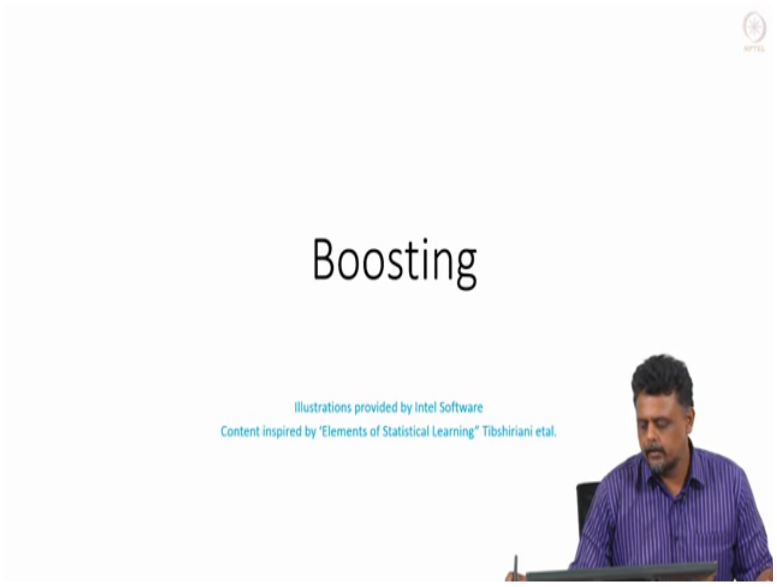
Machine Learning for Engineering and Science Applications
Professor Dr. Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology, Madras
Boosting

(Refer Slide Time: 00:13)

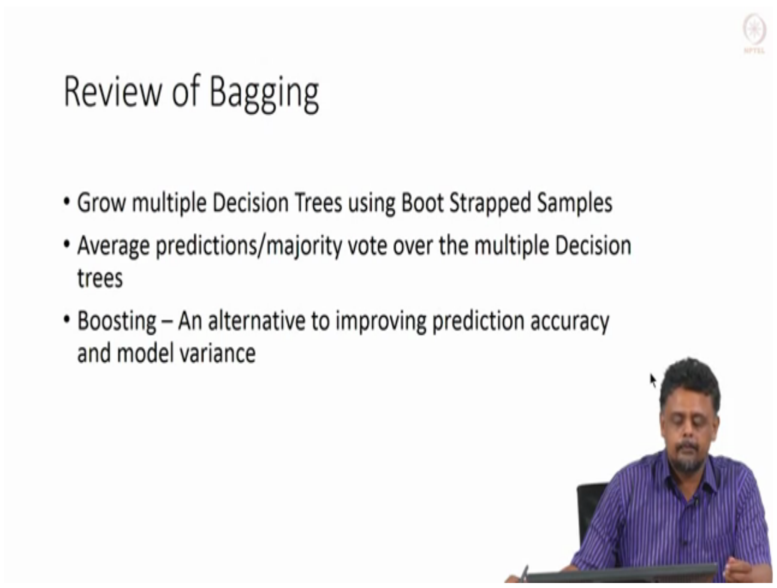


Boosting

Illustrations provided by Intel Software
Content inspired by 'Elements of Statistical Learning" Tibshiriani etal.

Hello and welcome back, in this video we will look at boosting which is another technique for improving the performance of decision trees most of the slides, graphics and illustrations were provided by Intel software and the content is inspired by the textbook elements of statistical learning, ok.

(Refer Slide Time: 00:32)



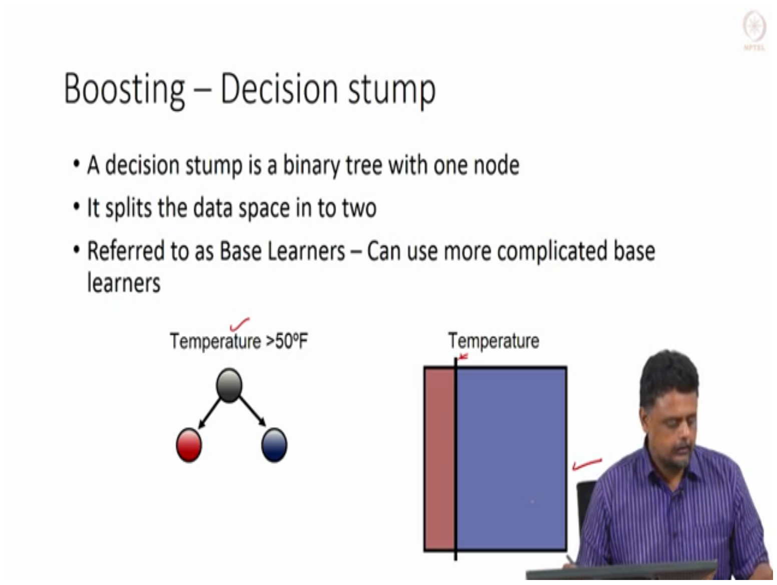
Review of Bagging

- Grow multiple Decision Trees using Boot Strapped Samples
- Average predictions/majority vote over the multiple Decision trees
- Boosting – An alternative to improving prediction accuracy and model variance

So we saw that in the last lecture bagging improves performance primarily by reducing the variance and that is accomplished by training multiple decision trees over bootstrap samples of your data ok and the output is basically the average of all the decision trees entire when you are looking at a classification problem or to take a majority vote if the output is the average from all the trees when you are looking at a regression problem or you take a majority vote if you are looking at classification problems.

So in this lecture we will look at boosting which is an alternative to bagging and also improves our prediction accuracy, ok.

(Refer Slide Time: 01:20)



Boosting – Decision stump

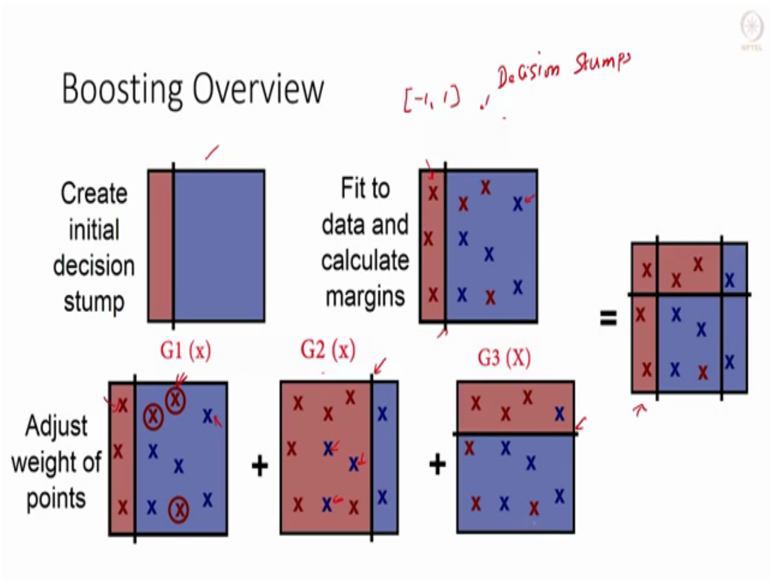
- A decision stump is a binary tree with one node
- It splits the data space in to two
- Referred to as Base Learners – Can use more complicated base learners

So we will take a brief overview of boosting, I just to get an understanding of the algorithm and one of the classifiers used in boosting is a decision stump it is basically a band is a tree with a classification tree with one node and it splits the data space into two these decision stump are referred to as based learners and if you want for booting if you want for boosting or bagging we can use these stumps or more complicated decision trees.

So as I mentioned we earlier the building block for boosting is a decision (trump) stump primarily and it has one node so if you look at particular data set which has temperature as one of its features, so we can split the feature space into two at this particular node based on a threshold on the temperature, so it is just illustration of how the decision stump space is split into two and this is your classification boundary.

So the all the data elements data points the right of the classification boundary or belong to one node and the left to the other, ok. So we will use a collection of these to perform boosting let us see how it is done, ok.

(Refer Slide Time: 02:50)



So before we so we how we create an initial decision stump based on one of the features to split the data space into two ok, so just for the sake of illustration here we just showing a 2d plot of course we know that an input data can have multiple features so we are looking at just one feature and maybe here this is a visualization of a couple of features ok, so well based on the splitting ad by the decision stump then we have two classes, so again we are looking at a binary classification problem where the output is either minus 1 or 1 ok.

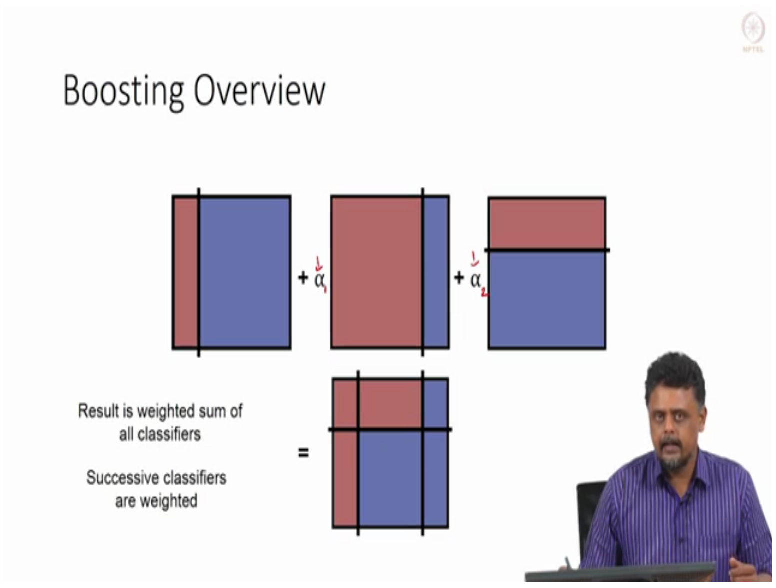
So we have the decision boundary and we have correct classification of the red ones red data points to the left and the correct classification of the blue data points to right these red crosses on the right or basically misclassifications right, so then what we do is we adjust the weights of those points so we assign some data waiting to those points when you calculate the loss function for decision stump we do a further classification after assigning weights.

So think of assigning weights as you know if you take a simple least squares cost function you will assign a higher cost to these data points compared to the others or if in fact you can even separates or make the contribution of these data points minimal compared to that is the data points that have been classified accurately to a minimum and you can assign higher weights to or higher weights in the sense higher loss for miss classification loss to the data points that have been misclassified in the by the previous decision stumps.

So then we have this riveted data points which you classify again to get a new classification mode in this case again all the red points have been classified correctly while the they well there are a few miss classifications for the blue data points, so one and then one more time we do the riveting and get a new classification boundary here, so the output of boosting is basically the sum of all these boundaries of all these classifiers, so each one of these we call this G 1 x, G 2 x, ok so each of these classifier G 1, G 2, G 3 provide you a classification boundary and based on the classification boundary you have classification as minus 1 or 1 the points that are misclassified are accorded higher weights for the successive classifier.

So most of the time these are decision stumps that is what we saw earlier versions stumps right, so the next G 2 is a decision stump which takes the same data as input but the data points are which have been misclassified by the previous decision stump for given a higher waiting finally the classifier that you use in the end is basically the summation of all the individual classifiers, ok.

(Refer Slide Time: 06:15)



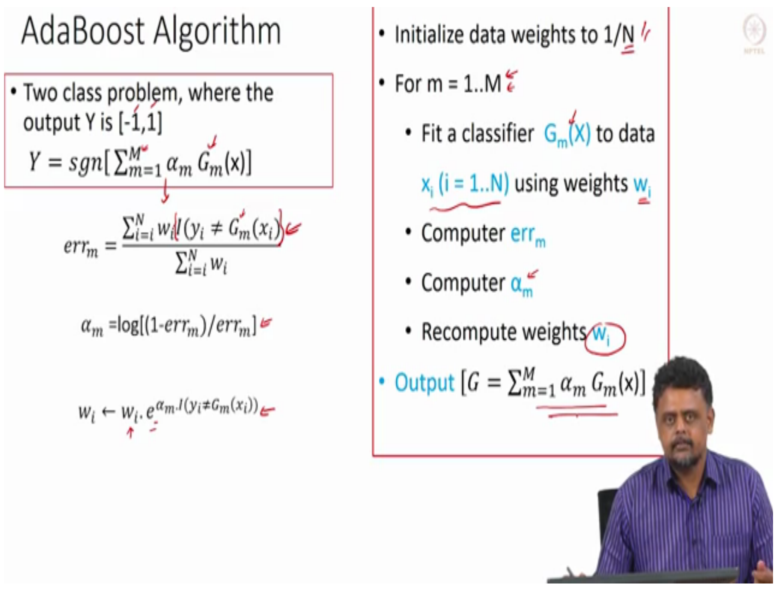
Boosting Overview

Result is weighted sum of all classifiers

Successive classifiers are weighted

So and in order to improve this since they are prone to over fit you typically have a weighting factor here alpha that is I could call it so this should be different for each one of them so I will call alpha 1, alpha 2 so on and so forth to get your final decision boundary, ok. So the result of boosting is a weighted sum of all classifiers and successive classifiers are weighted and successive classifiers are weighted according to some scheme which we will see in a later slides, ok.

(Refer Slide Time: 06:52)



AdaBoost Algorithm

- Two class problem, where the output Y is [-1,1]
  $Y = sgn[\sum_{m=1}^{M} \alpha_m G_m(x)]$

$$err_m = \frac{\sum_{i=i}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=i}^{N} w_i}$$

$$\alpha_m = \log[(1-err_m)/err_m]$$

$$w_i \leftarrow w_i . e^{\alpha_m . I(y_i \neq G_m(x_i))}$$

- Initialize data weights to 1/N
- For m = 1..M
  - Fit a classifier $G_m(X)$ to data x_i (i = 1..N) using weights w_i
  - Computer err_m
  - Computer α_m
  - Recompute weights w_i
- Output $[G = \sum_{m=1}^{M} \alpha_m G_m(x)]$

The Adaboost algorithm which is one of the boosting algorithm start up boosting algorithm it is a very popular boosting algorithm, we look at it in the context of a two class problem where the output of the classifier is minus 1 or 1 is basically the so it is a binary classifier and the resulting classifier is the weighted sum of individual classifiers we have Capital M classifiers each of them can be decision stump.

So we have a weighted sum of these classifiers as the output so the sign of the weighted sum of these classifiers is the output ok that is the problem we are trying to solve, so how does the Adaboost algorithm progress so initialized data weights to 1 over N so when we start out we assign the same waiting 1 over N where N is the total number of data points to each of the individual data points x i ok and for M classifiers capital M classifiers for each iteration you fit a classifier to the data x i using the weights w i.

So in the first iteration the weights are just 1 over N and they are all the same ok so and then we compute this term error this error term after the classification which is given by this expression so this is nothing but the weighted error term, so if you look at this particular expression here this is nothing but the number of samples that have been misclassified so G m of f x i is the output of the M classifier and y i is the ground truth and you are just counting i is the indication or indicator function, so we are just counting the number of samples x i that have been misclassified by this G m, ok.

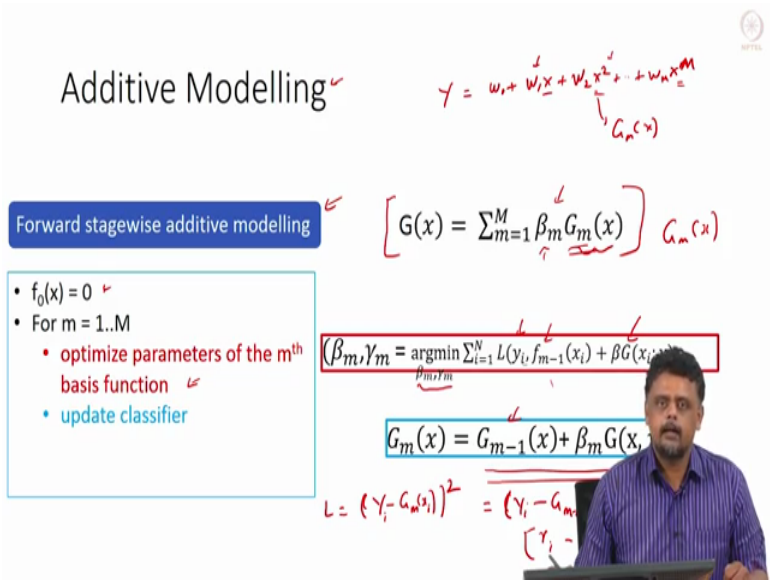
So the error that we calculate for this classifier in the M iteration is the weighted error based on the weights which for the first iteration is 1 over N ok so how do we update the weights that is what we will see, we compute also have to compute these alpha m which are the waiting is for the individual classifiers, so we can calculate alpha m using this expression ok and once we have calculated the Alpha m then we can go ahead and update the weights.

The weights for the next iteration are given by this expression where again it is a weights from the previous iteration times this exponential factor ok, so this is the loop which is performed M times in the end we output the classifier as the weighted sum of the individual classifiers, ok. So this is just the algorithmic summary for what we saw earlier in a very illustrative form, so in every iteration you fit your data to a what is called a weak learner or a weak classifier ok and you look at all the data points that have been misclassified and you when you assign higher weights to those data points so that is what this step does.

So you are assign higher weights to those data points and then you take those data points all the data points including the ones that have been misclassified but then with higher weighting and you if you and you perform one more and if and you fit that data again using another classification tree or as or other classifier and you repeat that as many times as the number of classifiers that you want to use, ok then it is easy to see that it that the since this M is none of the seems like a free parameter it can most easily over fit, so that is one of the reasons why we iterative calculate the waiting is so that the learning up happens very slowly, ok.

So over a large number of classifiers you will come to a we weighted linear sum would give you a good classification accuracy the point note is that this individual G i is or G m is can be weak learner in a sense they perform as well or maybe poorly than random classification, so if you accumulate all of them over M i capital M i iterations you can get to a very high performing classifier, all right.

(Refer Slide Time: 11:16)



Additive Modelling

$Y = w_0 + w_1 x + w_2 x^2 + \dots + w_n x^M$

$G_m(x)$

Forward stagewise additive modelling

$G(x) = \sum_{m=1}^{M} \beta_m G_m(x)$

$G_m(x)$

- $f_0(x) = 0$
- For m = 1..M
  - optimize parameters of the mth basis function
  - update classifier

$(\beta_m, \gamma_m = \underset{\beta_m,\gamma_m}{\text{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta G(x_i$

$G_m(x) = G_{m-1}(x) + \beta_m G(x,$

$L = (Y_i - G_m(x_i))^2 = (Y_i - G_m$

$[Y_i -$

So where does this idea come from, so it is basically has roots in this additive modelling so where we have this particular model where our classifier is nothing but a linear combination of multiple classifiers, so each of this G m can be a decision stump as we saw earlier or they can also be regular classification trees, ok. So now the problem is in we have a cost function when see that the individual classifiers will have their own parameters and we also have to estimate this beta m is ok, so how do we go about doing that? So we are then that is accomplished using this forward stage wise additive modelling, so let us have a look at that.

So we start off with a function initialize to 0 then for M steps so as many classifiers as we have optimize the parameters of the M basis function or classifier so the way it is called basis function is that so if you might all be familiar with the linear regression class that you have been through, so we can let us say if you are doing regression let us say Y we can model as W naught plus 1 x in this case plus x to the M write capital M.

So this is one of the models that we have you that is that we have seen in linear regression so we can consider each of these this x, x square x to the M these polynomials as the basis functions, so we can write it in this screen so here these correspond to the G m of x ok, so we can think of each of these classifiers as some basis function and we are trying to iteratively estimate the parameters corresponding to each of them and also the weighting that we have to assign.

So there is a weight or in this case these x are very this is a very simple polynomial so we can assign for instance instead of x we can use some basis spline functions ok they will have multiple parameters that we will have to determine, ok. Now when we put this in a cost function it becomes very difficult to determine, so what we go about doing is in every iteration we only estimate that particular G m, so an mth iteration we will only estimate G m of x ok without chaining the parameters of the first M minus 1 classifies that we have built ok.

So for every iteration optimized parameters of the mth basis function ok, so which is basically given by this formula here, so argmin with respect to the beta and gamma M L is the loss function for one data point so it is summed over n data points so L is a function of, of course your well takes as input your ground truth y i your classifier from the previous iteration and this is the classifier that we want to estimate ok.

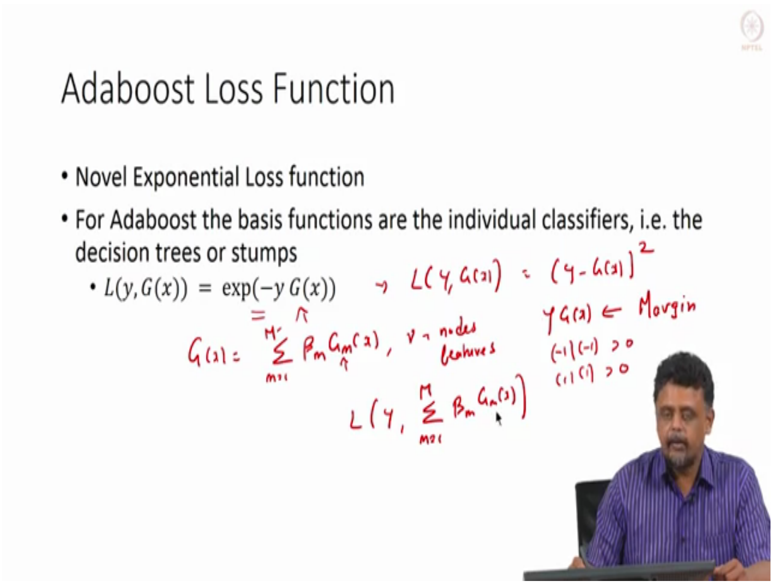
So the beta and G or what we have to figure out in this iteration when we make no changes to the parameters involved in the classifiers and the beta from the previous iterations, so once you have done that then we update the we update the classifiers as by adding with an appropriate weight is the basis for the Adaboost algorithm that we have seen, so in this case the loss function we have not specified exactly what it is so we can actually look at a very simple loss function let us say we are looking at Y minus I will just call it G m of x square which is your least squares cost function so based on this this L is just that which we can write it as I am just going to use x i I am going to do it for one data point right because the sum of all of them so which is y i minus G m minus 1 of x i minus beta G of x i square.

So we have developed a classifier till the M minus 1 iteration right and we have just substituted this expression here and we have instead of G m of x y we have just used this expression G of m minus 1 x plus beta M G of x, so we this is really the cost function that we are optimizing if you use a least squares cost function now you can easily see that this is nothing but the residual from the previous iteration, so there is it so here we can just call it r i minus beta G of x i square, ok.

So if you use a large least squares cost function what you are what you will end up doing is you will fit a particular regression algorithm to machine learning algorithm to it I would say regression tree, so we will get you will fit a regression tree to your data and then you will calculate the residual so for every x i you will get an output and you subtract that from the ground truth so we will have residuals and in the next iteration you will take those residuals and feed them with your input data, ok.

So that is the that is how it progresses if you are if you use a least squares cost function now instead of a least square cost function if you use an exponential cost function what you end up with skating Adaboost that is what we will see now.

(Refer Slide Time: 17:29)



Adaboost Loss Function

- Novel Exponential Loss function
- For Adaboost the basis functions are the individual classifiers, i.e. the decision trees or stumps
  - $L(y, G(x)) = \exp(-y\,G(x))$ $\rightarrow L(y, G(x)) = (y - G(x))^2$

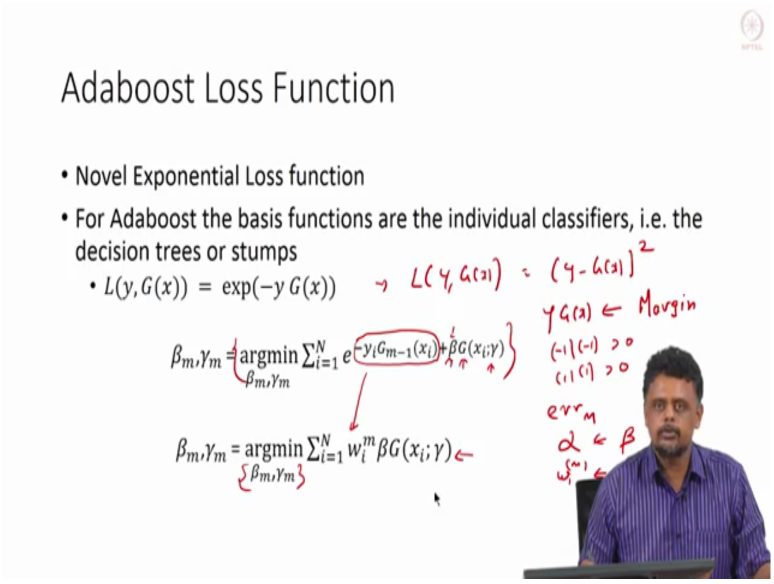
$y G(x) \leftarrow$ Margin

$(-1)(-1) > 0$

$(1)(1) > 0$

$G(x) = \sum_{m=1}^{M} \beta_m G_m(x)$, $\gamma \rightarrow$ nodes features

$L(y, \sum_{m=1}^{M} \beta_m G_m(x))$



Adaboost Loss Function

- Novel Exponential Loss function
- For Adaboost the basis functions are the individual classifiers, i.e. the decision trees or stumps
  - $L(y, G(x)) = \exp(-y\,G(x))$ $\rightarrow L(y, G(x)) = (y - G(x))^2$

$y G(x) \leftarrow$ Margin

$(-1)(-1) > 0$

$(1)(1) > 0$

$$\beta_m, \gamma_m = \left\{ \underset{\beta_m, \gamma_m}{\operatorname{argmin}} \sum_{i=1}^{N} e^{\boxed{-y_i G_{m-1}(x_i)} + \beta G(x_i; \gamma)} \right\}$$

$$\beta_m, \gamma_m = \underset{\{\beta_m, \gamma_m\}}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^m \beta G(x_i; \gamma) \leftarrow$$

err_m

$\alpha \leftarrow \beta$

$w_i^{(m)} \leftarrow$

The Adaboost loss function or the algorithm that we saw for Adaboost comes from using an exponential loss functions and the basis functions or the weak learners or the individual classifiers this can be decision trees or stumps. So previous remember previously we saw this L of y comma G of x I am omitting the subscripts i we saw that as Y minus G of x square this is what we use to be as an example in the previous slide, so instead of that we have an exponential y minus y times G of x, ok.

So this is this product for a classification problem is often referred to as a margin right, so you can easily see that let us say your output is a sub so your category is minus 1 and let us say your classifiers also inputs minus 1 then it is greater than 0 right let us say your category is 1 and you are in your classifier also outputs 1 then it is again greater than 0, so for all positive values of this product you have correct classification and all negative values you have a miss classification, so that is the margin or you can think of it as the distance from the decision boundary is what we look at.

So when you introduce this exponential function as the loss function in order to solve your additive model again to recall we saw that for these Adaboost comes under this additive modelling, so we wanted to get to a classifier which is the sum of individual classifiers ok right, this is what we sort out and we wanted to optimize an appropriate cost function to estimate each one of them.

Now if you want to do it the right way then you would (actua) eventually we have a loss function that will try to estimate all of the parameters of G m and beta m in one shot by optimizing a cost function but that becomes very complicated let us say Capital M is let us say 100 weak learners then there are 100 sets of parameters for each one of the G m is here ok or each in this case decision tree, so you might wonder what would be what are the parameters for addition tree the parameters are denoted by gamma and they are basically the nodes and the features that was split on ok, so that you have to estimate for every tree that you fit.

So to elevate this problem we have the additive stage wise modelling wherein we initially estimate we have we do M i iteration capital M i iteration where we start off with one decision tree typically in this model suggestion tree which has the one decision tree or a relation stump and you estimate that the parameters of the decision tree in this case how you grow it and the nodes and the features and successfully update by, so if you have let us say m i small m i iterations you do not change the parameters of the first m minus one iteration but only consider the parameters of the mth titration right only update the parameters corresponding to the m transition.

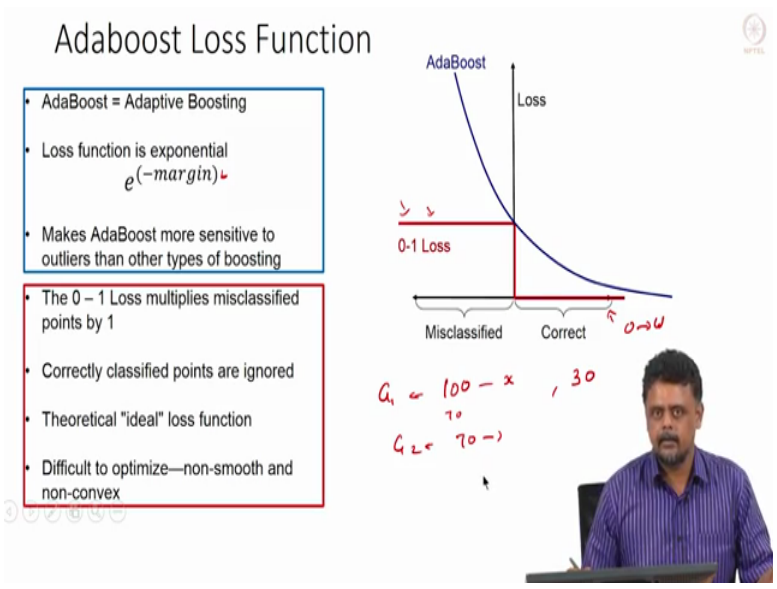
So that is this stage wise modelling helps to solve this problem so this is the easier way to so if you will have a loss function which takes as input the y ideally it will take as input the y and your model right so then you will estimate all of the parameters for all of the individual decision trees for the weak learners in one shot but this is a difficult problem to solve so that is why we saw that we take as input because you are y this is the ground truth and we have a this is an additive model we coded G of m minus 1 x and plus some beta times G of x and we do not touch any other parameters here but only estimate the parameters corresponding to this ok, so that is the idea behind using this additive stage wise modelling, ok.

So for if you use an exponential loss function so this loss function if it is an exponential loss function exponential of minus y G x n we end up with the Adaboost algorithm we will just take a brief look at the loss function itself I am going to erase this so that you can clearly see the formulas involved, so the problem that we end up solving if we use a exponential model is this argmin of this loss function because all of this to all of the terms are in the exponent right and you realize that since we saw earlier that the these the parameter (corre) here or not at all affected by this optimization problem because we will only be optimizing the parameters of G in this case this gamma and beta at that particular iteration.

So we can write this as replace this by a weight ok, so this is the loss function that we end up optimizing for the Adaboost algorithm, so again we saw some of the update steps there when we when I outline the algorithm we can calculate the error calculate the error rate m from there we derived an alpha right which is the of course which is this which is related to actually beta that we see here in addition we also did a weight update so the weights every iteration are updated right times some factor, ok so all of these can be derived by from first principle using these two expressions all I will do is optimize for gamma so in this case we are we are optimizing for beta m and gamma m which is gamma m or the parameters of your decision tree.

So you figure out the best decision tree by optimizing the misclassification rate or minimizing the best classification rate and then once you hold that once you fix that then we can optimize for beta which is which can be done by taking a derivative of this loss function with respect to beta setting it to zero and then plugging in the, the optimal decision tree these weights the update rule for these weights the estimate for beta extra can be obtained by just optimizing this loss function even analytical expressions can be obtained, so this is the basis for the Adaboost algorithm.

(Refer Slide Time: 25:11)



Adaboost Loss Function

- AdaBoost = Adaptive Boosting
- Loss function is exponential $e^{(-margin)}$
- Makes AdaBoost more sensitive to outliers than other types of boosting

- The 0 – 1 Loss multiplies misclassified points by 1
- Correctly classified points are ignored
- Theoretical "ideal" loss function
- Difficult to optimize—non-smooth and non-convex

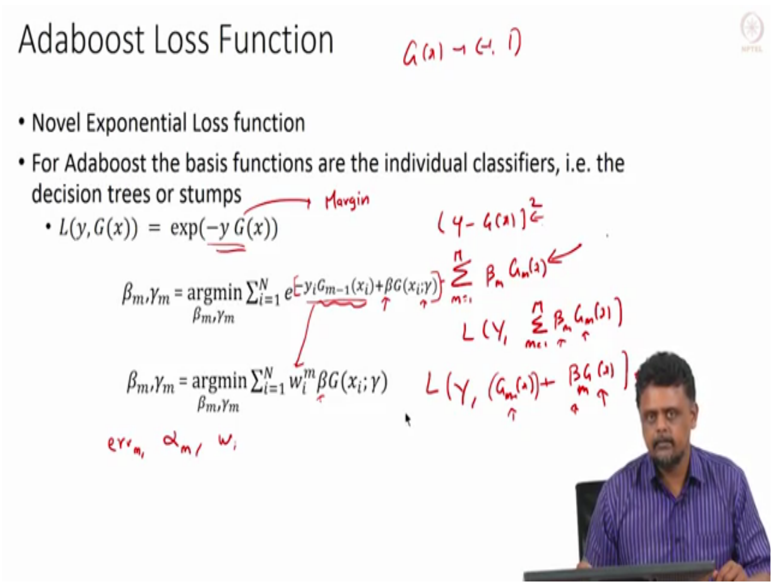
So we look at just to have a look at understanding what this algorithm actually does, so we look at this red curve here so it is a 0 1 loss so let us say we have we have a binary classifier which classifies your output as 1 or minus 1 so what we ideally want to do is to assign 0 weight 0 to w to all the correctly assigned data points and a maximum weight of 1 to all the misclassified data points and then move on to fitted fit it with another regression tree or a classification tree depending on the problem ok but this kind of loss function is difficult to optimize, so that sense it is replaced by this exponential loss function that is what Adaboost does, ok.

So the exponential loss function as we saw is the e rest to minus margin where margin is where this, this makes the Adaboost more sensitive to outliers then other types of boosting ok the theoretical loss function even though is very you know in to do we can see that here so really what we are trying to do is we take it as I said let us say 100 data points in ideal cases what we do 100 100x ok data points let us say some 30 of them are classified correctly and some 70 of them are misclassified ok then in the next step so this is G 1 and G 2 what you will do is you will take 70 and then fit another classifier ok but this is a this problem even though it looks very appealing or does not give good results because it is a difficult problem to optimize.

On the other hand having a exponentially decreasing loss function instead of this step loss function helps improve your fitting ok, so in the next class we will look at a further modification to this called gradient boosting so the procedure we outlined here works very well for the exponential loss function however gradient boosting techniques are good for pretty much every kind of loss function that you can come up with.

So it is a very generalized procedure for doing boosting and it is also one of the apparently too many sources one of the more popular techniques to win these kaggle competitions, so we will look at those in the next video.

(Refer Slide Time: 27:51)



Adaboost Loss Function

$G(x) \to (-1, 1)$

- Novel Exponential Loss function
- For Adaboost the basis functions are the individual classifiers, i.e. the decision trees or stumps
  - $L(y, G(x)) = \exp(\underline{-y\, G(x)})$ Margin

$(y - G(x))^2$

$$\beta_m, \gamma_m = \underset{\beta_m, \gamma_m}{\operatorname{argmin}} \sum_{i=1}^{N} e^{[-y_i G_{m-1}(x_i) + \beta G(x_i;\gamma)]}$$

$\sum_{m=1}^{n} \beta_m G_m(x)$

$L(Y, \sum_{m=1}^{n} \beta_m G_m(x))$

$$\beta_m, \gamma_m = \underset{\beta_m, \gamma_m}{\operatorname{argmin}} \sum_{i=1}^{N} w_i^m \beta G(x_i;\gamma)$$

$L(Y, (G_m(x)) + \beta_m G(x))$

err_m, α_m, w_i

So the Adaboost loss function is an exponential loss function and the basis functions as we had mentioned earlier nothing the other individual classifiers or the decision trees or the stumps ok. So recall that the previous slide we looked at a least squares function right wherein we did y minus G of x square right and this additive modelling proceeds or the stage wise additive modelling precedes by trying to optimize trying to estimate not of trying to estimate this see we want to get this classifier which is nothing but and we can use either alpha or beta but just use.

So if you have a loss function it will take as input your ground truths and your model just what we did, right and ideally we would end up estimating all the parameters of each one of these individual classifiers as well as the weighting factors for each of those classifiers in one (())(29:07) but when you are using let us say decision trees this becomes a slightly difficult problem to solve, so we solve this problem in a stage wise manner so that is why we saw earlier that we ended up with you are initially we start off with this model that is your G x plus ok.

So this is the mode of x here what it means is that we will not touch the parameters up to in this model or not affected at all in the when you are optimizing this loss function you are only optimizing for beta m and this G at this particular step in particular stage m ok and so we saw how this works for the least squares cost function and instead of the least squares cost function if you use an exponential loss so what does it look like right.

So this is the so I have so L is given by so this function where this particular term y times G of x is the margin ok so you can think of it as the distance of the data point from the classifier further away it is the worse the classification you can think of it in that way or if for a binary classification task you see that y times G of x is always positive right because G of x is the output of the classifier at the end your class and your classes are minus 1 and 1 so Y times G of x will always be positive when your classifier is right and negative when it is not, ok.

So we have instead of the least squares we saw it like this we have put in the exponential loss function ok and in the mth stage we will not modify the parameters of these class G m minus 1 but rather only focus on estimating beta and gamma here, so for the in the case of regression trees or classification this gamma represent there is a nodes and the features used in the split ok. So in every stage you will estimate 1 you will figure out 1 decision tree or stump and you will add the add that to the classifier that you have already estimated so far till the m minus 1th stage right.

So since this is not involved anywhere in the optimization we can (estim) interpret this as a weight ok if you recall the I can go back and look at the Adaboost algorithm that the outline then we had this quantities error and from where we calculated alpha m as well as the update to the weights these alpha and the beta are related ok so alpha turns out to be I think beta or 2, so for how do we so the estimates can be derived analytically by fixing beta and estimating G which is nothing but the best classification tree you can get, so you get the best classification tree by minimizing the misclassification rate once you have figured that out then then we can then fix that and then take the derivative of this expression with respect to beta and we can do further analysis, ok.

So if you do that then we get all the update tools that we saw earlier so that is the basis of the Adaboost algorithm it is way it starts from this additive model right here ok, so it states like a linear basis function model ok so G m of x or basis functions in this case we have used decision trees we can use other structures also if possible and we and every step we successively improve the classification or the prediction accuracy.

Of course so far we have only looked at the binary classification problems it is also applicable to multiple classes and another thing is that we have only looked at this discrete Adaboost wherein we assume that our classifier returns minus 1 or 1 we can modify this you know in the case when our classifier returns a probability values map with a 0 and 1 corresponding to classes minus 1 and 1 ok, this can be also be done for the sake of illustration we only looked at the binary classifiers.