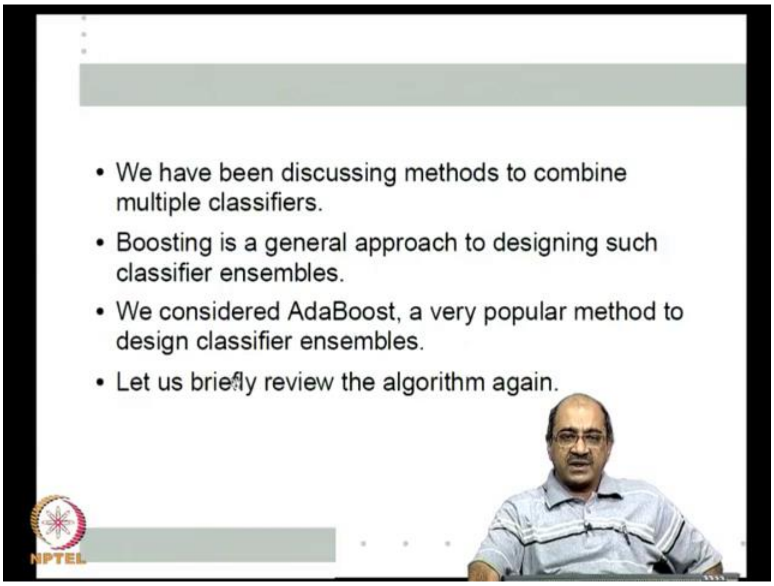
Pattern Recognition
Prof. P. S. Sastry
Department of Electronics and Communication Engineering
Indian Institute of Science, Bangalore

Lecture - 42
Risk minimization view of AdaBoost

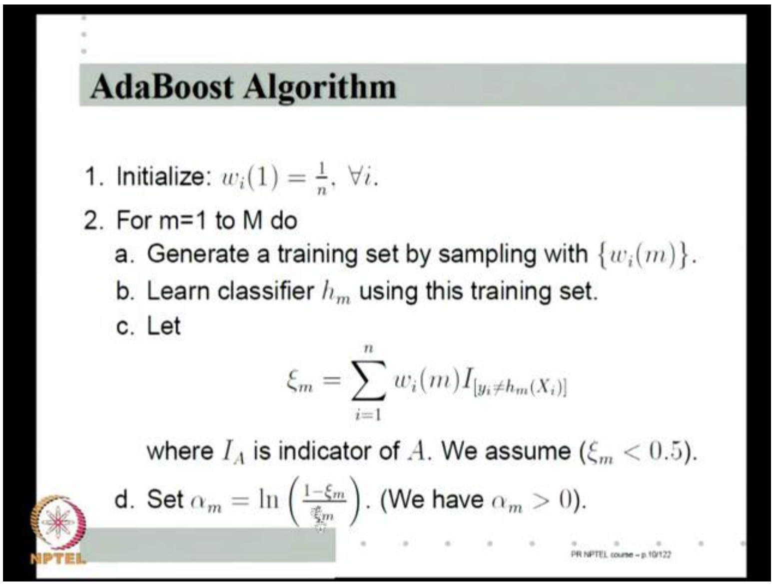
Hello and welcome to this final lecture in the course of pattern recognition, we have been last class we been looking at classifier ensembles, that is how can we combined many different classifiers, we learn for the same pattern recognition problem. So, that we can improve the accuracy, we have briefly looked at an algorithmical called AdaBoost today one of the best classifier ensembles algorithms. Today's class will complete our discussion on AdaBoost, we look at AdaBoost also from a risk minimization point of view.

(Refer Slide Time: 00:53)



So, to continue we said boosting is a general approach for designing classifier ensembles. So, we looked at no a simple intuitively designed boosting algorithm of learning three classifiers just to recognize about boosting is about. And then in general, there many techniques of boosting, and AdaBoost is a very popular method of designing such ensembles, because it allows to design an ensemble of almost any size, and in practice it works quite well. So, let us go back to the algorithm we have discussed it last class, but we will look at it again.

(Refer Slide Time: 01:36)



So, the idea of AdaBoost is that I assign weights to each of the examples given, I got an example x i y i. So, each example I assign a weight changes from iteration to iteration at any given time, the weights are always kept normalize. So, at each iteration I use the weights as a probability distribution, and sample from the examples. That is example x i y i, I keep and taking n random samples from this example set so at each time is sample the independently chosen. And a fact that a x i y i the probability of an x i y i coming into my training set is given by w i, the weight at that time w i f m if m with y iteration.

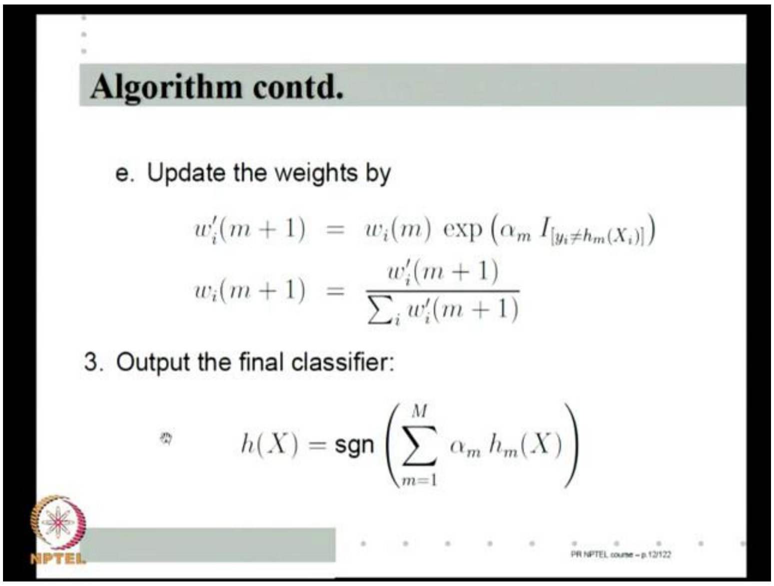
So, essentially the examples with higher weights are more likely to come and of course, this is because I am using w i as a probability distribution and sampling i i d, it is like sampling from the example set with replacement. So, multiple times I draw an example I might get the same example. So, this particular true if a particular example is very high weight it may be represented more than once in the training set, but without counting such I mean not bothering about such repetition, I have size n for my training set each training set is of size n and of course, some of the examples may be repeated because they are sample with w i.

So, the algorithm starts by initializing the weight to 1 by n. That means, initially each example is as likely to be chosen as any other example because we have no preference on examples. Then I keep learning classifiers capital M is a total number of classifiers ensembles sot that is my iteration count. So, at the m'th iteration I am learning the m'th classifier. So, what do I do? I generate a training set by sampling with this weights, then you learn a classifier h m using this training set, we have not said anything about how you learn this classifier. It can be any method, it can be a neural network, it can be s v m, it can be linear classifier, it can even be dish entry.

We have not consider dish entries except in the first introductory class I talked about we won’t say what the classifier is also. As I said on different iteration you may use different classifiers, all that is of that that does not effect the algorithm. So, we whatever is our preferred way of learning the base classifier each of the h m's are called base classifiers. We learn it and then we calculate the error x i m is the weighted error see indicator of y i not is equal to h m x i is simply means, on this I example, I met an error right and the error is weight by the weight of that example at this time.

Mind you this is not naturally, the training set because the training set is randomly chosen, but this is an all the examples I have x i y i. For each example is classified by the newly learn classifier and wherever, I misclassified my actual error is weight by the weight of that classifier. So, I want to minimize this weight or some in a sense I can say. I want to learn a classifier h m which minimizes this weighted error. We assume that each of the classifiers we learn is said that the error less than 0.5, in practice all it means is if I had a classifier whose error is greater than 0.5. I throw it away and learn again than set alpha m is l, n 1 minus x i m by x i m because we assumed x i m less than 0.5, 1 minus x i m is greater than x i m and hence, alpha m is always positive.

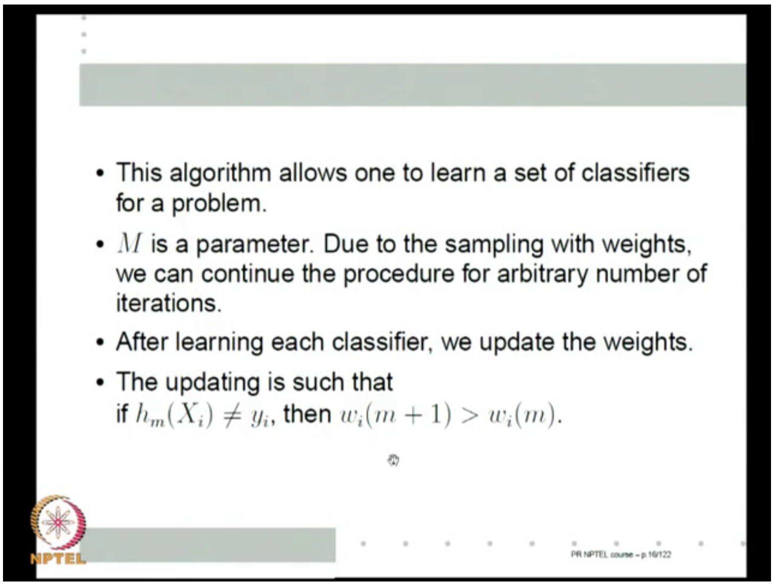
(Refer Slide Time: 05:33)



Than for the next iteration I learn a new weights. So, actually I want to keep weights normalize. So, the unnormalize once whatever the old weight I increase the weight, if I miss classifier this example, at this stage if y i is not equal to h m x i. Then I multiply w m by exponential alpha m because alpha m is positive, this is greater than 1 exponential alpha m is greater than 1. So, I increased the weight and then I finally, normalize all the weights.

And my final learn classifier after learning all the h m's my final classifier is simply summation is equal to 1 to m alpha m h m x. Where this alpha m's are what we calculated, which is l n 1 minus x i m, x i m where x i m is the weighted error of the n'th classifier. So, alpha m is some kind of a weight for the accuracy of h m. So, I take this sum and its sign is what I am using as h x.

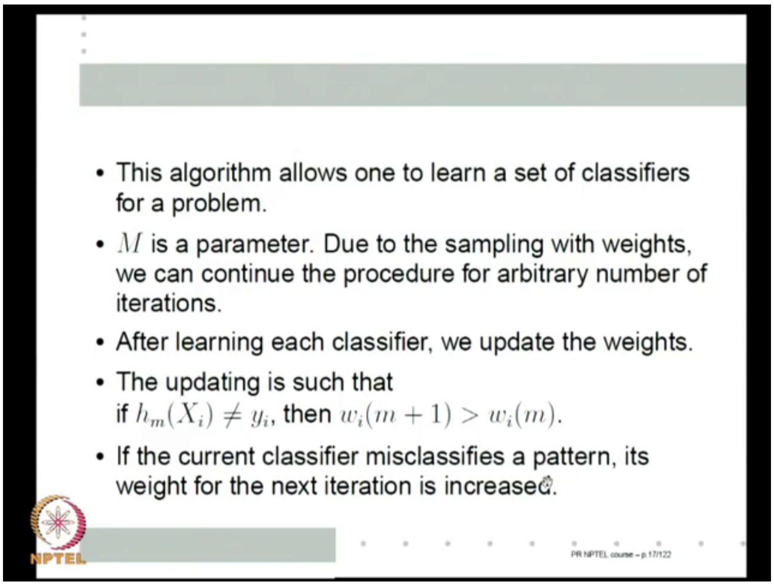
(Refer Slide Time: 06:28)



So, this is the algorithm that we cancelled last time, the algorithm allows one to learn a set of classifiers for a problem. M is a parameter as we said we can learn arbitrary number of classifiers. So, we can learn ensemble with any number of classifiers in principal. After learning each classifier we update the weights, the whole idea of the weights is weight update is such as a just mentioned is that, if h m x i not equal to y i then w i m plus 1 is greater than w i m.

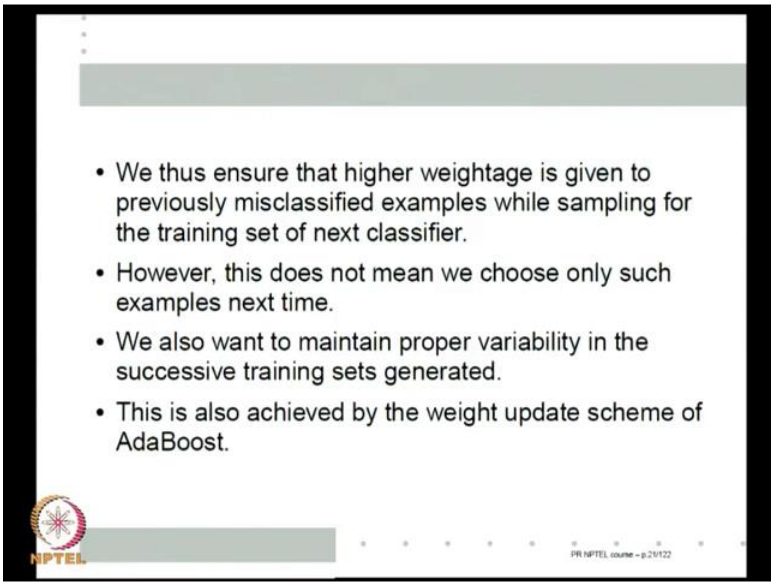
If y i not equal to h m x i then this will be exponential alpha m this factor and alpha m is greater than 0 alpha m is positive. So, this is greater than 1. So, w i prime m plus 1 is greater than w i m and hence, of normalization also l term. On the other hand if y i is equal to h m x i than this factor is 1. So, w i prime m is w i m. So, essentially in w i prime I am increasing weight of all examples, whose which is miss classified leaving others where they are and then the renormalizing.

(Refer Slide Time: 07:38)



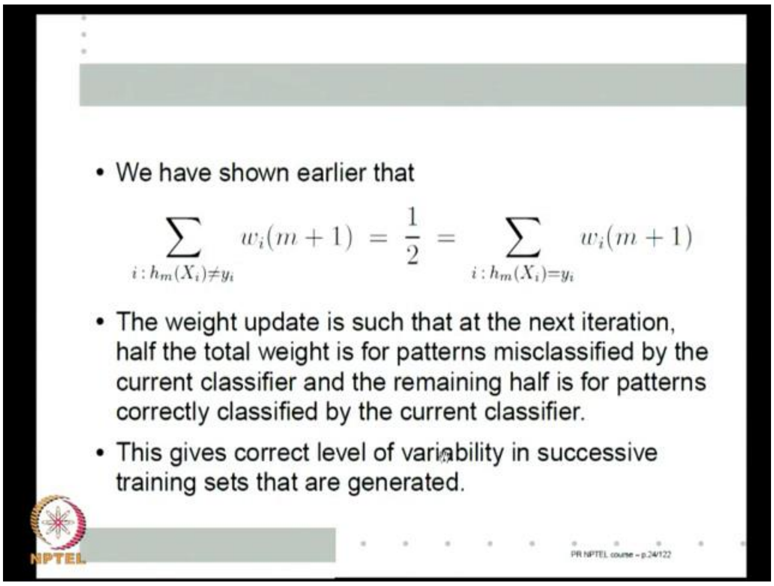
Which means, if I have misclassified error example, then its weight for the next iteration increases.

(Refer Slide Time: 07:52)



So, we ensure higher weightage is given to previously misclassified example. Essentially, that is the whole idea of generating successive training sets. However, we do not want only misclassified examples, we already looked at it when we construct general boosting that does not give me enough variation the training set right. So, is not that we can just arbitrary increased the weightage have to know, sample only if in the misclassified examples for the next iteration. We also want to maintain proper variability in the, in the successive training sets such that a generator. As you seen you know we want a good balance between misclassified examples, and properly classified examples. So, this is also achieved by weight update scheme of AdaBoost, this is what we saw last.

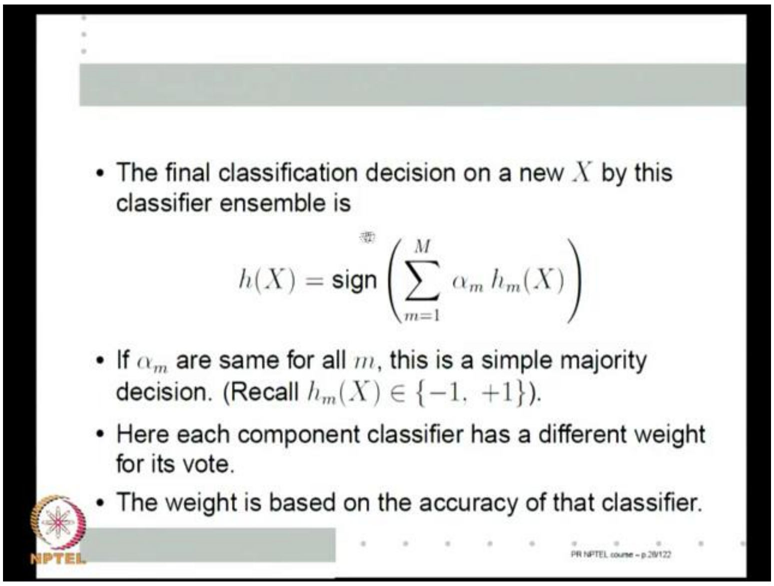
(Refer Slide Time: 08:48)



A matter of fact last class we proved that the weight of date is such that, if I sum all the weights over all i such that, x i y i is misclassified by h m that sum is equal to some of all weights over i such that h m x i is equal to y i. So, because total weights is 1 and this two sums are same each is half. What is that means? The weight update is such that at the next iteration half the total weight is for pattern misclassified by the current classifier, and half the total weight is for patterns correctly classified by the current classifier.

This is automatically achieved by my weight update equation, my weight update equations form is such that this is achieve. We proved it last class that this happens to be so. Now, this is a very, very interesting thing about algorithm the algorithm. So, this way the algorithm automatically, can generate as many training sets as you want of required variability. We do not have to worry about it, the weight update equations are such that they maintain this. This give us sufficient you know level of variability in successive training sets.

(Refer Slide Time: 10:03)



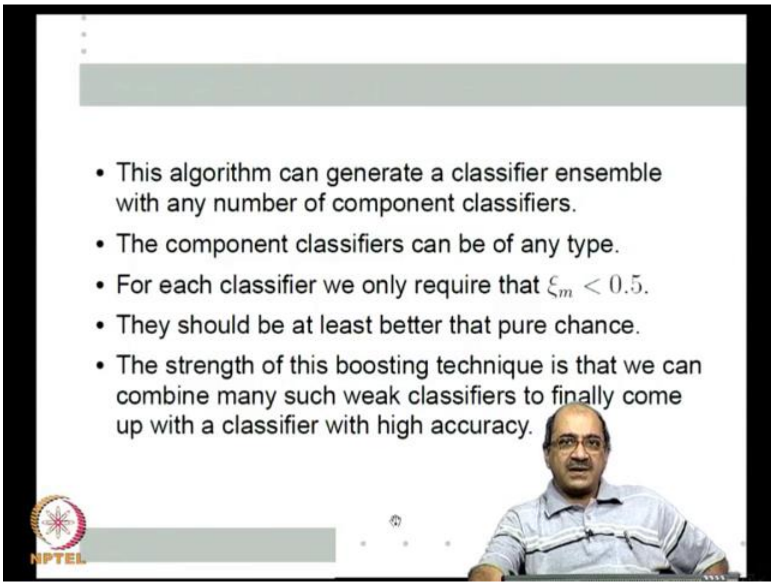
A word about the final classifier, what is the final classifier of this sign ensemble after learning all h m's, alpha m is as we seen l n 1 minus x i m by x i m. Where x i m is the weighted error. So, lower x i m higher is alpha m. So, I take alpha m h m x and take summation take it sign as my final classifier. Please remember all this algorithm everything, we considering only for the two class case, where the class labels are plus 1 and minus 1.

So, when I take the sign function if the sum is positive than the sign is plus 1 otherwise, it is minus 1. Essentially, can easily see that because each h m itself is a classifier, we assumed that h m x is either plus 1 or minus 1 for every x. So, because each of the plus 1 if all alpha m's are constant alpha m is constant, and positive then it can come out of the n function. Then if the sum has more plus ones finally, sign will be plus on the sum as more minus ones finally, the sign will be minus 1. So, essentially it would be a simple majority voting if I did not have alpha m.

So, if I had alpha m what it means is that it is a weighted majority vote. If I did not have alpha m each classifiers vote has the same weight as. So, I am asking how many classifieds a plus one and how many classifieds a minus 1 and I am take the majority. Now, I am not doing that because each vote has a weight, I am summing up the weights of all the plus 1 classifieds summing of all the weights in minus 1 classifieds, I am asking which is better.

The weight is alpha m weight depends on how good that classifier has been performing, the weight is based on the accuracy of that classifier. So, this a very interesting way to combine and once again a very, very general purpose way to combine classifiers, what is called a weight majority vote, this also very interesting feature of this AdaBoost algorithm.

(Refer Slide Time: 12:02)



The interesting thing about algorithm is it can generate a classifier, ensemble with any number of component classifiers. There are many applications face recognition, image classification, you know many, many areas people have used this and there are in many applications people use 100, 200 classifier ensembles. So, algorithm can generate classifier ensemble with any number of component classifier, as you can see our weight update is such that successively, I can keep generating size n training sets which have the right properties by updating weights, and sampling with those weights. And hence, the algorithm is you know very general purpose algorithm with lot of power.

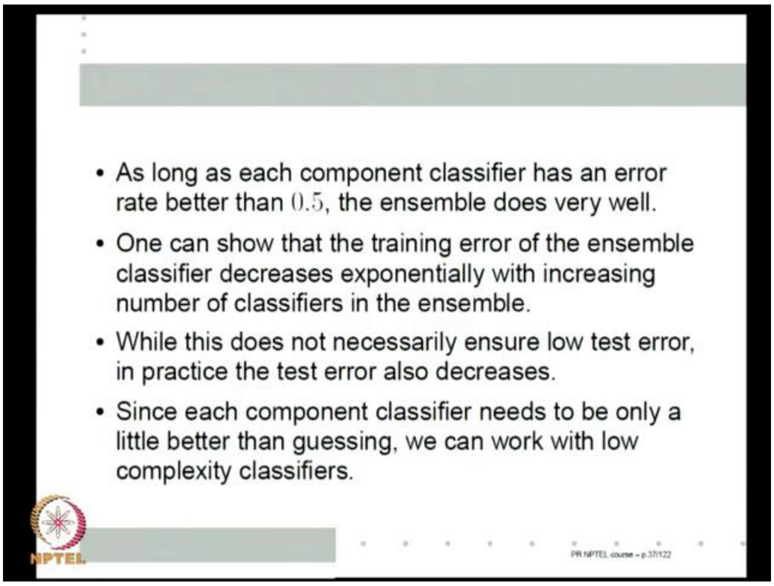
Second thing nice about the algorithm is I have to I do not have to worry about is if not as if applicable only for one type of component classifiers, you can put dish entries you can put neural network. So, you can put things learnt linearly squares you can learn what have you there is no restriction on the type of the component classifiers, that you can use all that we are asking for each classifier is that x i m less than 0.5. I said in practice, I keep doing it and if any time my algorithm returns a classifier with x i m greater than 0.5 it a way and learning again, but in general what we will asking is that is each of a classifier should at least tell xi m less than 0.5.

So, in this weighted sense they should be better than pure chance, the idea is that if the weight is high that particular example will be multiply represented in your training set. So, even if I am counting training set error if I misclassify that one example it will actually, be counted as 3 sent, or 4 training set examples are misclassified depending on the weight. So, in that sense essentially because the training set itself comes with the weight.

The weighted error being less than 0.5 is roughly same as saying that I can get I will get right classification more than 50 percent with the time. So, basically what I am asking is that I can’t have very bad classifier, classifier should be at least good enough to be better than pure chance cursing. Just that much I am know these are very weak classifier, I am not asking much of I am not saying that accuracy should be very high, all I am saying is that accuracy should be just little better than pure chance.

So, that is the strength of this kind of boosting techniques, right? Such classifiers are called weak classifiers because they can only achieve a little better than pure chance classifier, but there is good enough if you give me a many, many such weak classifiers while nobody alone can get very good accuracy. If you give me 100 or 200 of them combining them properly, I can get very high accuracy. So, this strength figure out an AdaBoost is that it can combine many weak classifiers into coming up with a very good high accuracy classifier.

(Refer Slide Time: 15:07)



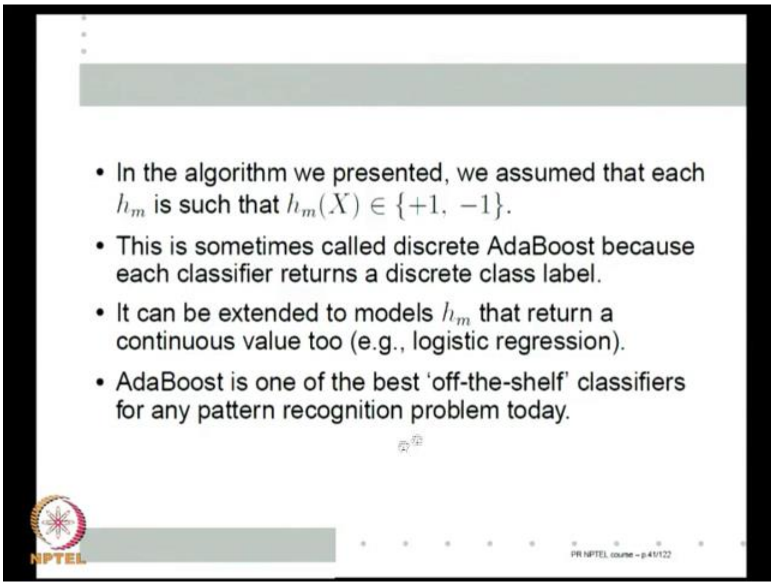
As long as each component classifier has an error rate better than 0.5, the ensemble in practice does very well. A matter of fact just a little algebra, we can show even though in the class we are not showing it, but is not very difficult to show that the training error the ensemble classifier decreases exponentially, with increasing number of classifiers in the ensemble. As long as we maintain this weighted error less than 0.5 in for each component classifier, one can show that the training error of the ensemble decreases exponentially fast with increasing number of classifiers in the ensemble that is a very, very nice property to have.

Of course, we know that low training error does not necessarily, mean low test error we can be doing over fitting. The reason why over fitting may not be occurring AdaBoost is to for firstly, we know it does not in the sense in practice in on many, many interesting applications many, many benchmark problems it does very well. One possibility is that because each component classifier, we ask for need to be only just a little better than pure chance guessing, can work with low complexity classifiers we do not have to take very high, high complexity classifiers.

Because of that right essentially, low training error could mean low test error and because ensemble gives me sufficiently, low training error it also gives me low test error.

Roughly, one way of looking at it is that ensemble allows me to get the higher accuracy, I do not have to go to very, very complex classifier structures. I can use simple classifiers let us say just linear classifier, but I use many of them that is how that is the strength of boosting.

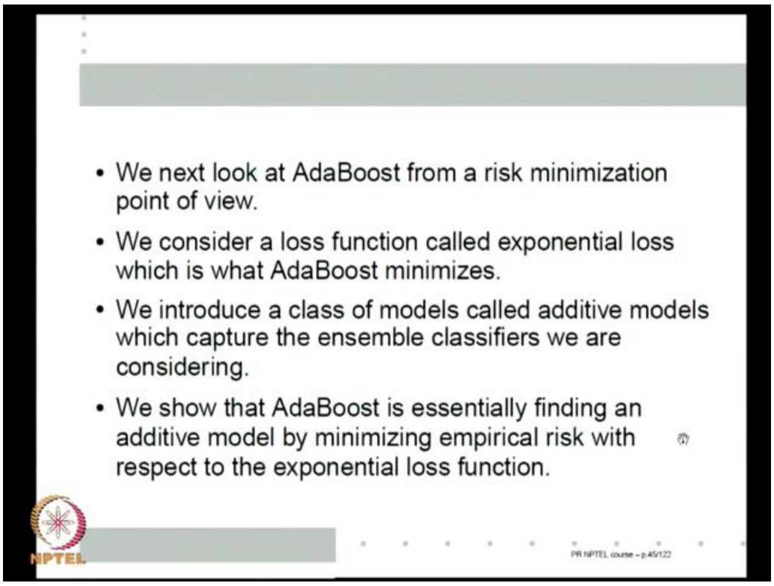
(Refer Slide Time: 16:56)



Of course, the particular AdaBoost algorithm we presented is such that we assume that each h m is at classifier, that is just a binary value output h m of x takes only values plus 1 and minus 1. So, this AdaBoost is called a discrete AdaBoost because each h m takes a takes on only discrete values the class labels. Of course, the AdaBoost algorithm itself it can be extended to other cases, where h m returns a continuous value. Say for regression problems also for classification problems, where h m could be a neural network. So, h m x is a continuous value or say logistic regression.

So, I can any such classifiers also the algorithm is slightly different, but still similar conclusion hold. So, one can also extend the algorithm that we presented to the case of h m that return a continuous value, rather than it discrete class labels as value. So, in this sense AdaBoost is possible one of the best of the shelf classifiers that are available today. Just like kernel base classifiers are good, and as I said the they should be the first choice whenever, looking at a classification problem. Similarly, a boosting AdaBoost is a very good off the shelf immediately, implementable strategy for classifier learning a classifier, on many problems. And that is that is why it is quite popular and that is the reason why, this is the only classifier ensemble that we considering in this course.

(Refer Slide Time: 18:38)



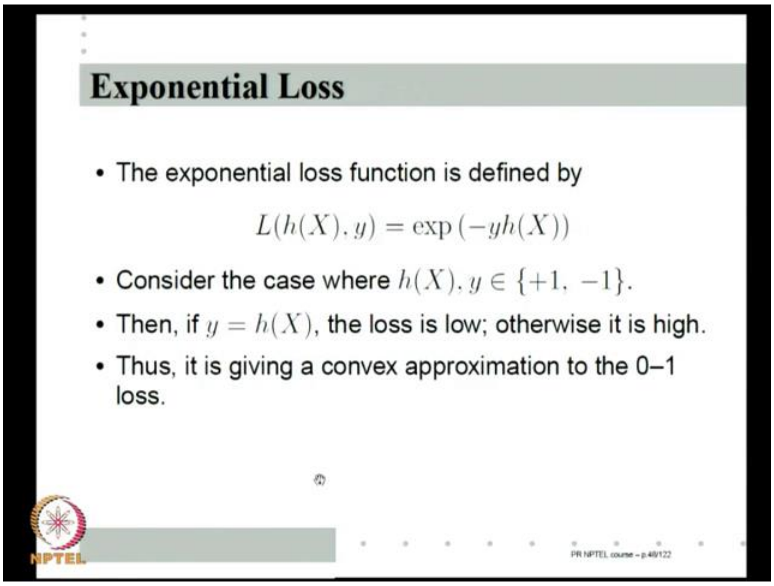
Having said that lets go back and look at AdaBoost a little more closely, all that we done so far is giving you an algorithm, which looks like a black bag box why those kind of weight updates, why that particular alpha m should be the weight in the final classifier, nothing is clear have we I mean. As a matter of fact it utilize black magic, because you do not even no where we pulled out all those expressions from. So, what we will do for the rest of this class is we look at AdaBoost from what is called from our standard risk minimization view point.

So, we have been looking at all our classifiers as essentially, you choose a appropriate loss function and then you poss your classifier learning problem, as finding a classifier with minimum risk of course, I can not calculate risk. So, we do empirical risk minimization, all our classification algorithms are essentially empirical risk minimization methods. Except that we choose different loss functions. So, the same thing we did in s v m case, first we present s v m as a interesting optimization problem then shows that it also has a risk minimization view.

So, like that we will now show that AdaBoost has an interesting risk minimization view, the loss function that is appropriate for AdaBoost is what is called an exponential loss function, that is what AdaBoost minimizes. So, we will first briefly defined exponential loss, and the set of classifier the or the hyposis space over which, this algorithm such is not individual classifiers, but ensemble classifiers. So, we have to introduce a proper you know model class for this so we will introduce a class of models what that a called additive models, which exactly capture the ensemble classifiers.

Then we show that AdaBoost is essentially, finding an additive model by minimizing empirical risk with respect to exponential loss function. Though if you think we are using exponential loss function, and want to me minimize empirical risk. While, searching over this class of additive models, the resulting algorithm looks like AdaBoost just to be more precise is not app a full minimization different a global minimum, but let say minimizer which uses a greedy heuristic. I will tell you shortly what a greedy heuristic means, but a slightly approximate way of minimizing empirical risk on the class of additive models is what AdaBoost is that is another way of looking at why AdaBoost gives good performance.

(Refer Slide Time: 21:10)

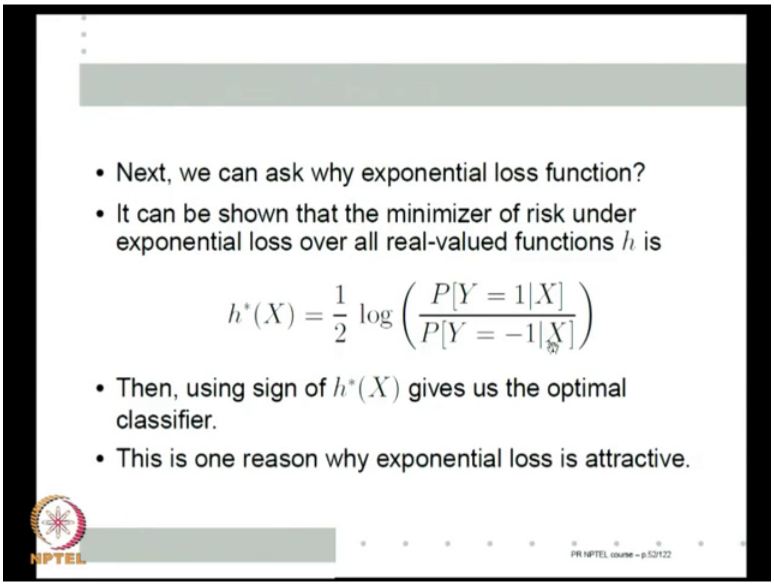


So, first let us look at the exponential loss function, exponential loss function simply l of h x coma y is exponential minus y h to h x. We have seen that many loss functions can be written as functions of y h x. So, the y h x is negative say 0, 1 loss function simply says if y h x is negative that is y and h x are opposite sign, then the loss is 1. If they are positive the loss is 0 that is of course, a non convex stuff function. Then we seen the hinge loss of s v m which tries to give a convex approximation to this.

We also saw that the square loss is another convex approximation to the 0, 1 loss function. So, similarly, exponential also is an approximation in the following sense, if you think of x and y i h x and y both are plus 1 minus 1. Then it is very easy to see that whenever, there of the same sign the loss will be exponential minus 1 and whenever, there opposite sign loss is exponential plus 1. So, that is a lot of difference. So, say essentially a loss is low if y and h x are the same sign high otherwise and of course, the same conclusion holds you even h x.

For example, takes real values because it will only came so as long as y and h x are the same sign it will be exponential minus something. Whenever, a y and h x have opposite sign exponential plus something. So, between correct and incorrect classification there is a lot of difference, and in that sense it essentially like a 0, 1 loss function for analyze 0, 1 loss function it gives me a convex loss function is a convex function y h x. So, this is a is one more convex approximation to the 0, 1 loss function.

(Refer Slide Time: 22:57)

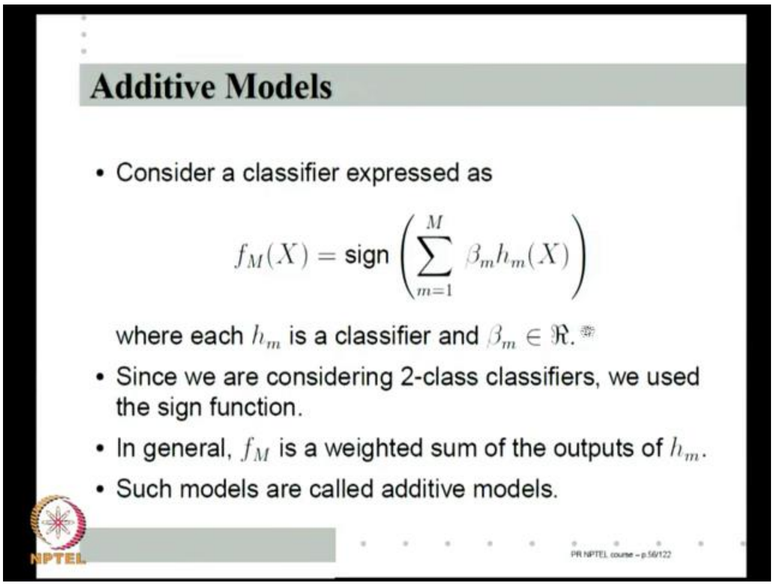


Let us of course, good enough reason to consider exponential loss, but we can still ask is this we have already so many loss functions why one more loss function. Well exponential loss has many, many interesting properties I will just state 1. It can be shown that if you have minimizing risk under exponential loss over all real valued functions h not just bind the valued functions h as we are considering here, but if you minimize the risk over all real valued functions h then the minimizer is half log half odes.

So, y is equal to 1 given x is posterior probability of class one, y is equal to minus 1 given x is posterior probability of class minus 1. So, this is essentially in a true class case this is odes for class 1. So, if the posterior probability class one is greater than posterior probability class minus 1, then this factor will be greater than 1 and log will be positive. And the other hand, if the posterior probability of class one is less than posterior probability of class minus 1. And the factor will be less than 1 and log will be negative.

So, essentially if I use sign of h star x i get the optimal classifier. So, if I can find h star which is the minimizer of risk, and exponential loss and use its sign that will give me the base optimal classifier. Of course, this is over all real valued functions, but that not the risk any these a very interesting reason why, one should look at exponential value. This is as a matter of fact one of the reason for we because of which, exponential loss function is attractive and used in many pattern recognition applications. So, with this we will we will take there we want to use exponential loss. So, let us look at what are so this is the loss function that AdaBoost uses and let us look at the model class as I said model class is additive models.

(Refer Slide Time: 24:53)

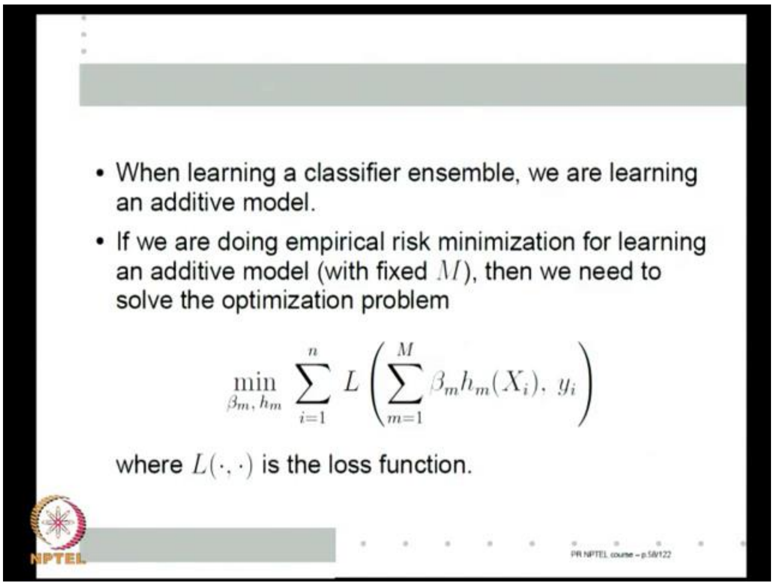


So, this is the kind of classifiers we want to concern f m of X f subscript m of X is sign of some summation m is equal to 1 to m beta m h m X. Where each h m is a classifier and each beta m is some real number. Now, we take sign essentially, because we considering two class classifiers here because we considering two class classifiers, we take this the sum of all the classifiers. And take it sign for our final classifier, but in general I do not have to take this sign. So, any f m that can be express as sum of classifiers like this is essentially, a ensemble classifier.

So, in general f m is a weighted sum of the outputs of h m of course, f m anyway have to be classifiers it could be a regressor. So, it could be just a function model. So, in general any of f m that is a weighted sum of outputs of so many h m is called a additive model. So, an additive model is simply addition of many other, many other simple models. So, if each h m is some function some regression function, or classifier. Then summation m is equal to 1 to beta m h m. So, this is what an additive model is all about.

So, the set of the classifier additive model are simply obtained as weighted sums of ordinary base classifiers or ordinary sigma functions. Let say if we are learning an ensemble where essentially, learning a negative model. As you already know this is the kind of a, this is an ensemble classifier if each of the h m are classifiers, this is like a weighted majority we have to somehow combined h m's. So, this is one we have combining so we can think of classifier ensembles.

(Refer Slide Time: 26:43)

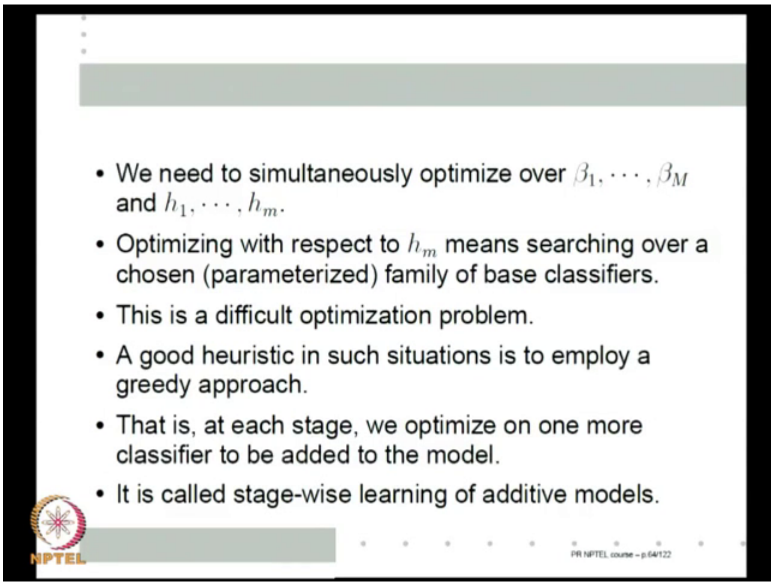


As essentially, learning a negative models. So, let us say we want to learn negative model that means, a model of this form let us assume m is fixed. So, the model of this form any specific member of this model class is specified by specific values of beta m and h m for m is equal to 1 to n. So, that is what I have to learn actually. So, let us say we want to learn an additive model by doing empirical risk minimization. What is empirical risk?

Summation i is equal to 1 to n l of your model say f of x i f m of x i coma y i so of that 1 by n does not really matter in the empirically. So, stop the 1 by n. So, i is equal to 1 to n l of what is my model m is equal to 1 to m beta m h m x i coma y i. This is my if l is the loss function, then this is the loss on x i y i because this the first term here is the value of my of a model specified by beta m and h m on x i this is y i. So, this is my loss if I sum over i is equal to 1 to n that is my empirical loss.

So, essentially if you want to do empirical risk minimization, I have to minimize this expression over beta m and h m. That is what empirical risk minimization would be, in our particular case l is a l h x coma y is exponential minus y h x. So, will be exponential minus y i into this sum, right? That will be the empirical risk with respect to exponential loss and that is what we want to minimize. Of course, this minimization is to over beta m h m m is equal to one to capital M, we have to learn all of them we have to learn beta 1, beta 2, beta m, h 1, h 2, h m.

(Refer Slide Time: 28:33)



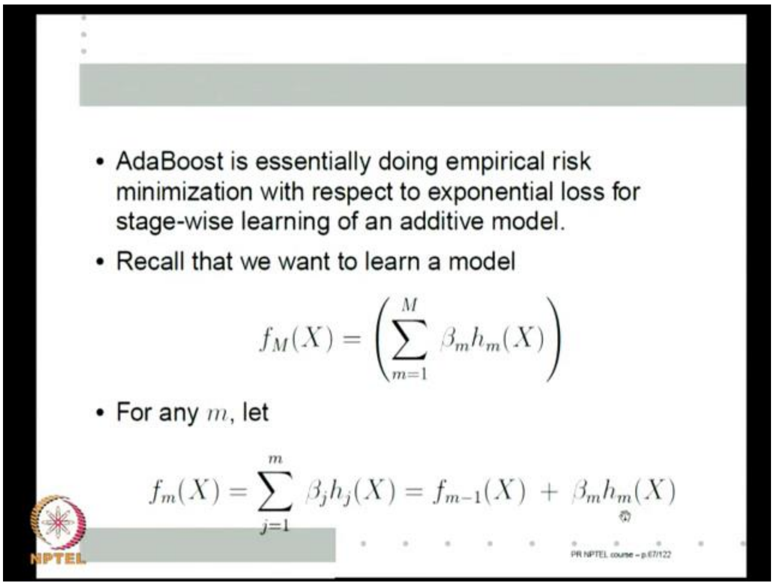
So, we have to simultaneously learn simultaneously optimize over beta 1 to beta m and h 1 to h m. What do you mean by optimize over h 1 to h m? That means, see we for each h m we are searching over a class of base classifiers, may be they will be some parameters parameterized a family of base classifier. So, those parameter vectors is what we have to learn. So, we have to optimize over so many parameter vector at a time. This is a really hard optimization problem, but whenever you have this kind of additive models what standard heuristic is what is called a greedy approach. I want the m classifiers.

So, really I am asking which particular set of hundred classifiers is best, instead of doing that what I can say is suppose, I want to use only one, what is the best classifier I take that. Now, if I want to use two classifiers I am saying only given that this is the first classifier, I am using what the second best classifier of course, that may not always be the best. If I am using the only one classifier what is best may not be even a member when I want to use two or three classifiers ensemble, but this called a greedy approach because I accumulate one by one.

So, at each stage we optimize for putting one more classifier in a model. So, if I already learnt m classifiers, when I am learning say first classifier I buck, I act as if I am only one classifier having learnt that one classifier. Now, I want to add one more while adding one more I cannot change, what I already learnt that is the characteristics of greedy approach. So, at each stage we optimize on one more classifier to be added to the model, without changing anything that we have already learnt.

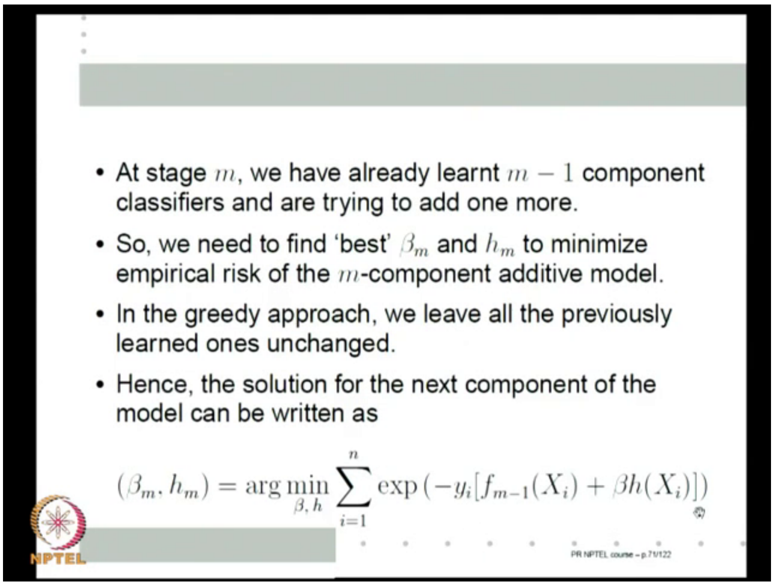
This is called a stage wise learning of additive models. So, of course, I would still be minimizing this with respect to the entire model because I am adding one more to the model so that is the model consider, but while optimizing over that model. I am only optimizing over the new element new component, I am adding to the model that is what the greedy approach means.

(Refer Slide Time: 30:39)



AdaBoost is essentially, doing empirical risk minimization with respect to exponential loss for stage wise learning of an additive model. So, is not learning the optimal additive model, but doing the greedy approximation to the optimal. So, this is the model class right we have to learn (( )). So, let say as a as a rotation this capital M can be variables. So, for any little m f m x is sum of this from j is equal to 1 to m x, which also mean f m x is f minus 1 x plus beta m h m x, that is what meant by stage wise so stage m I have already learnt the m minus 1 stage classifier. And now, I want to add one more stage to the classifier. So, this is the new model I consider, I am asking what is the best new model for me, but when I say best I can only tweak beta m h m.

(Refer Slide Time: 31:35)

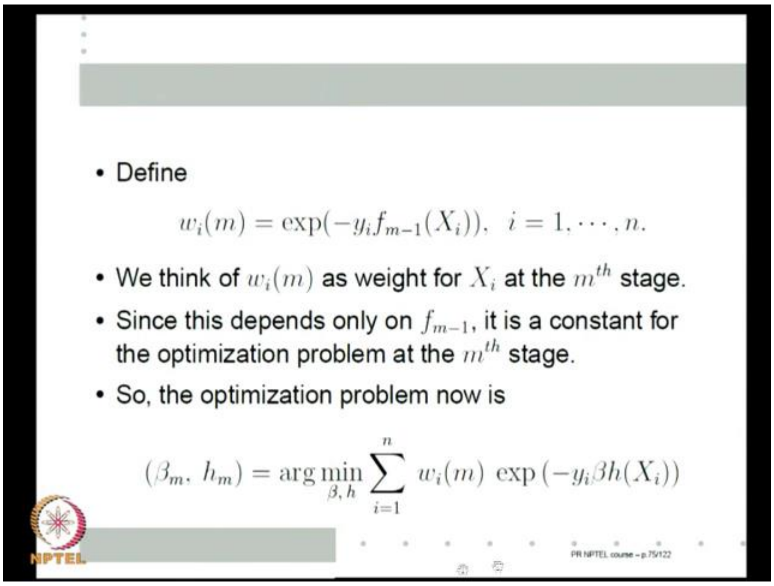


So, at stage m we already learnt m minus 1 component classifiers and I trying to add one more. So, we need to find the best beta m h m to minimize the empirical risk of the m component additive model, that is the whole idea of the approach. In the greedy approach we leave all the previously learned ones unchanged. As I said we are asking what is the best f m at the m th stage, when I say best f m I minimize empirical risk by taking f m minus 1 x plus beta m h m x as my model, but in this model the only thing that is tweak able is beta m h m. I will not tweak f m minus 1 f m minus 1, I consider fixed that is the whole idea of the greedy approach.

So, the greedy approach we leave all the previously learned classifiers unchanged. Hence, the solution for the next component of the model is see this is my model and I am using my exponential loss. So, my empirical risk is sum over the examples i is equal to 1 to n exponential minus y i into my model my model is f m minus 1 x i plus beta h x i beta h is what I want to add at the m th stage beta into h. And I am minimizing it all over possible beta on h, this is my empirical loss empirical risk under the exponential loss.

So, if I have, if I decide to use the beta, and the h as the parameters of the next component I am adding to my model then this is the empirical risk I get. So, I am minimizing it over all beta and h and that minimizer is what I am calling beta m and h m. That is what I am going to add as the n th component in my model.

(Refer Slide Time: 33:30)

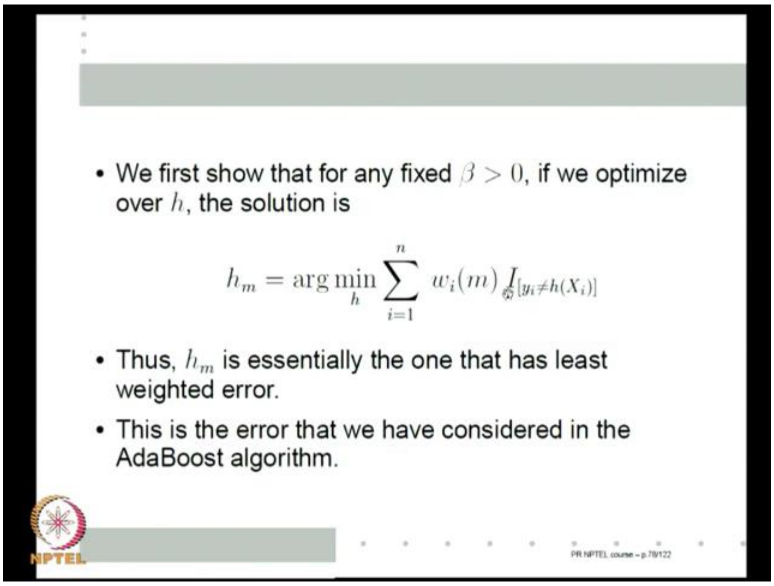


So, let us define w i m is exponential minus y i into f m m minus 1 x i. Why do I want to do that? See if I look at just the empirical risk expression, I have exponential minus y i m minus 1 x i into exponential minus y i beta h x i exponential minus y i m minus 1 x i does not change by changing beta and h so is outside this optimization. So, that is why we can take that factor out. So, w i m is exponential minus y i m minus 1 x i we can think of w i m essentially, at the weight for x i at stage m this are the we will ultimately show that these are the weights that AdaBoost is actually using.

Since, this depends only on f m minus 1, it is a constant for the optimization problem at the m th stage, it does not effect the optimization problems at m th stage. Now, I can write the optimization problem effectively, yes minimize over beta on h, i is equal to 1 to n w i m exponential minus y i beta h x i. So, I find the beta h that minimizes this and those once is what I called beta m h m because what are means. So, that will be the solution for my n th stage component classifier. So, this is what I have to do if I am doing stage wise empirical risk minimization, using exponential loss.

Then at the n th stage I want to add n th component, n th component you know model is represented beta m into h m x beta m to h m. So, my empirical risk for the model will be some, some unspecified weight at this point because that depends on the f m minus 1 part of the model into exponential minus y i beta h x i. And which is summed over i is equal to 1 to n to that be the empirical risk, and this is minimized over beta and h that the minimizing beta and h happened to be my n th component. What we have first going to show is that if I minimize this, this expression over only h by keeping beta fixed. For any fixed beta I want to minimize it over all possible classifiers h.

(Refer Slide Time: 35:58)

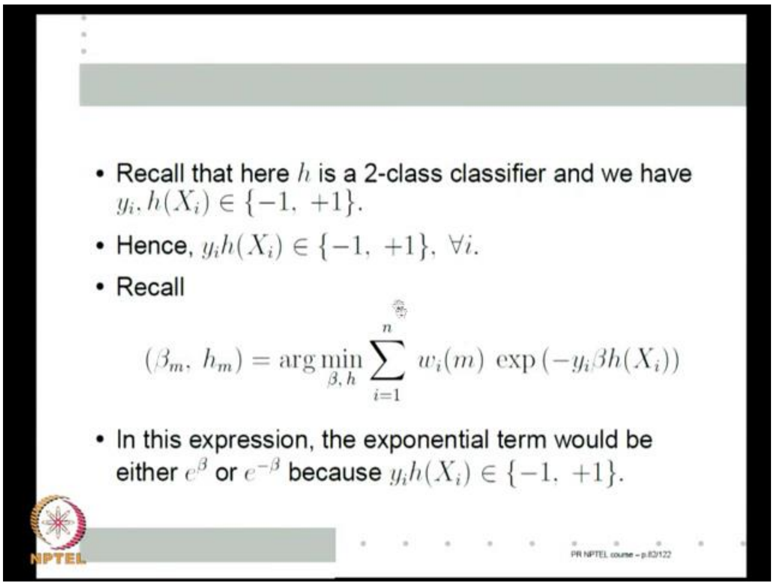


Then the h that is that minimizes this is the h that minimizes this i is equal to 1 to n, w i m indicator y i not equal to h x i this is nothing but the weighted error. See the if you if you look at i is equal to 1 to n w i m indicator y i not equal to h x i, this is the weighted error of a classifier h. Weighted according to this w i m's, and we were saying my h m is the classifier that has the least weighted error. If these w i m happened to be the same weight that we use in AdaBoost then this exactly, what AdaBoost do.

H m is essentially, the one that has the least weighted error and this is the error that we have considered in the AdaBoost algorithm. Of course, we have not at shown that this w i m which we define with respect to f m minus 1, has anything to do with our weight, but we will show that later. So, essentially we will first show that if I minimizing this, this is this is my m th stage optimization problem, for stage wise learning of negative models. So, if I did this optimization over only h for fixed beta no matter what value of fixed beta as long as beta is positive, this will be the solution.

There is very interesting because that exactly what the solution AdaBoost also looking because for any given classifier AdaBoost calculate this as the error, and we are looking for classifiers with low error. So, essentially AdaBoost is at the n th iteration AdaBoost is choosing h m that has least value for this may not be doing this optimization fully, but that is what it is tweaking.

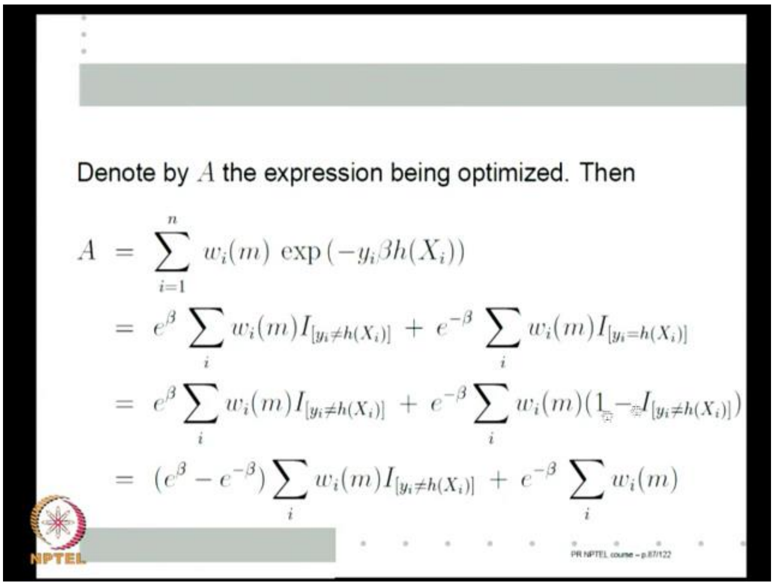
(Refer Slide Time: 37:39)



Let us show this to show this lets first remember that because both y i and h x i are minus 1 plus 1. That is the only the binary two class case what we considering because of that a product is also binary because individually, each y i and h x i minus 1 plus 1 y i into h x i is also the minus 1 plus 1. Now, this is what I want to optimize. So, inside the exponent I have y i h x i forget the beta. So, y i h x i factor takes only minus 1 plus 1 values this means, for each i this exponential fact that is a constant does not depend on i, I mean depends on i, but it can only take one of two value.

And each i it is either exponential minus beta or exponential plus beta. So, this expression can be either e power beta or e power minus beta because y i into h x i is always minus 1 plus 1. We use that to rewrite this expression let say a is the expression that is being optimize that is a is this summation, the see we optimizing we said we are optimizing this only over h now keeping beta fix. Let us call this expression some A.

(Refer Slide Time: 38:45)

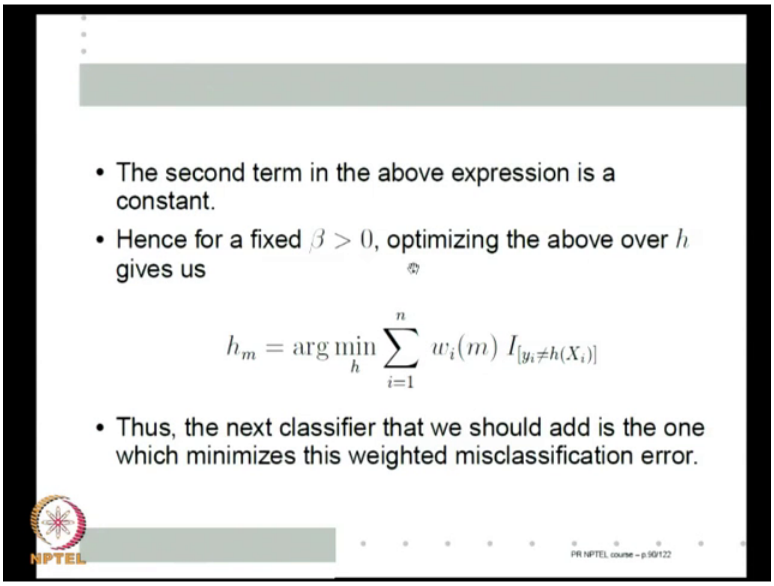


So, A is this expression i is equal to 1 to n w i m exponential minus. Now, we know that the y i is not equal to h x i this y i into h x i will be minus 1. So, this factor will be exponential beta y i is equal to h x i this factor will be exponential minus beta. So, I can write this as sum over i w i m into indicator y i not equal to h x i. So, this will take w i m for only those i for which y i is not equal to h x i. Otherwise this will be 0 so for all of them it will be e power beta plus it will be e power minus beta for all i's such that the w i is equal to h x i. So, I can write this summation. Now, like this.

Now, I can further manipulate this indicator y i is equal to h x i is same as 1 minus indicator y i not equal to h x i y i indicator y i equal to h x i is one when y i equal to h x i if y i equal to h x i y i equal to indicator y i not equal to h x i will be 0, and wise a versa. So, this indicator is nothing but 1 minus this now, I can combined this term with this negative term here to get e power beta minus e power minus beta of w i m indicator y i not equal to h x i and what is left is the constant term e power minus beta w i m, this of course, does not depend on h.

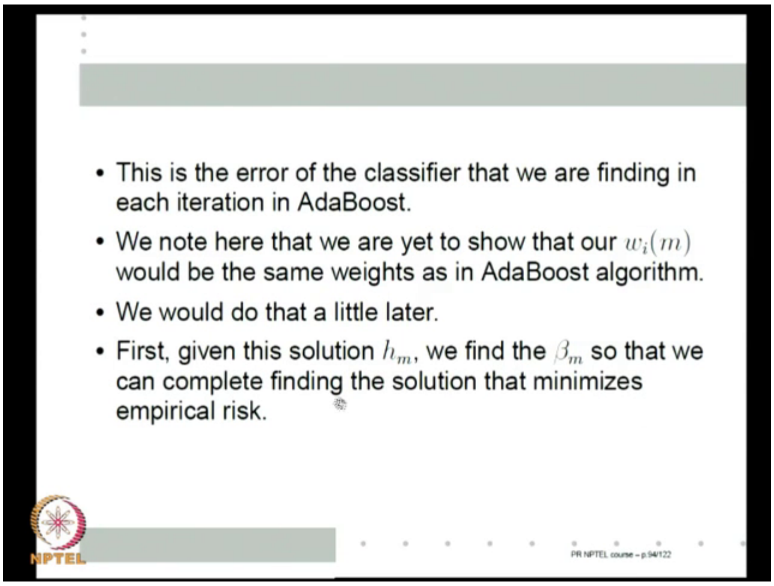
So, minimizing h i only need to minimize over this, even here as long as beta positive e power beta minus e power minus beta is a positive quantity. So, it does not effect my minimizing so minimizing this whole expression over h is same as minimizing, only this expression over h that is the result we looking for.

(Refer Slide Time: 40:23)



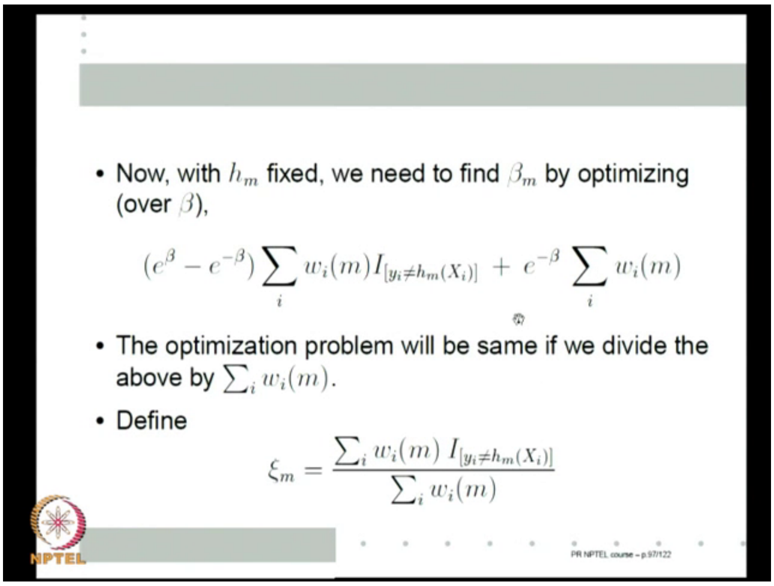
The second term in the above expression is a constant hence, for a fixed beta greater than 0 optimizing, the above over h gives us h m is equal to arg min h i is equal to 1 to n w i m y i r. That is this term this is what we wanted to show right this is interesting as we already said the next classifier that we should add is the one which minimizes, the weighted misclassification error. So, the weighted misclassification error that AdaBoost looking at is very, very important of course, provided this w m trans works to be weights there that will see the end.

(Refer Slide Time: 40:58)



So, this the error that of the classifier that we are finding in each iteration in AdaBoost also, we note of course, that we have not yet shown that w i m would be the same weights as in AdaBoost algorithm. We will do that before the class is over, but lets complete our minimizes empirical risk to minimize empirical risk, we have to find h m as well as beta m we show we first showed that no matter, what be what the beta value is as long as positive. And we can have only positive weights so for any positive beta we know how to find h m. Now, that we found h m with this h m we have to find the best beta that will be my beta m.

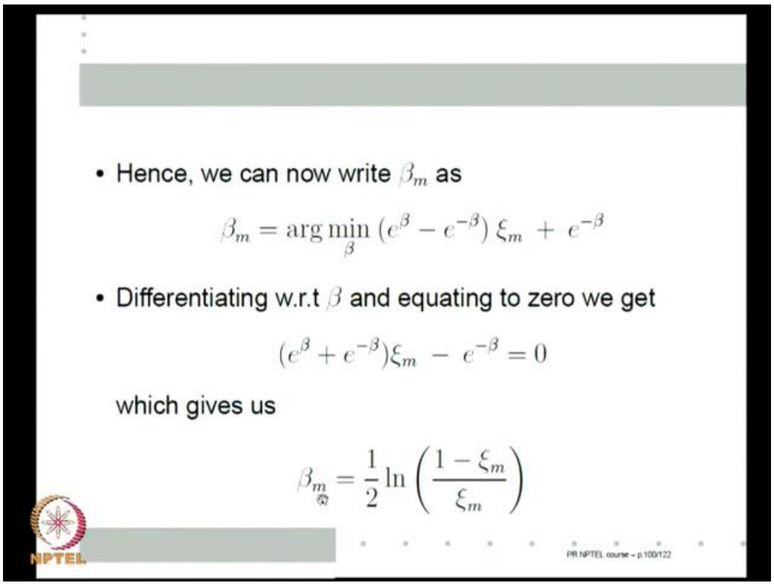
(Refer Slide Time: 40:35)



So, with the given h m we need to find beta m to optimize over beta. So, what do I have to optimize. So, you already seen this is what to been optimized originally, and that is same as this. Now, I know this term is optimize by h is equal to h m. So, I will put h m there now a putting h m there, I have to optimize this whole thing with respect to beta optimize meaning minimize with this whole thing with respect to beta. So, define beta m by optimizing over beta minimizing over beta this term. So, to make it simpler let us remember that because summation over i w i m will be a constant, see w i m's are exponential something were define it to be exponential y i f of m minus 1 x i.

So, the exponential something is always positive. So, this weights are all positive some of weights will be positive. So, if I just multiply this entire objective function by a positive constant it does not change the. So, the optimization problem will be same if we divide it by this we divided we get here summation over i w i m i of indicator of y i not equal to h m x i whole divided by summation i y w i m will give some name to it. We will call it x i m summation over i w i m indicate y i not equal to h m x i whole divided by summation over i w i m. And this term this will go away I will get only e power minus beta. So, this factor which does not depend on beta we give it some name x i m.

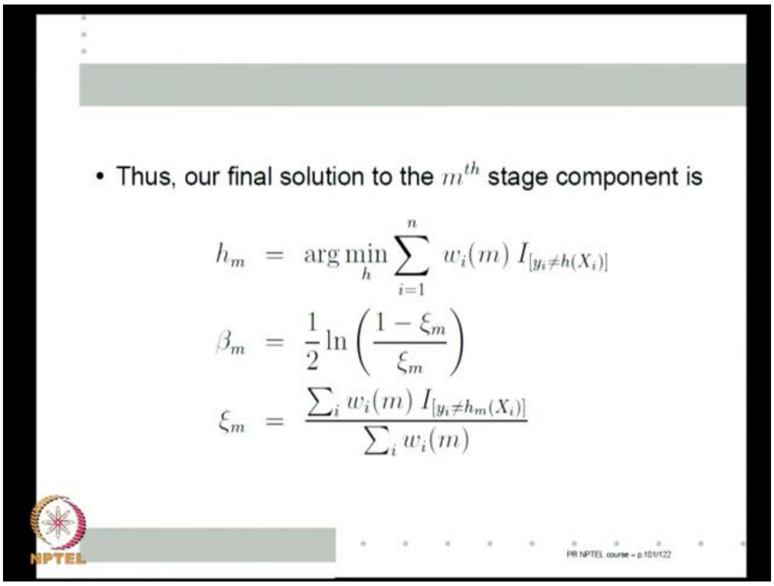
(Refer Slide Time: 43:18)



Now, what we need to a minimize over beta is e power beta minus e power minus beta x i m plus e power minus beta this is minimizing, this over beta is what gives me beta m. Now, this is very simple function of beta lets differentiate it equate to 0 if I differentiate it equate to 0. I get e power beta plus e power minus beta into xi m minus e power minus beta is equal to 0 you solve for beta. So, that gives us to the optimal beta beta m as half l n 1 minus x i m by x i m.

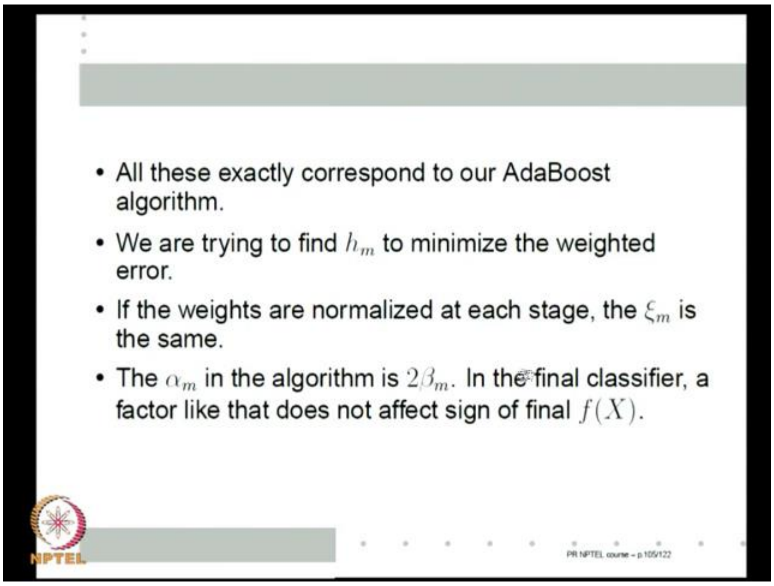
I hope all of you still remember l n 1 minus x i m by x i m from the AdaBoost algorithm that is alpha m. That is the vote that m th classifier has in the final classifier ensemble, you know model beta m is the vote that the m th classifier had the final ensemble, this beta m happens to be just half, half of alpha m.

(Refer Slide Time: 44:01)



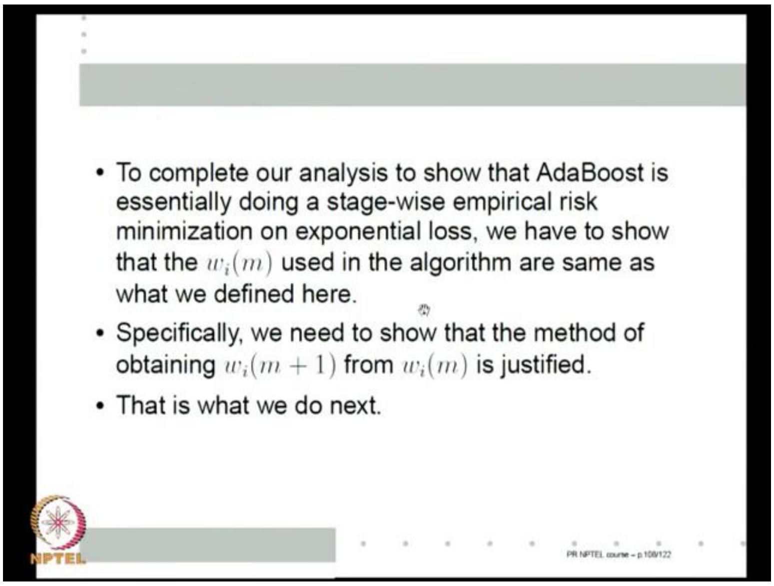
So, this is our final solution at the m th stage, the m th stage component beta m h m that you want to add. Basically, my final classifier is sign of beta 1, h 1 plus beta 2, h 2 plus beta 3, h 3 so on. So, having length up to m minus 1 I am learning m th one at the m th stage the best h m is one that minimizes the weighted error, beta m is half of l n 1 minus xi m by xi m where xi m is given by this.

(Refer Slide Time: 44:31)



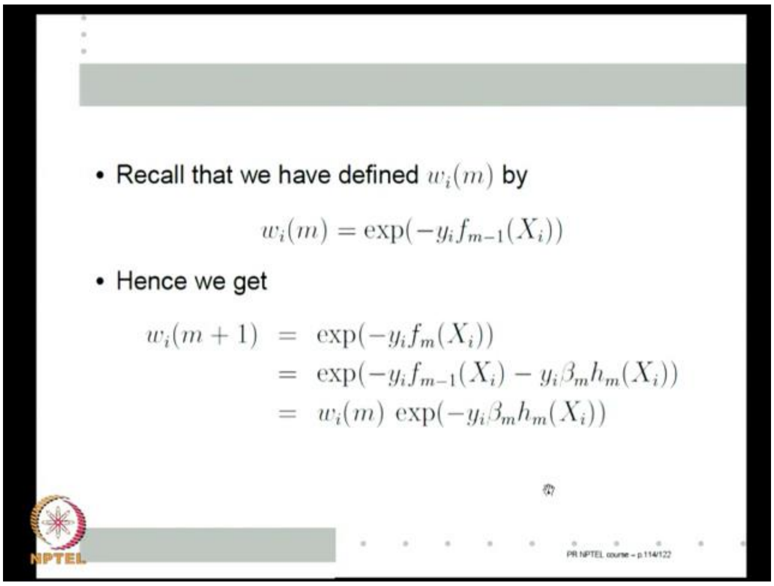
This exactly correspond to AdaBoost algorithm. Let us see one by one by we are trying to find h m to minimize the weighted error that, that is exactly what AdaBoost also does. Then we will always keeping w i m normalize. So, summation w i m is 1 so x i m is same as summation over i x i w i m indicator y i not equal to h x i. So ,that is exactly the x i m that we define in the algorithm with respect to the x i m, we define in the algorithm in AdaBoost the alpha m is l n 1 minus x i m by x i m, whereas for us the beta is twice alpha m. So, alpha m met the final vote in the weight of the vote beta is the weight of the vote in my model, but essentially because I am taking sign whether I put 2 beta m or whether I put beta m alpha is only changes a little bit. So, but any case the in the final thing the, what AdaBoost is between all alpha m here putting 2 beta.

(Refer Slide Time: 45:38)



Now, to complete analysis now so what we showing is that essentially, what we learn is what AdaBoost algorithm is doing. So, if essentially doing empirical risk minimization using exponential loss function, the factors will get are same as we get in AdaBoost if this w i is same as the way weight behaves in the AdaBoost. So, to complete our analysis we have to show that the w i m are actually, the weights used in the algorithm there. Specifically, what we have to show because any way at the at the first stage f 0 is arbitrary. So, we can an because we want to keep weight is normalized, it will be 1 by n. So, we only have to show that w i m plus 1 to w m algorithm is what I get.

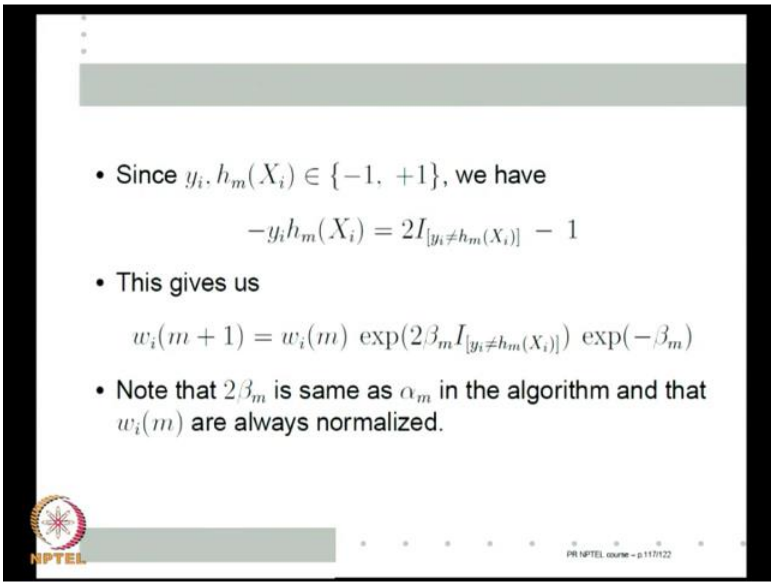
(Refer Slide Time: 46:29)



This is what we do next, recall our notation f m is j is equal to 1 to m beta j h j this additive model up to little m. So, which means, f m x is f m minus 1 x plus beta m h m x. Remembering this we know w i m is exponential minus y i into f m minus 1 x i this what how we defined w m i m in our empirical risk minimization. So, w i m plus 1 will be exponential minus y i into f m x i. Now, we know f m is this. So, I substitute that will became minus y i f m minus 1 x i, minus y i beta m h m x i.

Now, the first factor is what w i m is so this w i m into exponential minus y i beta m h m x i. So, under our definition w i m plus 1 is w i into this of course, even in the AdaBoost this next, weights are obtained by multiplying the current weights with the exponential factor. So, it is just show that this is the right exponential factor as we know because ultimately, you are traumalizing this w' s a constant does not make any difference. So, let is see how this factor is same as what we put with the algorithm. So, the factor we have is exponential minus y i beta m h m x i.

(Refer Slide Time: 48:03)

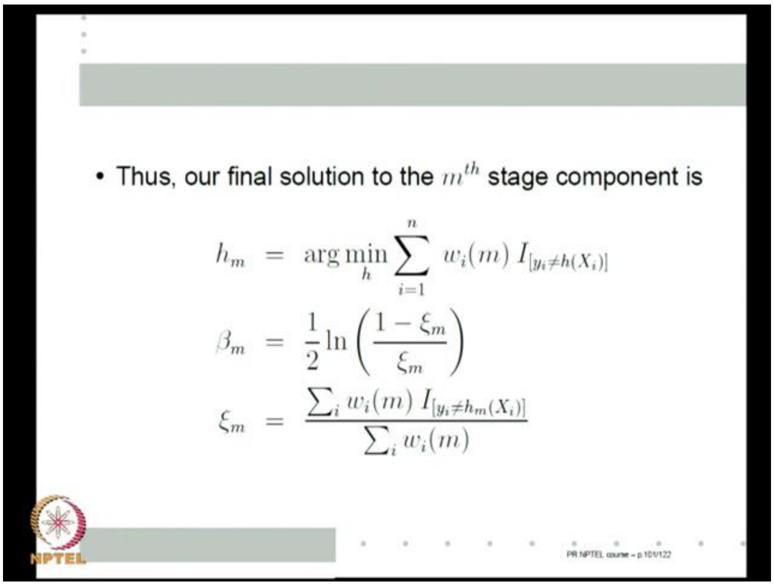


Since, y i h m x i belong to plus 1 minus 1, we can write y i h m x i as two times indicator y i not equal to h m x i minus 1 y. Suppose, w y i and h m x i are of opposite sign then this indicator will be 1. So, this will be 2 minus 1 will be 1 on this side there opposite sign. So, minus y i h m x i will be plus 1 conversely if y i and h m x i are of the same sign then this indicator will be 0. So, it will be minus 1 and this side also it is minus 1 because y and h m heart of the same side.

So, I can always write y i minus y i h m x i as this. Now, I have minus y i h m x i here is beta m into minus y i h m x i. So, I can substitute minus y i h m x i there so that gives me w i m plus 1 is w i m into exponential, what do I have y i h m x i y i h m x i is 2 into indicator minus 1. So, 2 beta m indicator y i not equal to h m x i beta m minus is already there right y i h m x i into bet m that’s why the beta m comes from. So, is exponential there is won’t be any minus because this minus y i h m x i is what is given by this.

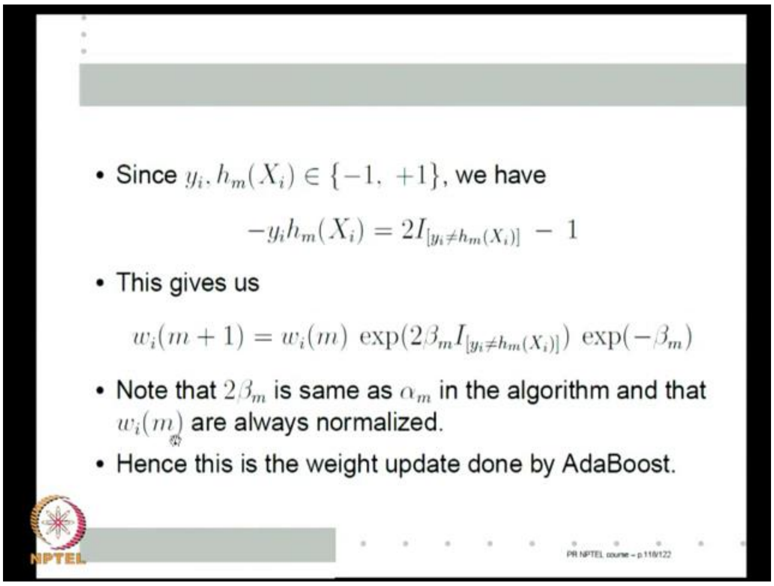
So, 2 beta m indicator of this into this minus 1 will simply give me exponential minus beta m. Now, the 2 beta m is a exactly equal to alpha m. So, this is w i m plus 1 is w i m exponential alpha m i y i not equal to h m x i into some exponential minus beta m. In the AdaBoost case, we first do w i prime m plus is w i m into exponential 2 beta m which is same as alpha m into indicator y i not equal to h m x I, and then normalizing because I am normalizing, if I multiplying all weights with the constant that does not depend on i that will go to accumulate normalization. So, this weight update is exactly same as the weight update that is used in AdaBoost, because two beta m is same as alpha m that w i m are always normalized. So, let us go back to as you seen.

(Refer Slide Time: 50:43)



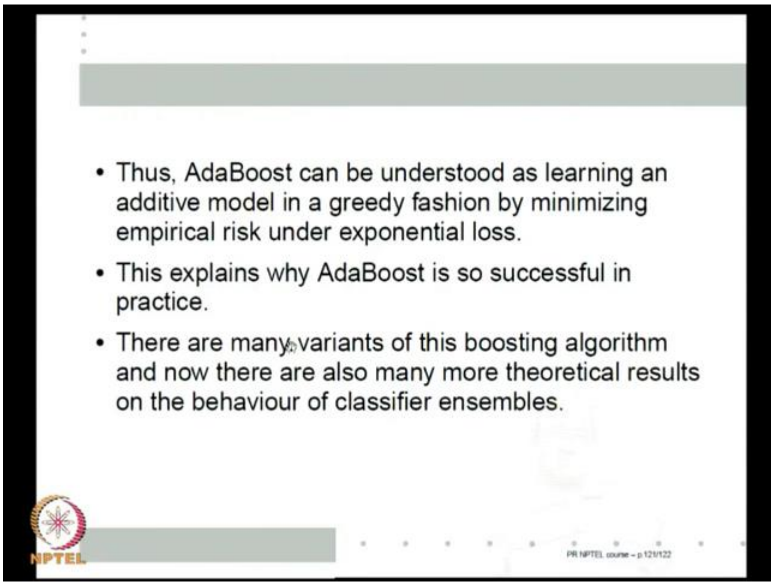
If I do stage wise minimization of empirical risk under exponential loss, this is my m th stage solution, you find h m that minimizes the weighted misclassification rate, calculus misclassification there if w i m are normalized than this is simply x i m is summation w i m this. And then the weight itself is given by this right this exactly, what the or algorithm is doing as long as w i m's are normalize weights. And that is what we have shown now, w i m exponential alpha m to i y i not equal to h m x i into some constant, this constant does not matter because I am normalizing it. So, note that 2 beta m is same as alpha m and the w i m are always normalize.

(Refer Slide Time: 40:58)



Hence, the weight update is done by AdaBoost. So, what we have shown is essentially, the AdaBoost algorithm is finding a minimizer of empirical risk not a true minimize, but in approximate minimizer using a greedy approach, under exponential loss function.

(Refer Slide Time: 51:57)



Thus AdaBoost can be understood as learning an additive model, in a greedy fashion by minimizing empirical risk under exponential loss. So, this is one way in which I can assume AdaBoost successful in practice, AdaBoost as its mentioned earlier is fairly successful even very simple almost simple minded classification algorithms. If I take enough of them, learn through AdaBoost, learn through this proper weighting gives very good classifiers.

A matter of fact to for AdaBoost work well it is often more convenient, if the component classifiers are a little less accurate, in the sense if I am using very high complexity component classifiers, one classifier may have very high accuracy and it may dominate everything. So, it is often better not to do that. All kinds of things have been used including neural networks, (( )), and so on. And today, there are many other variants of this boosting algorithm. And as a matter of fact more theoretical also available in terms of information theory and so on, as to avoid this boosting vector.

Boosting is one very major new area in pattern recognition new meaning, it in last ten-fifteen years this is a great idea to come up into pattern recognition. So, there many, many more new results in boosting, but you know this a interactive course. So, we just consult one basic boosting algorithm, but this is one this most often use and very useful in practice. So, this brings us to the close of the course we have completed all our lectures.

So, I will just briefly review back what all we have done. We started with the basic idea pattern recognition feature extraction classification at two step model, we seen that learning functions in general has the same flavor. So, was I am learning any model of f, the basic problem is your you give me examples x i y i, i is equal to 1 to n and we have to learn some model f, such that f of x i is a good approximation to y i not just in the training set, but any future things at I may get.

Now, we let this problem in general look many examples and then decided there we look it at only the statistical sense, this entire course is only about statistical pattern recognition. So, we looked at decision statistical decision theory we looked Bayesian decision making. So, we defined loss function risk, what is the optimal base classifier. And shown that essentially, if I can calculate posterior probabilities of classes then I can implement a optimal classifier, optimal in the sense minimizing misclassification rate or any other kind of a cost for misclassification.

We have also seen that minimizing miss misclassification may not be the only criterion, then other criterion's are such as name, and person where we want to keep one kind of error always low, we seen how that can such classifiers can be derived. Then we spend some time asking, if this is how we want the implement to statistical pattern recognition. How can we implement them? So, we need to calculate posterior probabilities which means we need class conditional densities and prior probabilities, and we have spend time in asking how we can estimate them.

So, we consider both parametric and non parametric ways for estimating in the parametric method, we looked at both maximum likelihood and Bayesian estimation, we looked at nearest neighbor and kernel density estimation, non parametric estimates. And then we also looked at mixture densities, these are all for single densities, but we also looked mixture densities and the m l estimate using e m algorithm. So, one method of implementing classifiers would be through density estimation, then we looked at other methods. Then we looked at discriminate functions, linear discriminate functions. We looked at perceptron we looked at adalene which is a linear regression learning algorithm.

We in general looked at least squared method of learning linear models, both for classification, regression looked at the relationship between least squared. And you know as a Bayesian or m l estimate under some assumed models. We looked at regularization, why regularization needed, why model complexity is an issue. We looked at from the (( )) algorithm all the way to logistic regression, and fusilier discriminate. Then we step back and looked at statistical learning theory, we developed the theme of risk minimization little more, and showed why empirical risk minimization is useful.

When it will work, when it will not work, what is the complexity of model class, we characterize the complexity of model class using (( )). We explain what (( )) is and essentially, justified empirical risk minimization. Then we moved on to looking at non-linear classifiers, we only consider two classes for non-linear classifiers. One based on neural networks both feed forward network is sigma activation as well as real bases function networks, and then we looked at SVM's. Unfortunately due to lack of time, we did not concert one other very important topic that is among the first level topic that should be cover in a base pattern recognition course.

One topic that is not covered in this course is dysenteries. That is a very good class of classifiers, but for lack of time we did not cover the dysenteries, but we cover the other two major class of non-linear classifiers neural networks and SVM's. In SVM's we also looked at kernel base methods, which are now the most popular pattern recognition methods and lot of theories the error on them. After that we looked at model assessment, model estimation, bias variance trade off. We looked at how to estimate the final error of the model, how to chose model parameters, we looked at class validation, we looked at bootstrapping, arising out of that we ended the course by looking at classifiers, and some bulls, bagging, and boosting, and AdaBoost. I hope you people benefit it out of this course.

Thank you very much.