashion right, what do you do in bagging whenever you add a new classified.

So in the older textbooks how they describe is that what you do in bagging is every time you generate a new sample you sample uniformly right, with replacement right you with sample replacement and boosting the differences every time we generate a new sample you use the prediction error from the previous thing there is only difference between bagging and boosting right.

But operationally if you think about it so there is only difference between bagging and boosting, but then boosting is inherently serial and then there is this error minimization property right, but that never comes across and people just tend to think of boosting as bagging with the different sampling distribution right, whether it is incorrect at the fundamental principles of the two things are very different okay.

So we have found km now right, so we all know how to find km you do some kind of weighted error minimization you find km, so what is next, what is next we need to find αm right, see regardless of what value of αm you choose the minimizer is the for km is the one that gives you maximum weight into Wc, correct. But then having chosen a km I now have to choose an αm that gives me the error detection, so how do you go about doing that.

(Refer Slide Time: 25:28)

In fact we can do our, so set is equal to 0, so αm is essentially 1/2 ln 1- the error rate it is error rate is essentially the weight of the data points on which you are making a mistake divided by the total weight right, so this is for the km classifier alone right, We is the data points on which the m^{th} classifier alone makes the error not Cm but km correct, so that is what we divided these things into right, so this is the thing where km makes error, so essentially that so the data points on.

So essentially it tells you how good the classifier, if the classifier is really good right not just on the data points that you are interested in but on the entire data set and if the classifier is very good then the weight will be high that is in fact the classifier has an error of 0 what will happen rate will be infinity, because the only classifier you will need right you have a header of 0 on all the data points why do you need other classify just that one is enough right.

But then suppose it has a very high error, error close to 1 where it will be 0 okay, so depending on how good the classifier is this way it will vary okay, and then anything else that you have to do I have found km, I have found αm what do I have to do, I have to change my W's now for the next stage right, so what is my Wi is $e^{-yiCm-1(xi)}$ right, so now it has to become $e^{-yi.cmxi}$ so what is the right best way to do that, this multiply the existing W by $e^{-yi.kmxi}$ right, does it make sense after you have done that you come here okay, I do not erase that part right. So because you need the αm here for your update, so once you find the αm you come back here and change the weights of

all the data points by this amount okay, as it makes sense so that is a plain simple version of ADABOOST okay.

So in fact we can show that the exponential loss function is very closely related to the deviance right, and in fact an equally popular version of boosting called logic boost exists, where we actually use the deviance the logistic function right, the log odds function that we used for logistic regression you can use the same error function and then derive all the update rules that we just did for the exponential loss function you can do the same thing for the logit function the log odds function also and you can come up with similar update rules okay.

So the recent ADABOOST is so popular is because it deals such very simple updates right, if you think about it all the computation you do is okay, you find a classifier that minimizes this weighted the error right then you come back and compute this αm and then you go back and change the weights and then repeat until you are happy with the performance of the total classifier right, and both with basting I mean bagging and boosting the commend both here start basting things anyway if you do both decision trees are very popular classifiers for this, okay.

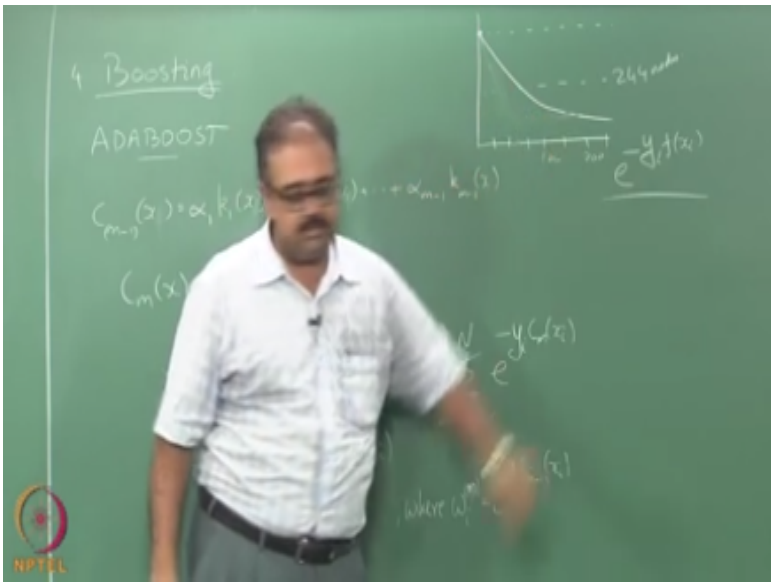
In bagging it seems to make sense right, why you want to bag decision trees they are notoriously unstable so if you want to, if you bag decision trees you get more stable estimates, why would you want to bag, why do you want to boost decision trees, are they weak classifiers. Exactly, so what do you do with decision trees in fact you can do the most extreme thing you can just have one node, just have the root node right, one node what can you do with one node decision tree.

Yeah, that is somewhat like linear right yeah, so somewhat likely linear I agree but people call it decision trees right, so one node decision tree because of the way I choose which feature I pick right, I will use information gain or Gini index or one of those things okay, I will at least take 50% classification otherwise I would not even split right on that 50% will I will be better than 50%.

I will be better than random even if I split on one node right, so I will split on one node and or maybe if the performance is too weak I can perform I can do a two level tree okay, these are called decision stumps, I do not build a full tree but it is like chopped off at a very close to the root right, so one not two levels of the trees though they are v-classifiers and they take very little

time to estimate and I can do many, many, many of these very quickly essentially what I do is, I boost these decision terms okay. In fact there is one result in the book if you look at it right.

(Refer Slide Time: 33:38)



So I do not remember the exact scale on the y-axis but the x-axis is the number of levels of boosting that they do right, and so on so forth, the number of levels of boosting that they do so 100, 200, 300 and so on so forth, so a single stump it gives some performance level at that height okay, just one strum the best single stump gives you a performance there and they trained it on the full data and they get a performance here and this is like a 244 node tree, 244 notes is a fairly complex thee they built and that is the performance that they get right.

And then they did boosting the start here obviously with a single node right, and then they do boosting and then they find that the tree the performance just keeps improving as I do ADABOOST. Remember and these are all single node trees okay, they are all single node trees and so essentially by the when they reach 100that means they have only 100 nodes basically and

they are way better than the 244 nodes that you get with a singletree right, and they reach 244 notes they are like more than twice as good as the single tree they built with 244 notes.

Because the objective function you are minimizing is something very, very different right at every stage you are changing the function and you are focusing your efforts on actually getting to the harder parts of the space right, so that essentially it is little magical. In fact this is more dramatic is it something like this but look at look at the book crippled for the exact figure right, so this is really amazing posting is very powerful and in fact I talked briefly about random forest in the next class which is you do not even have to do any decision making right, you do things randomly but then do a lot of them right that is also very powerful.

So random forest is not a boosting technique by the way random forest is a bagging technique right but then that is also very powerful.

IIT Madras Production

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

www.nptel.ac.in