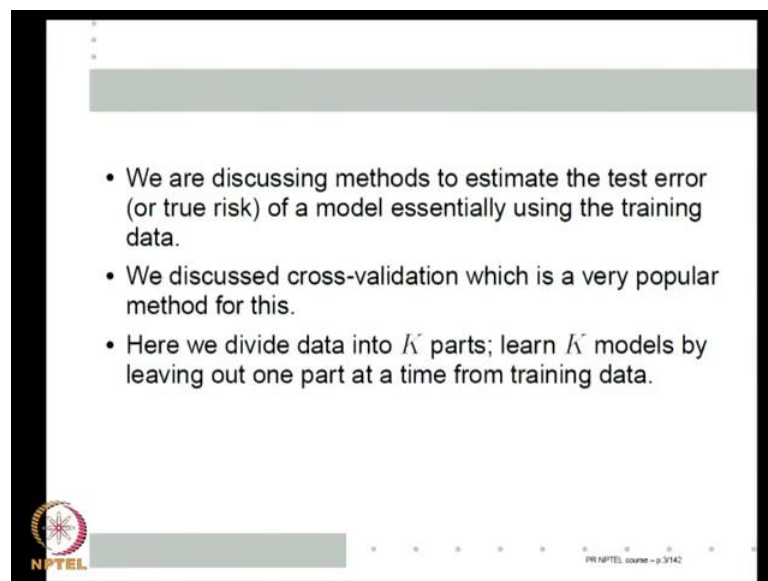


Pattern Recognition
Prof. P. S. Sastry
Department of Electronics and Communication Engineering
Indian Institute Of Science, Bangalore

Lecture - 41
Bootstrap, Bagging and Boosting; Classifier Ensembles; AdaBoost

Hello and welcome to this next lecture on pattern recognition, we have been discussing last class on model selection, model assessment. Mainly techniques that can use training data itself to estimate test error, and you know we discuss various issues such as bias, bias dilemma. How one can think of test error and so on, but in cases where training data is small, which is the majority of applications. We have been looking at various techniques so, that we can use all the training data for learning a classifier and as well as for estimating final errors, for model selection everything. The one method that we considered in detail is the cross validation.

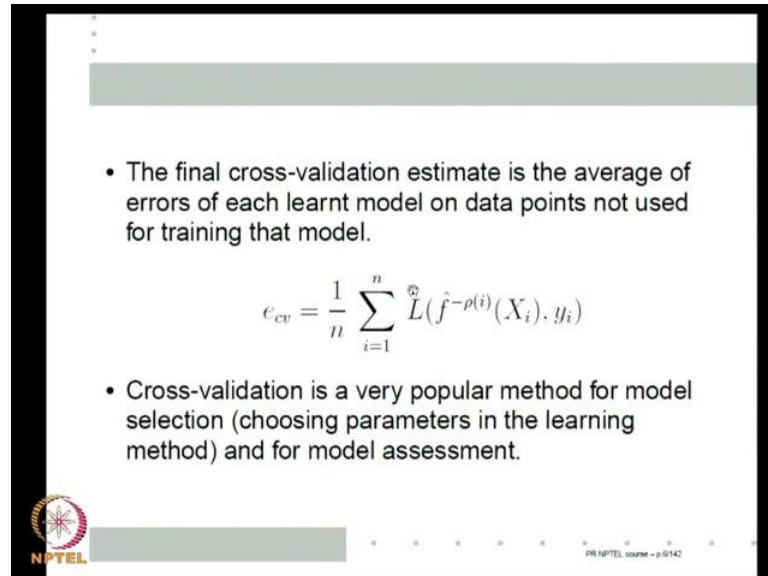
(Refer Slide Time: 01:19)



So, the cross validation as we seen is a very popular method for doing this essentially, using the same training data. We learn a model as well as make an estimate of the true test error. So, the idea for cross validation is the following we divide data into k parts 1, 2, 3, 4 k . So, first time we use parts 2, 3, 4 all the way up to k for training and then test this classifier find the classifier error on part 1. Next, time we sue parts 1, 3, 5, 7 and so on and then test that classifier on the left out part and so on. So, essentially we divide the

data into k parts, then learn k models by each time leaving over 1 part from the training data which is what is used for testing.

(Refer Slide Time: 02:19)



- The final cross-validation estimate is the average of errors of each learnt model on data points not used for training that model.

$$e_{cv} = \frac{1}{n} \sum_{i=1}^n L(\hat{f}^{-\rho(i)}(X_i), y_i)$$

- Cross-validation is a very popular method for model selection (choosing parameters in the learning method) and for model assessment.

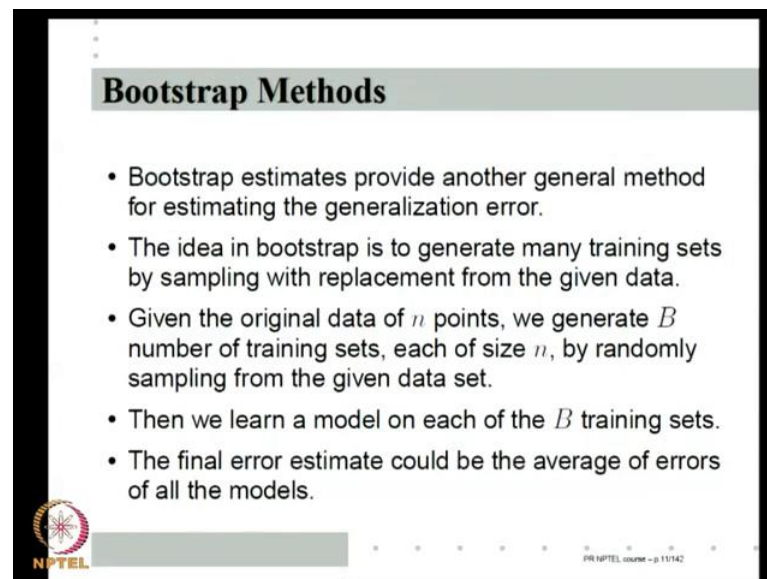
So, using this the final cross validation error estimate, the estimate under cross validation for test error would be the average of errors which learnt model, on data points not used in the training of that model. So, each model in each model we are using k minus 1 parts for training one part we are not using in training that model. So, we test each model on the path that was not used in training that model, and we take average of all this and that is our final cross validation of the estimate. So, specifically this is what the cross validation estimate is.

So, f of minus rho i rho of i denotes the part number into which X_i is put. So, f minus rho i is the model that is learnt on part rho i that is when we left out part rho i . So, the training data at X_i is tested on that particular f as we have seen in last class, this is very popular method for model selection. Essentially what you do is when we want to do model selection there will be some alpha some parameter of the model class right. So, for different values of the model class parameter, we calculate the cross validation error estimate, and then choose the best model class based on the cross validation error estimate.

As we seen model class parameters could be essentially, parameters learning algorithm you know step size is the c value in $s v m$, or the number of hidden nodes in the neural

network and so on so forth. So, generally cross validation is very often used for model selection and it is also used for final model assessment. So, this is one way of estimating the true risk model by using the training data itself so to say. So, here like in cross validation we are using all the data for training and also for testing right. When data size is small we want each data element to pay. So, this, these are very attractive techniques.

(Refer Slide Time: 04:27)



Bootstrap Methods

- Bootstrap estimates provide another general method for estimating the generalization error.
- The idea in bootstrap is to generate many training sets by sampling with replacement from the given data.
- Given the original data of n points, we generate B number of training sets, each of size n , by randomly sampling from the given data set.
- Then we learn a model on each of the B training sets.
- The final error estimate could be the average of errors of all the models.

NPTEL
© NPTEL course - p. 11142

This another technique which you also briefly considered last class, which are called the bootstrap methods. Bootstrap estimates provide another general technique for doing similar kind of estimation, like in cross validation in bootstrap. Also we learn many, many models, but essentially in cross validation deterministically we are separating over data into k training sets different training sets. Of course, here large number of things, but also each training sets are some part of its own right, because every time we are leaving one of the k parts. But here in bootstrap we generated many training sets by simply random sampling with replacement from the given data.

We have got data points 1 to n , so every time we want a data we want a new training data set, what we are do is we keep generating random numbers between 1 and n . Whichever number comes up we pick up that particular point of course, some points may be picked up twice some points may not be picked up like this. So, essentially we generate many multiple training data sets by sampling with replacement from our one given training data set. So, if given n original data set of n points, we first generate n random numbers

each uniform distribution between 1 and n that gives me one training data set. Once again I repeat it, I get another training data set. I can like this get as many training data sets as I want.

So, we generate let us say capital V number versus training data sets each will be of size n by random sampling from the given data set. So, they may have of course, many things in common some of them may be some of the points may be unique data set. Essentially, the variation in different n eta sets is coming because of the random sampling here. Then like in the cross validation case we learn model in each of the training data sets, with beta training data sets. Let us say so we learn b models each capital B models each one on one of the training sets. So, now we have b of f hat learnt b number of f hats learnt each one from its training data set. Now, we have the original data set anyway so, I can choose the final error as the average of errors of all the models on the original data set I have.

(Refer Slide Time: 06:51)

- Unlike in cross-validation, here we can have as many training sets as we want (all with size n).
- However, they may all be very similar.
- Let \hat{f}^b denote the model learnt using the b^{th} bootstrap sample, $b = 1, \dots, B$.
- The final bootstrap estimate of error is

$$e_{boot}^1 = \frac{1}{B} \sum_{b=1}^B \frac{1}{n} \sum_{i=1}^n L(\hat{f}^b(X_i), y_i)$$

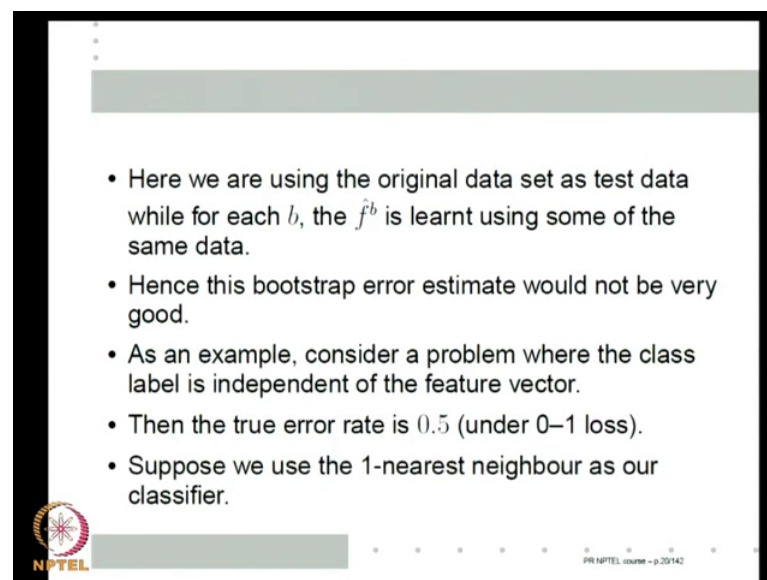
So, basically a what is the difference between cross validation bootstrap unlike in cross validation here we can have as many training sets, as we want and all of them of the same size right because the variation comes by random sampling in cross validation because I am actually, partitioning data at most I can participation to n parts the leave on out cross validation be on that I cannot partitionate, but here because I am random sampling, I can generate as many training data sets as I want. All of them will have size

in the few of set of course, is that while I can generate thousands of them many of them might be very similar because I am just sampling with replacement from the same underlying data.

So, maybe they might all be similar any case, let us first look at the error estimate let us say f at b at small b . Denote the model learnt using the b -th bootstrap sample b takes values 1 to capital B that many bootstrap data sets are there. So, let us say f at b denotes the one learnt on the b -th one then my error estimate is very simple I will call it e_1 boot the one because we will have another error estimate later on, so this is bootstrap error. So, if I take the inner term $\frac{1}{n} \sum_{i=1}^n f$ at b $X_i y_i$ this is the average error on the original data set of the b th model that is learnt.

Now, I take average of all the b models the average errors of all the b models that is my final error estimate. Seems quite reasonable except that there is only one problem see essentially, this is like for each model f_b . I am using the entire original data set as a test set, because this is like a test like error on the entire original data set, and then I am taking average of these errors.

(Refer Slide Time: 08:49)



So, here we are using the original data set as a test data set so to say, but for learning f at b we used some of the same data points. So, obviously because f at b as used some of the $X_i y_i$ used some of the $X_i y_i$ for learning f at b this error estimate is likely to be put matter of fact, it can be very poor. Let us take an example, let us take an extreme case

where the feature in the class label are independent. That means, I get some feature vector, but the class label for each feature vector each pattern is independent with 0.5 probability is class 1, 0.5 probability the other class.

Now, then if I am using a 0-1 loss function my true error rate is just 0.5 because class label is not correlated at all with the feature vector. So, because class label is random the best I can do is 0.5. Now, let us say I am using a nearest neighbor classifier one nearest neighbor classifier. So, my classifier would be I take my training data set stored it as my prototype every time I get a new pattern, I look for the nearest neighbor in my training data set further. Give the same class in the like k near is the one nearest neighbor nearest neighbor classifier.

(Refer Slide Time: 10:20)

- Then on any X_i , the classification by \hat{f}^b would be correct if X_i is in the b^{th} bootstrap sample.
- Otherwise, it would be correct with probability 0.5.
- Now,

$$P[X_i \in \text{bootstrap sample } b] = 1 - \left(1 - \frac{1}{n}\right)^n$$

$$\approx 1 - e^{-1}$$

$$= 0.632$$

Now, what happens because you are using the same x_1, x_2, x_n for testing on any X_i the classification with \hat{f} would be correct, if X_i is in the b^{th} bootstrap sample, if X_i is in the b^{th} bootstrap sample. Then X_i will be one of the prototypes \hat{f}^b . Now, when you are testing using X_i , I ask what is the closest pattern to X_i it will be X_i itself and I give the correct class label. So, on any X_i the classification by \hat{f}^b would be correct if the particular X_i happen to be in the b^{th} bootstrap data set. If it is not there then obviously there will be something like that is nearest.

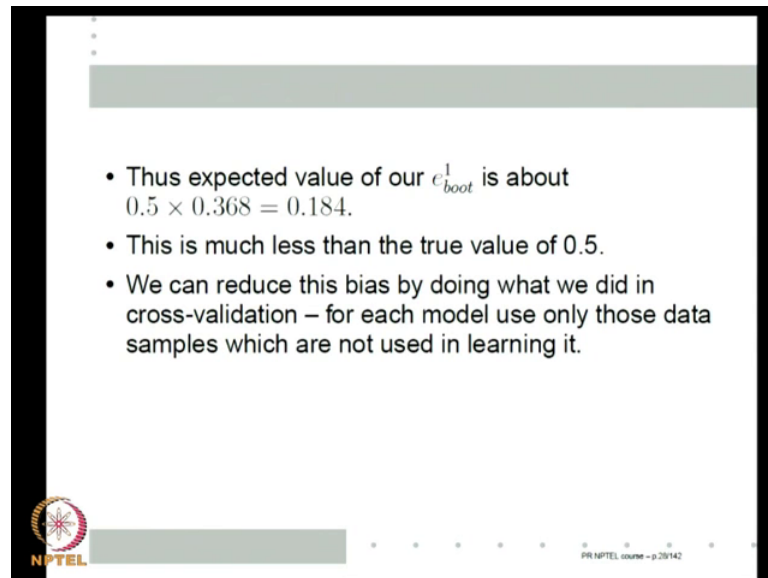
But it is really does not matter because class labels are random I can get correct class label with probability only 0.5. So, for my one nearest neighbor the bootstrap error

expected bootstrap error would be 0.5 times, the probability that a random X_i will not be in the random bootstrap as an arbitrary bootstrap sample. Let us we can calculate this probability let us say I want to calculate the probability for some i and some b . Probability X_i belongs to bootstrap sample b . How do I generate the bootstrap sample, I keep the sampling uniformly over the 1 to n samples.

So, the only way is first let us ask when will X_i not belonging the bootstrap sample. So, when I made 1, when I did 1 sampling from the data a specific X_i not coming is $1 - \frac{1}{n}$. I did n such samples, so the specific X_i not coming is $1 - \frac{1}{n}$ into the power n . So, X_i coming is $1 - (1 - \frac{1}{n})^n$ it is clear, because each sample can come with probability $\frac{1}{n}$. Whenever I sample once a specific X_i not coming is $(1 - \frac{1}{n})^n$.

So, when I do n time sample the specific X_i not coming on all the n times is like getting n heads in n tosses of a coin. So, $(1 - \frac{1}{n})^n$ and if I want the probability X_i is in it is $1 - (1 - \frac{1}{n})^n$. If n is large we know $(1 - \frac{1}{n})^n$ goes to e^{-1} so goes to e^{-1} . So, this is roughly $1 - e^{-1}$, which is about 0.63. So, it does not depend on i or b for any i and b probability of X_i belong to bootstrap p is about 0.63 for large this approximation of this e^{-1} is fairly good. So, which means, the expected value of our e^{-1} boot is because this is 3, 6, 2 X_i not being in the bootstrap sample is 0.378.

(Refer Slide Time: 13:30)



The slide contains three bullet points and a footer. The footer includes the NPTEL logo and the text 'PR NPTEL course - p.20142'.

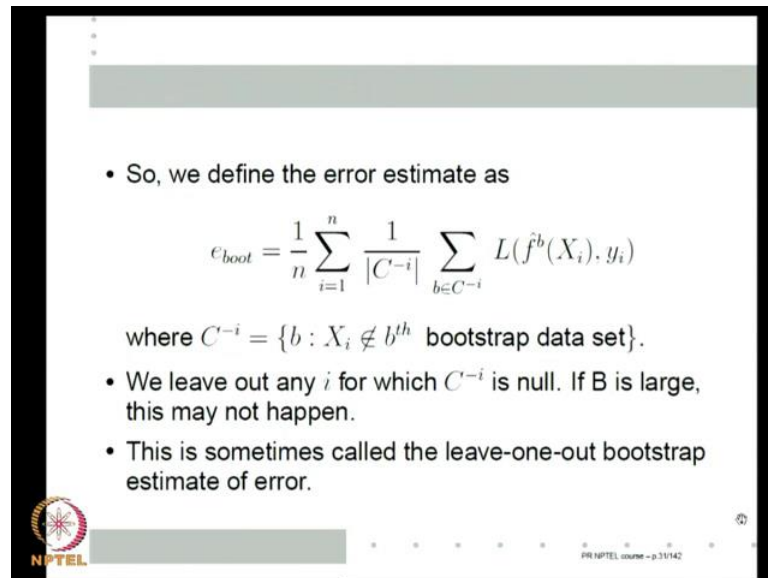
- Thus expected value of our e_{boot}^1 is about $0.5 \times 0.368 = 0.184$.
- This is much less than the true value of 0.5.
- We can reduce this bias by doing what we did in cross-validation – for each model use only those data samples which are not used in learning it.

NPTEL PR NPTEL course - p.20142

0.638 and into 0.5 will give me about 0.184 that is we see in that if it is not if it is in anyway do not make error, if it not in I will make error to probability 0.5, because it is not being in the bootstrap sample is about 0.36. So, that into 0.5 is about 0.18 this is the expected value of e boot. This is my estimate, the expectation of the estimate is 0.184 whereas, true value of the error rate is 0.5.

So, the expectation of e boot is very far from the true value. So, we have large amount of bias so, e boot e 1 boot is a high bias highly biased estimator of the true risk of the classifier and is biased down. Obviously, so it is underestimating the error rate. We can actually reduce this bias by using what we did in cross validation that is test each model only on those data samples, which are not used in learning it. Now, of course, because the data samples are obtained randomly just notationally or writing this is a little complicated, but let us write it anyway.

(Refer Slide Time: 14:55)



• So, we define the error estimate as

$$e_{boot} = \frac{1}{n} \sum_{i=1}^n \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} L(\hat{f}^b(X_i), y_i)$$

where $C^{-i} = \{b : X_i \notin b^{th} \text{ bootstrap data set}\}$.

- We leave out any i for which C^{-i} is null. If B is large, this may not happen.
- This is sometimes called the leave-one-out bootstrap estimate of error.

NPTEL PRE NPTEL course - p-31142

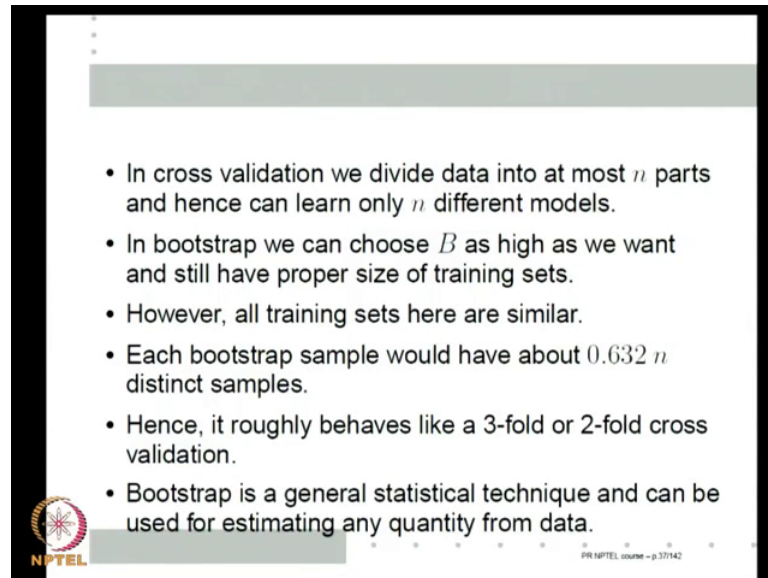
Now, we will define another error estimate this time I called it e_{boot} that one is not there. So, for each i I can test i on all \hat{f}^b in which it is not used. So, let us define c minus i the set c minus i to be the indices b bootstrap sample indices, bootstrap data sample indices such that X_i is not in that bootstrap data set. So, c minus i gives me the set of all bootstrap the indices of all bootstrap data sets in which X_i is not there. So, for each of this little b in c minus i , I can take the error on X_i so because I want average, I divide it by the number of elements in c minus i . This is the contribution to the final error estimate of X_i and I sum it through over i equal to 1 to n , and divide by 1 by n hop its clear.

Essentially each X_i can be used to test \hat{f}^b for all b such that such X_i is not used in training \hat{f}^b . So, the set of b on which X_i is not used in training \hat{f}^b is c minus i . So, I sum this over this b belonging to c minus i , and divide by the number of elements in c minus i that gives me the amount of error estimate contribution from X_i , I get it from all the training samples. Of course, there is a possibility that for some particular X_i c minus i is null, if c minus i is null I just do not take that X_i because you know I cannot abide by 0.

So, I will just omit that particular i for omit that particular i I make it 1 by n minus 1 right, but if n is large this is unlikely to happen, if n is large that I have been doing many large sampling. There is a particular X_i which never comes into any of the samples is a

is a very rare event. So, if capital B sufficiently large this case may not happen and in case this kind of a bootstrap estimate is sometimes called leave one out bootstrap estimate error of error. So, bootstrap is another good way to estimate test error.

(Refer Slide Time: 17:08)



- In cross validation we divide data into at most n parts and hence can learn only n different models.
- In bootstrap we can choose B as high as we want and still have proper size of training sets.
- However, all training sets here are similar.
- Each bootstrap sample would have about $0.632 n$ distinct samples.
- Hence, it roughly behaves like a 3-fold or 2-fold cross validation.
- Bootstrap is a general statistical technique and can be used for estimating any quantity from data.

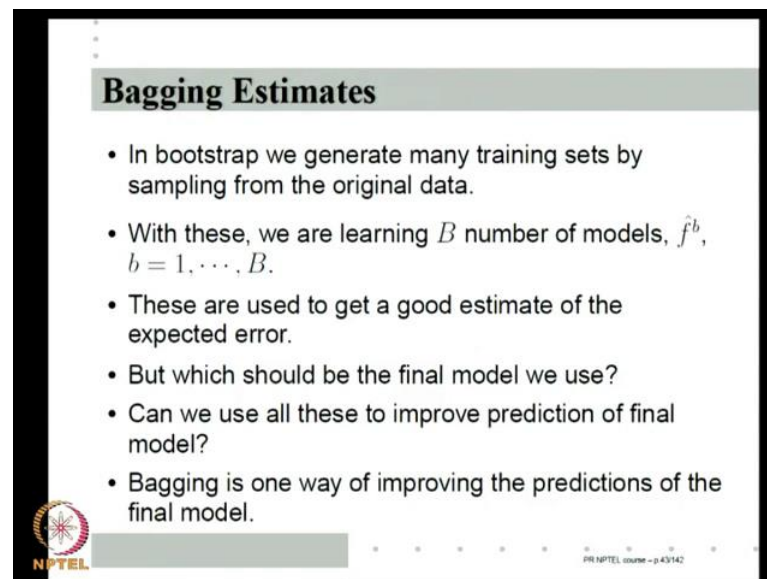
Basically, to contrast with class validation, in class validation we divide data into at most n parts and hence, there can be only n different models you can learn. On the other hand in bootstrap we can choose b as high as we want right, and still have proper size of training sets. Now, because we average it to about so many models we are likely to get low variance in the error estimate, but on the other hand the training sets that we can get by like this by sampling with replacement are all, are all quite similar right.

See for example, in cross validation we said we like to do 5 or 10 fold cross validation so, that you know we can keep the variance variation in the training data set. If we leave one out the variation deferent training data sets is very small. So, the models learnt may be similar so, the estimate may not be very good, the same thing may happen here. Another way of looking it at is that from what we calculated, any X_i being in any particular bootstrap sample is about 0.63, which roughly means that each bootstrap sample would have about $0.632 n$ distinct samples.

So, this is like a three fold or two fold cross validation right. So, it contains about 60 percent of the 63 percent of the total number of samples. So, if I three fold cross validation, each training data will contain 66 percent of the original data. Two fold cross

validation contain 50 percent of. So, essentially for the biases and so on, it will be here will like this, but of course, because I am generating many b 's it will be not just like three fold cross validation, the averaging because of b 's can give me a better estimate than a three fold cross validation. So, there are both plus s and minus s for bootstrap in relation to cross validation. This also has very general statistical technique can be used for estimating many quantities of data. This has also been used as a set of variance reduction technique in your estimate.

(Refer Slide Time: 19:48)



Bagging Estimates

- In bootstrap we generate many training sets by sampling from the original data.
- With these, we are learning B number of models, \hat{f}^b , $b = 1, \dots, B$.
- These are used to get a good estimate of the expected error.
- But which should be the final model we use?
- Can we use all these to improve prediction of final model?
- Bagging is one way of improving the predictions of the final model.

NPTEL PRE NPTEL course - p 43142

Next, we consider what is called bagging of the estimates, let us take bootstrap of course, what I am saying about bootstrap is also true of cross validation, but in bootstrap we are generating many different training data sets by sampling from the original data. So, we have one set of n examples $X_i y_i$. Then we generating many different training data sets by sampling with replacement from the we are doing this essentially. So, that we learn b number of models \hat{f}^b because we it because b data sets. What is the purpose of generating all these models?

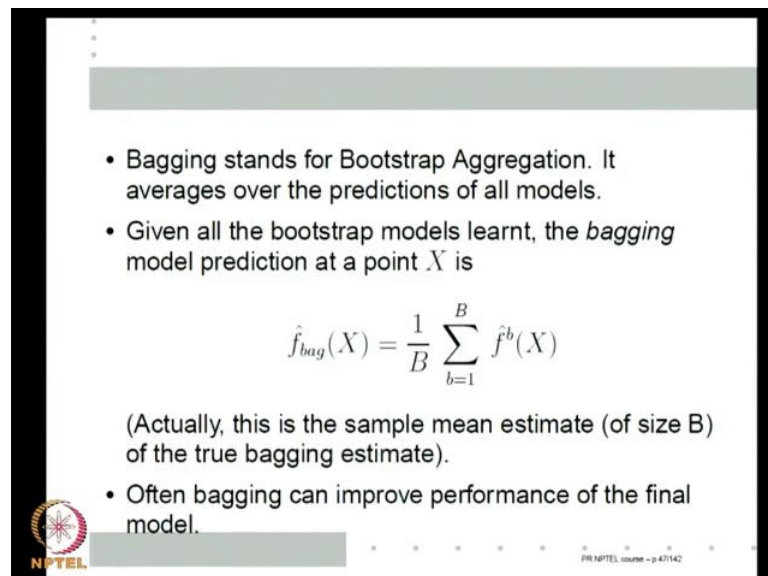
We use all these models together to get a good estimate of the expected error, this could be for model selection. So, we want to know how good a model class parameter α is so far that class α I learn so many models, and then I know how good it is? Ultimately, I can find what is the best α to use or I may use a use it for final model assessment and same thing is true for cross validation also. There unlike here b being

arbitrary that may not be arbitrary, but ten fold cross validation or whatever we are learning many different models.

Now, I got a error estimate I finished all this let us say I did model selection, I got the particular alpha for that alpha. I have many \hat{f}_b alphas or I have anyway learnt many \hat{f}_b 's. Finally, did my model assessment, but along the way I have learnt so many models. So, which particular model should I use as the final model, should I use as the final model should I throw up all these all away. Once again, relearn after all there is no difference from learning with one more training data set.

Say for example, cross validation any one of the models can be used as the final model, which model should I use a good idea is why cannot we use all this models to improve the prediction right. Any way I have learn so many models from so many different data set, why should I throw some of them away when you give me new patterns. From now on, I can use all the models and may be somehow combine them to get a better prediction. Now, this is the basic idea of bagging, bagging is one way of improving the prediction of the final model. So, instead of using any one of them if I use all of them then I may get a better prediction. So, what is the simplest way of using all of them?

(Refer Slide Time: 22:09)



The slide contains the following text:

- Bagging stands for Bootstrap Aggregation. It averages over the predictions of all models.
- Given all the bootstrap models learnt, the *bagging* model prediction at a point X is

$$\hat{f}_{bag}(X) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(X)$$

(Actually, this is the sample mean estimate (of size B) of the true bagging estimate).

- Often bagging can improve performance of the final model.

The slide also features the NPTEL logo in the bottom left corner and the text 'PR NPTEL course - p 47142' in the bottom right corner.

Average that is what simplest bagging estimate is so, actually bagging is a kind of word derived from bootstrap aggregation. That is I aggregate all the bootstrap models and also is a nice term, because I have a bag of classifiers bag of models. So, bagging stands for

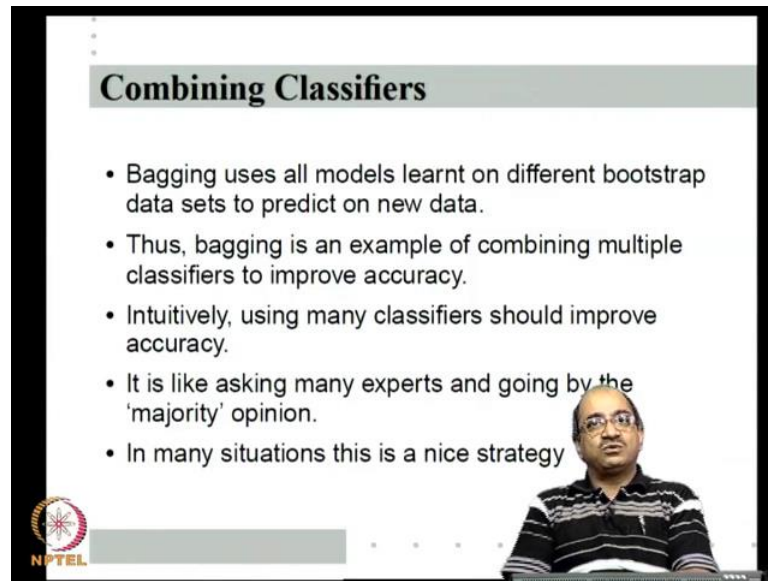
bootstrap aggregation, the idea of bagging is to average over the predictions of all the models. So, given all the bootstrap models that we have learnt \hat{f}_b is equal to $\frac{1}{B}$ to capital B. The bagging model prediction at a new point x is given as $\hat{f}_{\text{bag}}(x)$ is the average of all the $\hat{f}_b(x)$. Of course, the actual theoretical definition of the bagging estimate is not like this. It is the expectation of $\hat{f}_B(x)$, where B is a random bootstrap model that we obtained.

So, which means this expression actually is itself an estimate of the bagging estimate. An estimate derived as a sample mean because I do not know, how to find the distribution of how different bootstrap samples behave? So, for a random bootstrap sample what the model error is what bagging estimate is. So, to get that I just average over many specific actual bootstrap samples that I derived.

So, to be technically correct, I have to say that this is a sample mean estimate of size b of the true bagging estimate, but that is any way just a minor mathematical operation problem. Where essentially, a bagging estimate is to average out all the models you liked. So, often bagging can improve performance of the final model, instead of using any one of them if you use all of them and average out. Of course, I have to ask what averaging means in a classifier context, in a regression context I can immediately see what averaging means in a classifier context, what should I do?

Then maybe you know essentially, if it is a two class problem I can take \hat{f}_b to be plus 1 minus 1. Then I take the sign of this average, but if I am taking sign of course, that one by b makes no difference. So, there is various ways of doing this, but essentially, roughly at this point of same simply say average, for regression models average is what all of us think as average, you just true average because you are anyway predicting a real number. For classification currently average simply means majority, you look at all the predictions and take the class label that it that is output by the majority of a models, that is what averaging would mean for classification.

(Refer Slide Time: 24:54)



Combining Classifiers

- Bagging uses all models learnt on different bootstrap data sets to predict on new data.
- Thus, bagging is an example of combining multiple classifiers to improve accuracy.
- Intuitively, using many classifiers should improve accuracy.
- It is like asking many experts and going by the 'majority' opinion.
- In many situations this is a nice strategy

NPTEL

A small video inset in the bottom right corner of the slide shows a man with glasses and a striped shirt, likely the presenter.

Now, essentially the bagging estimate you wanted to use is one way of using multiple classifiers instead of one. So, far we have just been saying given one training data set how do I will learn one classifier? Now, bagging you just all models learnt from different bootstrap samples to predict on new data. So, we can think of bagging is an example of a method that combines multiple classifiers to improve accuracy. Now, intuitively instead of using one classifier if you use many classifiers, one should improve accuracy. If I this is like I asked many experts for something, if I am preparing for an exam I do not know a question.

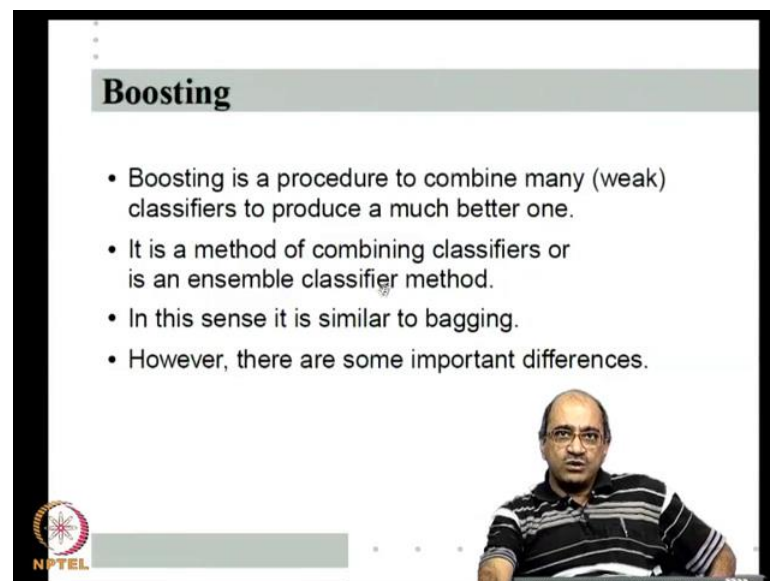
I do not understand something I can go to one of my classmates and ask him to explain it to me well I do not know which one to chose all other might be as bad as I am. So, instead of asking one if I ask 10 of them, may be you know together, they will give me all the things that are there to learn, this is like going to multiple experts. In many situations this is a nice strategy, a simple example which all of you would have seen is the audience poll in [FL]. [FL] there are when you would know the contestant does not know answer to the question, one of the lifelines is the audience poll.

Whereby you ask all the people in the audience of course, nobody in the audience is any great expert to know the answer, each of them may have only 50, 55 percent 60 percent chance of knowing the answer, which is same as the contestant's chance, but if you ask enough of them and go by the majority very often it works, right?

So, even if each individual classifier is not very good, may have only 55, 60 percent chance of being correct. If I have enough of them right like in the [FL] audience poll, it looks like I might be able improve my accuracy much better, then asking anyone of them in the audience poll. Suppose, instead of polling the audience, the contestant is allowed to chose a particular person in the audience and ask.

It would not work any of them is not particularly great is very difficult to choose somebody, whose likely to be correct on this one question, but if you have sufficiently many randomly chosen people there then averaging often gives you the answer right. This is the basic idea of combining classifiers, so that is what we are going to consider next in this course. So, that is more or less the last topic of the course. So, how does one combine classifiers.

(Refer Slide Time: 27:57)



Boosting

- Boosting is a procedure to combine many (weak) classifiers to produce a much better one.
- It is a method of combining classifiers or is an ensemble classifier method.
- In this sense it is similar to bagging.
- However, there are some important differences.

The slide includes an NPTEL logo in the bottom left corner and a video inset in the bottom right corner showing a man in a striped shirt speaking.

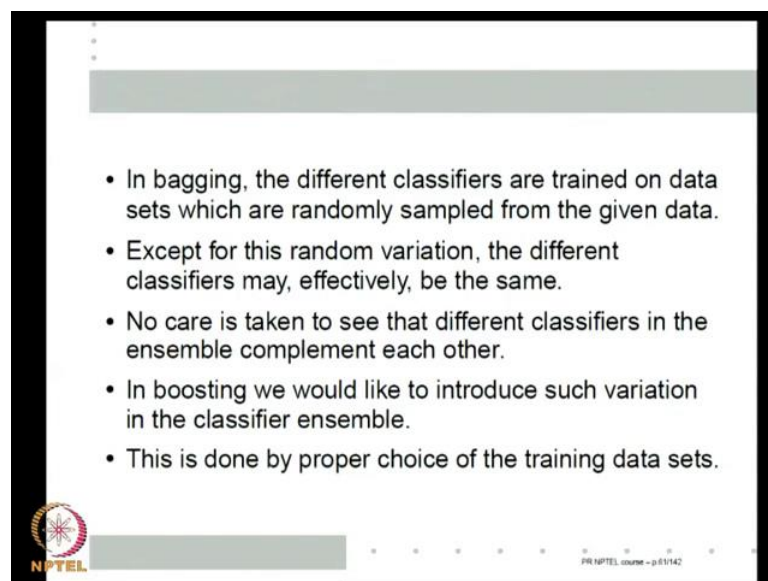
One basic technique for doing this is what is called boosting, boosting is a procedure that many what can be called weak classifiers to produce a better one. Weak in a sense low classifier individually is a great expert at classification. It may not get very high accuracy. Technically weak means, that there expected error is only is just less than 0.5, it may be only 0.5 minus epsilon, right?

So, there may not be many great classifiers, but if you are sufficiently many of them. The idea is that we can combine them to get a classifier with high accuracy. So, this is method of combining classifiers sometimes called an ensemble method, ensemble

classifier method because it uses an ensemble of classifiers. So, classifier combinations classifier combiner ensemble classifier, all these essentially mean the same thing. In all these cases the method used is multiple classifiers, and given any pattern the class label that it outputs is some combination of the class labels given by all the classifiers. Boosting is one such method ah one of the most important methods in classifier combination.

So, in this sense boosting is like bagging of course, bagging is also a classifier combination is simply says take n models and take majority, or average depending on you doing classification regression. So, in that sense boosting is similar to bagging, but there are some very important differences between the basic approach of boosting and what we seen in bagging. We just taking you know b random classifiers b classifiers train and random data sets and taking their average, but boosting is not quite that.

(Refer Slide Time: 29:43)



In bagging the different classifiers are trained on data sets, which are just randomly sampled from given data. So, any variation in different training datasets essentially comes out of this random sampling, there is nothing else there is no purposive variation in different training data set. We are not purposively looking for variation in the training data set just this random variation, which comes because of random sampling is the only variation present in different training data sets, and hence in different classifiers. We have not actually taken any care to see the different classifiers in the ensemble

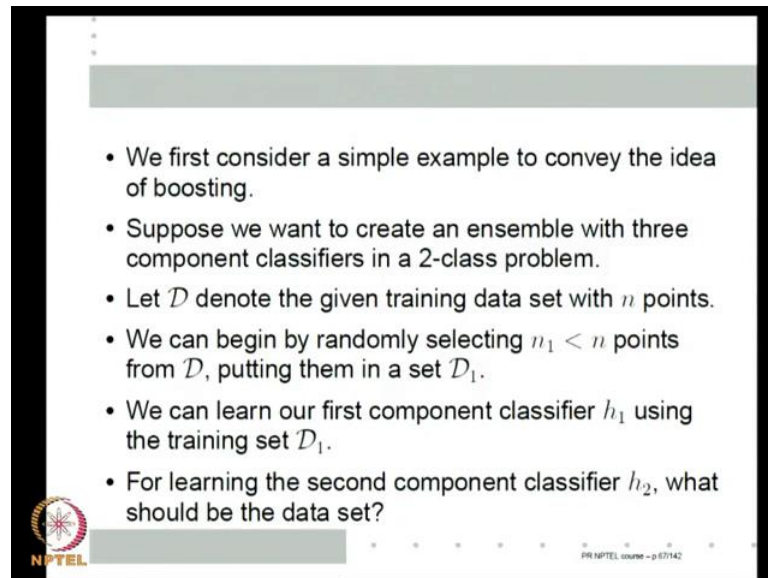
complement each other. See for example, if you are going for a quiz competition you send a team, it is like a classifier ensemble instead of having one person a team.

When you choose a team, you do not choose three people who all of whom performed very well, and some written test at the school conducted that may not be the best thing because all three of them, they have only one strength say sports that would not work. So, if even if somebody got a little less marks in that entrance test conducted, if he has complement test skills. Let us say he can cover music which nobody else knows, he is a very valuable member for the team.

Somehow, if you want a team of classifiers an ensemble of classifiers to do better than individual ones. Somehow each one the different classifiers in the set, have to complement each of the, each one has to cover for the other peoples deficiencies. Only then the team would be more than sum of the individual parts. So, that is what one would normally like right. So, for example, in my [FL] example if I put in the audience peoples who are clones of the same person it is of no use. In practice it works because they come from different works of life, they are randomly chosen from different socio-economic background different educational backgrounds.

So, there is a lot of variability in the individual classifiers somehow, we have to have that variability so that the team will be combined by. So, basically in boosting we like to introduce such variation in the classifier n samples, instead of just generating random data sets, and training classifiers of course, we still generate random data sets, but we want to purposely introduce some variation in the classifier ensembles. Now, how can I introduce variation? Essentially, the only way I can the only difference I in different classifiers is the training data sets on which they are trained. So, I have to somehow create different training data set, which are different which are the right variation.

(Refer Slide Time: 32:34)

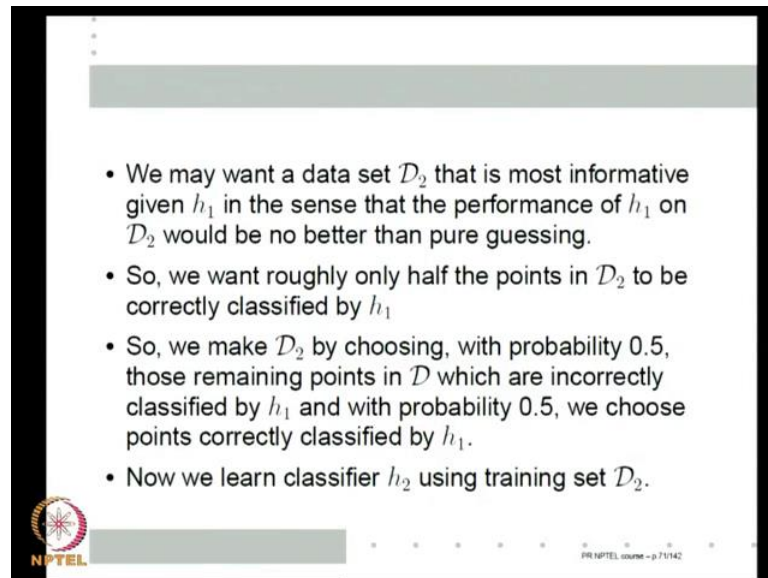


- We first consider a simple example to convey the idea of boosting.
- Suppose we want to create an ensemble with three component classifiers in a 2-class problem.
- Let \mathcal{D} denote the given training data set with n points.
- We can begin by randomly selecting $n_1 < n$ points from \mathcal{D} , putting them in a set \mathcal{D}_1 .
- We can learn our first component classifier h_1 using the training set \mathcal{D}_1 .
- For learning the second component classifier h_2 , what should be the data set?

So, let us look at a very simple intuitive example to convey this idea of boosting. Let us say I am wanting to create a classifier ensemble with three component classifiers, for a two class problem. So, let us say we have a training data set with n points, let us call that script \mathcal{D} . So, I have to now generate three separate training data sets to train my three component classifiers in the ensemble. What I do first? One is no problem, I have to just generate some training data set. Let us say I randomly select some n_1 points, n_1 less than n from \mathcal{D} , I put them in a set called \mathcal{D}_1 and then I train my first component classifier and \mathcal{D}_1 this is fine.

So, I just of course, this is not completely fine I have to know how large or small n_1 should be compared to n that is very much problem specific, and this is only any one example to give you some idea of you know, boosting trying to do some variation. So, let us stick to that let us say I chose some n_1 data points. So, we learn our first component classifiers, classifier call it h_1 using training data set \mathcal{D}_1 . Now, our idea is for learning the second component classifier h_2 , I do not want some other random n_2 points from \mathcal{D}_1 . Then I cannot really intuited that h_2 has right variation compared to h_1 .

(Refer Slide Time: 34:14)



The slide contains a list of four bullet points explaining the construction of a data set \mathcal{D}_2 for training a classifier h_2 . The points are:

- We may want a data set \mathcal{D}_2 that is most informative given h_1 in the sense that the performance of h_1 on \mathcal{D}_2 would be no better than pure guessing.
- So, we want roughly only half the points in \mathcal{D}_2 to be correctly classified by h_1 .
- So, we make \mathcal{D}_2 by choosing, with probability 0.5, those remaining points in \mathcal{D} which are incorrectly classified by h_1 and with probability 0.5, we choose points correctly classified by h_1 .
- Now we learn classifier h_2 using training set \mathcal{D}_2 .

The slide also features the NPTEL logo in the bottom left corner and the text 'PRE NPTEL course - p 75142' in the bottom right corner.

So, I want to ask, what is a good data set to train h_2 , one way of saying this is the data set say call it \mathcal{D}_2 the use on which I want to train h_2 , on which I want to learn h_2 should be such that, given the classifier h_1 that data set is most informative for classification. Most informative means; that data set is completely surprising for the classifier h_1 . If the classifier h_1 can easily explain away the data set then the training another classifier on that set makes no sense, because h_1 is doing well on this.

So, is not that on the data set h_1 should very well or very poorly, but that data set should be somehow surprising to h_1 . What do I mean by this surprising to h_1 ? The performance of the h_1 on \mathcal{D}_2 should be no better than pure guessing. If suppose, h_2 gets 0 character on it that means, I know all of them are very hard problems that may be simply complements of the (()) such things. The other hand h_1 does very well also is useless, but essentially is most informative, when a \mathcal{D}_2 the performance of h_1 on \mathcal{D}_2 is essentially, like pure guessing. Meaning roughly about half the points in the \mathcal{D}_2 are correctly classified by h_1 and half the points are incorrectly classified by h_1 .

So, h_1 does not you know making errors, some it gets right some it gets wrong, but it is completely surprising to it because it looks like, the particular characteristic of data h_1 length does not apply to \mathcal{D}_2 because on that characteristic, how the points are classified correctly? Half the points are classified incorrectly. So, in that sense \mathcal{D}_2 has the most surprise for h_1 . Now, can I generate this yeah I can generate this what do, I make \mathcal{D}_2 by

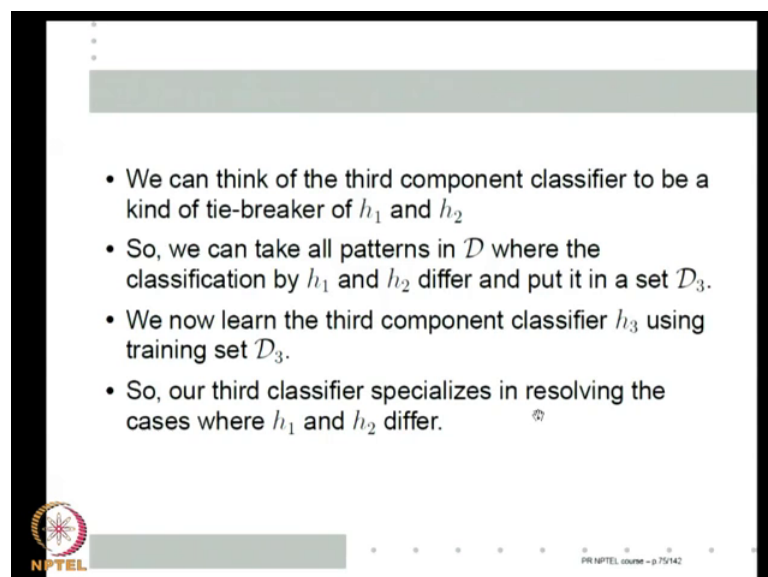
choosing from D_1 with probability half the remaining points in D_1 , which are correctly classified by h_1 probability half points incorrectly classified. What does that mean?

I first take away all the n_1 points from D . Now, in the remaining points I start from the beginning I keep looking, I first toss a coin I first generate a random number, the random number comes less than 0.5. That means, the coin falls heads then I am now looking for something that is incorrectly classified by h_1 . So, I keep going down the points in the the points in so to say $d - D_1$. Till I find a point which is incorrectly classified by h_1 put that in D_2 .

Now, once again toss the coin suppose this time coin falls tails that means, this time my random number goes more than 0.5. Then I now looking for a point in this set in the set $D - D_1$, which is incorrectly classified by which is correctly classified by h_1 , and so on. So, I keep choosing points removing points from $D - D_2$ like this. So, that with probability 0.5 I am choosing points, which are correctly classified by h_1 and with probability 0.5, I am choosing points which are incorrectly classified by h_1 .

So, that D_2 will roughly about 50 percent of the points correctly classified by h_1 , 50 percent of the points incorrectly classified by h_1 . So, this gives sufficient variation in D_2 . Now, we learn a classifier h_2 using D_2 right next what? What do I do with, how do I. Now, I know I am using only 3, if I am using many I can keep doing this kind of thing, but if I am because I know 3.

(Refer Slide Time: 37:42)



The slide contains a list of four bullet points explaining the role of a third classifier h_3 as a tie-breaker for h_1 and h_2 . The text is as follows:

- We can think of the third component classifier to be a kind of tie-breaker of h_1 and h_2
- So, we can take all patterns in \mathcal{D} where the classification by h_1 and h_2 differ and put it in a set \mathcal{D}_3 .
- We now learn the third component classifier h_3 using training set \mathcal{D}_3 .
- So, our third classifier specializes in resolving the cases where h_1 and h_2 differ.

The slide also features the NPTEL logo in the bottom left corner and the text 'PR NPTEL course - p 75142' in the bottom right corner.

h_3 is like a tie breaker now and I try to get h_1, h_2 learn as much variation as possible, but h_3 should be a tie breaker. So, if h_3 to be a tie break then what we can do is take the original data set and pick up all the points in which h_1, h_2 differ then put that in D_3 and the h_3 and D_3 . So, what did we do we made our third classifier specialize in resolving the cases where h_1, h_2 differ right. So, because we using only three classifiers we put sufficient variation for h_1, h_2 . So, that h_1, h_2 cover for most of the space and then we made the third classifier specialize in trying to classify, those cases where h_1, h_2 differ, right?

(Refer Slide Time: 38:35)

- We use this classifier ensemble as follows.
- Given a new pattern X , if $h_1(X) = h_2(X)$ then that is what we output; otherwise we output $h_3(X)$.
- Our combination of classifiers is not a simple majority scheme; the way we use the classifier combination is related to the way we learned the component classifiers.
- This is essentially the main idea of boosting.

NPTEL

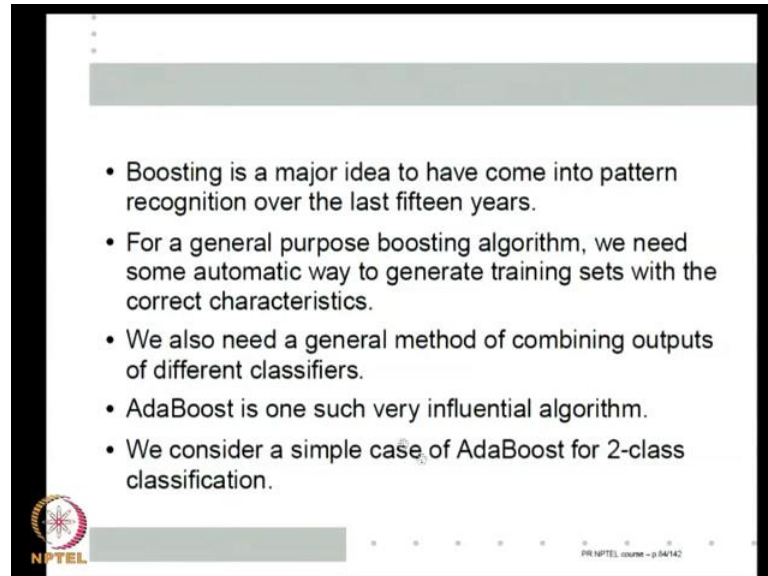
PR NPTEL course - p.79142

So, now how do I use this ensemble because I know how I train them, this is how I am going to use my ensemble. If you give me a pattern X I find $h_1(x)$ and $h_2(x)$ if $h_1(x)$ is equal to $h_2(x)$ that is what is the class I will get. On the other hand if $h_1(x)$ is not equal to $h_2(x)$ then what our $h_3(x)$ is what output right. Of course, this is actually a majority right if h_1 is equal to h_2 $h_1(x)$ is equal to $h_2(x)$ no matter what $h_3(x)$ is that is the majority, if $h_1(x)$ is not equal to $h_2(x)$. Once again, whatever h_3 because a two class classifier, what our h_3 says is the majority, but it is not just a simple averaging.

The idea is that because of the way I am trying to use this a classifier combination, I have trained them differently that is the basic idea, the basic idea is I put some variability in the training data. So, that this kind of averaging pairs well this kind of averaging gives me a better classifier, this is the basic idea of boosting that instead of just randomly

generating training data sets and just averaging, try and generate more informative training data sets for successive classifiers. So, that is the basis intent of boosting.

(Refer Slide Time: 40:01)



Boosting can be seen to be one of the major ideas who have come into pattern recognition last 15 years. As a matter of fact over the last 20 years, the two fundamental advances that this field has seen, one is the support vector machines and Kernel based methods. Other is this idea of boosting classifier combinations right these two are really fundamental advances to in theory and practice of pattern recognition. So, in such boosting is a very important technique.

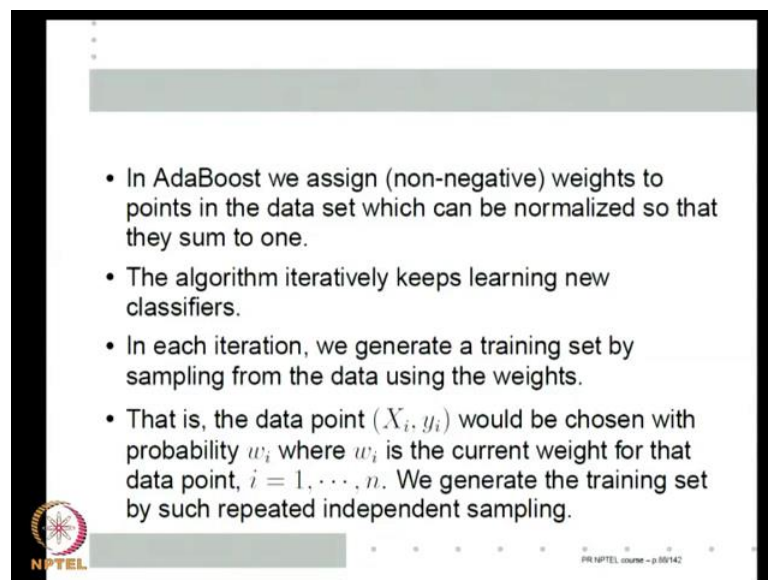
For a general purpose boosting algorithm we need some automatic way of generating training sets. This what I call more informative training sets. I have to have some way of generating see in that method I did, I do not know how much I want to chose. So, if I have to chose in the remaining n minus out of the remaining n minus $n - 1$ samples only for $D = 2$. I may have sufficient may not have sufficient and now, after that I am may not have any more samples left for training more.

So, that specialized thing is only an example to give me an idea for what it mean by choosing training data sets. So, that you know different classifiers are trained on nicely informative data sets, but we need some automatic way of generating training sets, from the given original data right. We also need some general method of combining outputs of

different classifiers, what averaging, what kind of majorities. So, any boosting algorithm has to do these two.

What we are going to do is to look at one such algorithm, which is called adaboost the influential one of the most important and most popular boosting algorithms. Of course, adaboost can be used for classification, regression, there are the different variations of adaboost. This is a first level course, we have to say and given that we are almost at the end of the course. We will consider only one simple case of adaboost; that is for two class classification, but it gives you the full idea of how that kind of general purpose boosting is done.

(Refer Slide Time: 42:19)



- In AdaBoost we assign (non-negative) weights to points in the data set which can be normalized so that they sum to one.
- The algorithm iteratively keeps learning new classifiers.
- In each iteration, we generate a training set by sampling from the data using the weights.
- That is, the data point (X_i, y_i) would be chosen with probability w_i where w_i is the current weight for that data point, $i = 1, \dots, n$. We generate the training set by such repeated independent sampling.

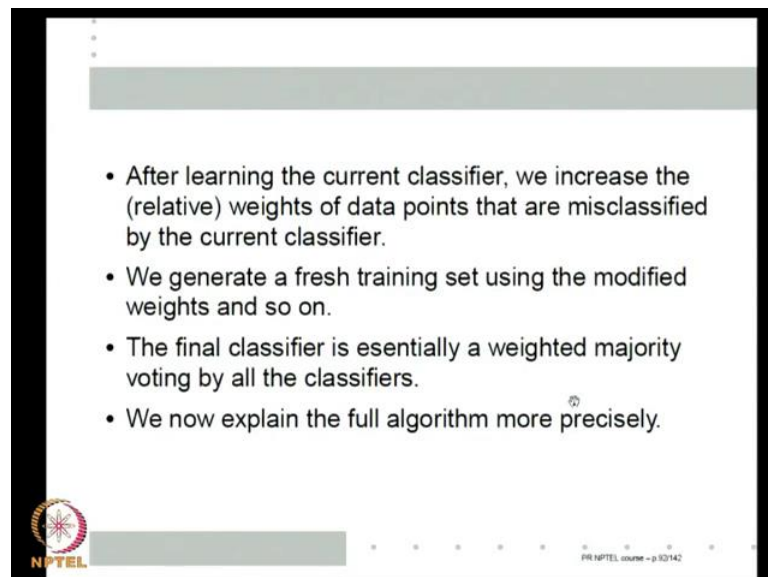
The basic idea of adaboost is following, in adaboost for the so we have the initial data set x_1, x_2 to x_n . So, for each X_i we assign a weight may be weight called w . So, we assign non negative weights to the points in the data set, and the weights are normalized. So, that they sum to one so you can think of the weight as a probability. Then the idea is that I iteratively the algorithm in each iteration learns a new classifier, just like the boost (()) we have learn 3. So, 3 can learn as many as you want.

At each iteration you have to learn new classier, to learn new classier I have to have a training data set. So, in each iteration, I generate a training set by sampling from the data using these weights. I use these weights as a probability distribution and generate i i d samples out of the distribution. That means, a particular X_i is chosen is a probability of

proportion to the weight, and a keep making many i i d realization of this to generate a training data set.

So, what I mean is a data point X_i would be chosen with probability w_i where w_i is the current weight for that point. Of course, as we said we always keep w 's normalize so that the sum to 1. So, we generate the training data set by I first you know chose a get a random number, distributed as w_1, w, w_n whichever one it falls in I pick that X_i . Once again do you know repeated independent sampling like this. So, we generate a training data set by repeating independent data sampling using the weights.

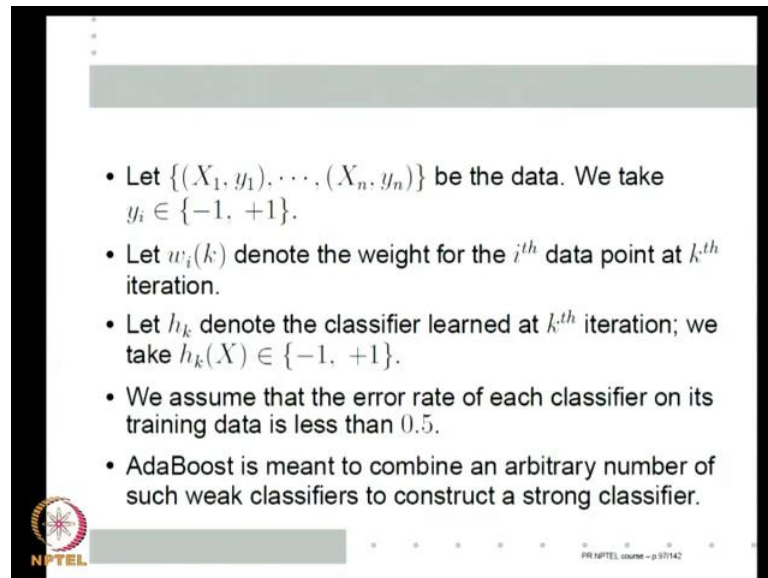
(Refer Slide Time: 43:55)



Then after learning the current classifier, what do I do we change the weights. We essentially increase the weights for the points that are misclassified by the current classifier whatever, points you have my data are misclassified by the current classifier. I increase the weights of those points. Then using the modified weights, we generate a first training data set and once, again learn a new classifier and so on.

And the final classifier we essentially, use a weighted majority voting where the weights are in some sense proportional to the accuracy achieved by each classifier. So, classifier with certainly higher accuracy would get higher weightage for their, but of course, is not directly accuracy we have to know we have to use the right scale to give the weights, but in a adaboost uses some such things. So, it gives a weighted majority for all the classifiers. So, this is the overall intuitive idea so, let us go for this algorithm.

(Refer Slide Time: 44:55)



The slide contains a list of five bullet points defining notation and assumptions for AdaBoost. The NPTEL logo is in the bottom left, and the slide number 'PRE NPTEL course - p.97142' is in the bottom right.

- Let $\{(X_1, y_1), \dots, (X_n, y_n)\}$ be the data. We take $y_i \in \{-1, +1\}$.
- Let $w_i(k)$ denote the weight for the i^{th} data point at k^{th} iteration.
- Let h_k denote the classifier learned at k^{th} iteration; we take $h_k(X) \in \{-1, +1\}$.
- We assume that the error rate of each classifier on its training data is less than 0.5.
- AdaBoost is meant to combine an arbitrary number of such weak classifiers to construct a strong classifier.

Let us review our notation we are considering a two class case, the data is x_1, y_1, x_n, y_n and data points and we take two class, we take y_i to be plus 1 minus 1. With $w_i(k)$ of k denotes the weight for the i^{th} data point that is X_i, y_i at k^{th} iteration we are doing learning classifiers iteratively. So, at the k^{th} iteration the weight assigned to i^{th} data point is denoted by $w_i(k)$ we denote successive classifiers h_k . So, h_k denote the classifier learnt at the k^{th} iteration and is a classifier. So, h_k of h_k for x for any x will be either minus 1 or plus 1.

We assume that the error rate of each classifier on its training data is less than 0.5 otherwise of course, the method would not work, but I will point it out in the exact algorithm, where it comes. Essentially, the idea is that I do not need my individual classifiers to be too big, I can chose them to be simple ones. So, if for example, if I chose those classifiers to be of low complex classifiers, small size neural networks or you know a simple decision tree or you know simple linear classifier.

Then because the model complexity is small even the small data item, within that class I can learn a fairly good classifier that classifier may not have great accuracy, but basically by boosting even though those individual classifiers, may not have very great accuracy there error rate is only may be 0.5 minus epsilon by having enough of them, I can do very well and because I do not want individual classifiers to be classifiers to be very

highly accurate, I do not have to go for very high model complex this is how the basic idea of boosting based.

(Refer Slide Time: 46:56)

AdaBoost Algorithm

1. Initialize: $w_i(1) = \frac{1}{n}, \forall i.$
2. For $m=1$ to M do
 - a. Generate a training set by sampling with $\{w_i(m)\}.$
 - b. Learn classifier h_m using this training set.
 - c. Let

$$\xi_m = \sum_{i=1}^n w_i(m) I_{[y_i \neq h_m(X_i)]}$$
 where I_A is indicator of A . We assume $(\xi_m < 0.5).$
 - d. Set $\alpha_m = \ln \left(\frac{1-\xi_m}{\xi_m} \right).$ (We have $\alpha_m > 0$).

So, let us consider the algorithm. So, to start with I put weights as 1 by n beginning there is nothing to chose one data point or other. So, all weights are equal and as you can see the beginning the weights summed to 1. Then I is an iterative procedure, let us say capital m is number of iterations that is capital M is the total number of classifiers, I am going to learn right little m is my iteration count. What do I do, I first generate a training data set by sampling with these weights, using my current iteration my current weights w I f m, I generate a training set.

Then I learn a classifier using this training set the classifier can be anything. You know as a matter of fact I can definite iterations I am, I am even allowed to learn different classifiers even though in a program sense, I have to decide and which iteration, which classifier I learn, but as far as the algorithm is concerned different classifiers need not have to be the same type either that is why on all even number iterations. I learn using linear lest squares on all odd number iterations, I may learn an (()) or a I learn neural network.

So, where essentially I learn some classifier h_m using this training data set then I calculate ξ_m , which is a kind of weighted error of this classifier. So, $\xi_m = \sum_{i=1}^n w_i(m) I_{[y_i \neq h_m(X_i)]}$ is the data set. So, if y_i is not equal to $h_m(X_i)$ the h the classifier h_m has made a error on X_i y i.

That error is weighted by the weight w_i^m mind you $X_i y_i$ may or may not be in the actual training data set using which I learnt h_m , but I do not care about that just after learning h_m for each I . If y_i is not equal to $h_m(X_i)$ that if i made an error then I add w_i^m for that data error.

So, the cost of error is not errors are not simply counted there weighted by the current weights that is the error rate right. The idea is if a particular I has high weight it is likely to be represented more than once in the training data set of h_m , and h_m would have certainly taken care of that. So, that is the whole idea. So, we assume that this j^m is less than 0.5 in practice what it means is after I generate a data set and learn a classifier, if my j^m does not happens to be less than 0.5. I just throw it away run the iteration again. I once again get another data set by randomly sampling with the current weights only and learn again very often many of these classifiers are reasonably good.

So, even at high data sets they can give greater than 50 percent accuracy. So, we mostly always get j^m less than 0.5. Then we calculate α_m which is log natural logarithm of $1 - x_i^m$ by x_i^m . Because we assume x_i^m to be less than 0.5, $1 - x_i^m$ is greater than x_i^m , which means, α_m is always positive. So, we calculate these weighted error, this is what we call error rate of n th classifier, and then we calculate this α_m . Now, that we have learnt the classifier, the next step is I have to now find the next set of weights.

(Refer Slide Time: 50:10)

Algorithm contd.


e. Update the weights by

$$w_i'(m+1) = w_i(m) \exp(\alpha_m I_{[y_i \neq h_m(X_i)]})$$

$$w_i(m+1) = \frac{w_i'(m+1)}{\sum_i w_i'(m+1)}$$

3. Output the final classifier:

$$h(X) = \text{sgn} \left(\sum_{m=1}^M \alpha_m h_m(X) \right)$$


© NPTEL course – p 105/142

This is my weight update. So, first give $w_i(m)$ calculate $w_i'(m+1)$ the whole idea is $w_i'(m+1)$ is not normalized. So, to get $w_i(m+1)$ I normalize w_i' . So, first $w_i'(m+1)$ is calculated by $w_i(m)$ into exponential α_m into $I(y_i \neq h_m(X_i))$ by the I indicate this. But I this I is an indicator function I of a is 1 if the event a is true is 0 if a false. So, $I(y_i \neq h_m(X_i))$ is 1 if y_i is not equal to $h_m(X_i)$ 0 to I is not equal to $h_m(X_i)$. So, that I how $w_i'(m+1)$ is calculated and then I normalize w_i' to get weights of the next iteration.

So, this is what I keep doing for m is equal to 1 to n right a, b, c, d, e once I finish it my final classifier, mind that each of h_m 's are plus 1 minus 1 right if sum is not there as we already seen if α_m is not there, this is simply a majority vote. So, I am weight weighing the $(h_m(X_i))$ by α_m . And α_m is some measure of the accuracy so, but anyway we come back to this later. So, this is the final classifier. So, this is the adaboost algorithm.

So, I generate training set by sampling with the weights learn a classifier then calculate the error of the classifier, which is equal to weights of all the data that are misclassified with the classifier. Then I calculate α_m which is $\frac{1}{n} \sum_{i=1}^n w_i(m) I(y_i \neq h_m(X_i))$ by α_m we assuming α_m less than 0.5. So, α_m is always positive the I update weights like this that is it, I do it iteratively for m and then use all the m classifiers.

(Refer Slide Time: 52:03)

- M is a parameter. Due to the sampling with weights, we can continue the procedure for arbitrary number of iterations.
- If, for an i , $h_m(X_i) = y_i$, then $w_i'(m+1) = w_i(m)$.
- If, $h_m(X_i) \neq y_i$, then $w_i'(m+1) > w_i(m)$.
- Hence, if $h_m(X_i) \neq y_i$, then $w_i(m+1) > w_i(m)$.
- If the current classifier misclassifies a pattern, its weight for the next iteration is increased.

PR NPTEL course - p.110142

Let us look at some of the characteristics of this algorithm m is a parameter first. So, I can choose any m because I keep sampling with weights, I can continue this process for arbitrary number of iteration. So, there is no limit except computational resources and things like that and how many classifiers, I could use to make combination? So, in that sense it satisfy my first requirement for a general purpose boosting algorithm, it should be able to keep automatically generating as many training data sets as they want.

Now, for a particular i if $h_m(X_i)$ is equal to y_i then this indicator is false, then w_i^{m+1} is equal to w_i^m because the exponential 0 is 1. On the other hand if $h_m(X_i)$ is not equal to y_i then this is one this exponential α_m , α_m is positive. So, exponential α_m is greater than 1. So, w_i^{m+1} is greater than w_i^m if y_i not equal to $h_m(X_i)$ meaning the i -th sample is misclassified. So, if i -th sample is correctly classified $h_m(X_i)$ is equal to y_i .

Then w_i^{m+1} is equal to w_i^m , on the other hand if $h_m(X_i)$ is not equal to y_i then w_i^{m+1} is greater than w_i^m and because is normalization, which does not change this relative. If i -th sample is misclassified then w_i^{m+1} is greater than w_i^m . So, as I told you beginning intuitively, our equations ensure that is the present classifier misclassifies the pattern. Then the weight of that pattern for the next iteration is increased.

(Refer Slide Time: 53:46)

• But if we keep arbitrarily increasing weights of misclassified patterns, then subsequent training sets will be predominantly only those patterns.

• This does not give proper variability in training sets.

• We want only about half the samples to be the ones misclassified by current classifier.

• The weight update scheme of AdaBoost ensures

$$\sum_{i: h_m(X_i) \neq y_i} w_i(m+1) = \sum_{i: h_m(X_i) = y_i} w_i(m+1) = \frac{1}{2}$$

NPTEL PR NPTEL course - p 114142

Of course, this by itself is not enough if you keep arbitrarily increasing weights of misclassified patterns, then the subsequent training sets can be pre-dominantly only misclassified pattern. As we said that is not enough that is not the right variation right, to have right variability, we want the pattern sets to contain about half the samples misclassified by the current classifier half the samples, correctly classifiable by the current classifier.

So, I cannot just arbitrarily increase weights, but this algorithm is a very nice weight increase formula, which ensures at every iteration that after update the weights, if I add a weights at the next iteration of all i , which are misclassified by h_m that will be same as the weights of all i , which are correctly classified and so, both of them be equal to half. So, half the weight is for patterns correctly classified by h_m half the weight at the m plus 1 iteration is for patterns incorrectly classified by h_m .

So, when I am sampling my data, my training data set sampling from training data at the m plus iteration. Likely that half the patterns will be once, which are currently classified of h_m of the pattern, incorrectly classified by h_m . So, let us just prove this, this is a very surprising result, but is worth proving and then we will we look at more details of this thing again.

(Refer Slide Time: 55:26)

Recall the weight update formulas:

$$\xi_m = \sum_{i=1}^n w_i(m) I_{[y_i \neq h_m(X_i)]}$$

$$\alpha_m = \ln \left(\frac{1 - \xi_m}{\xi_m} \right)$$

$$w_i'(m+1) = w_i(m) \exp(\alpha_m I_{[y_i \neq h_m(X_i)]})$$

$$w_i(m+1) = \frac{w_i'(m+1)}{\sum_i w_i'(m+1)}$$

NPTEL logo and footer text: PR NPTEL course - p.115/142

For the proof let us remember weight update formula I once again wrote here, j m is given by this alpha m is given by this and this is my weight update formula we will come

back to this again and again. So, by definition see j_m is this so, I can actually convert this onto index. So, sum over only those i for which y_i is not equal to $h_m(X_i)$.

(Refer Slide Time: 55:51)

• By definition, we have

$$\xi_m = \sum_{i: h_m(X_i) \neq y_i} w_i(m)$$

• Since the weights are always normalized, we have

$$\sum_{i: h_m(X_i) = y_i} w_i(m) = 1 - \xi_m$$

• Also, by definition, $\exp(\alpha_m) = \frac{1 - \xi_m}{\xi_m}$

NPTEL logo and footer: PR NPTEL course - p 115/142

So, I can write j_m is sum over i th that $h_m(X_i)$ is not equal to y_i $w_i(m)$. Now, since $w_i(m)$ are sum over all i of $w_i(m)$ is always 1, if instead of summing over i such that $h_m(X_i)$ is not equal to y_i . If i sum over $h_m(X_i)$ equal to y_i , I get $1 - j_m$ also α_m is \ln of $1 - j_m$ minus j_m by j_m . So, exponential α_m is $1 - j_m$.

(Refer Slide Time: 56:19)

• Hence, if $h_m(X_i) \neq y_i$, then

$$w'_i(m+1) = w_i(m) \frac{1 - \xi_m}{\xi_m}$$

• On the other hand, if $h_m(X_i) = y_i$, then


$$w'_i(m+1) = w_i(m)$$

NPTEL logo and footer: PR NPTEL course - p 120/142

So, what does this mean if $h_m(X_i)$ is not equal to y_i then $w_i^{\text{prime } m+1}$ is w_i^m into $1 - \xi_m$ by ξ_m if $h_m(X_i)$ equal to y_i then $w_i^{\text{prime } m+1}$ is w_i^m .

(Refer Slide Time: 56:33)

Now we have

$$\begin{aligned} \sum_i w_i^{\text{prime } m+1} &= \sum_{h_m(X_i) \neq y_i} w_i^{\text{prime } m+1} + \sum_{h_m(X_i) = y_i} w_i^{\text{prime } m+1} \\ &= \sum_{h_m(X_i) \neq y_i} w_i^m \frac{1 - \xi_m}{\xi_m} + \sum_{h_m(X_i) = y_i} w_i^m \\ &= \xi_m \frac{1 - \xi_m}{\xi_m} + (1 - \xi_m) \\ &= 2(1 - \xi_m) \quad \square \end{aligned}$$



PR NPTEL course - 0125142

Now, we can use this as follows suppose, w_i^m all the $w_i^{\text{prime } m}$ then I can sum over all i 's such that $h_m(X_i)$ is not equal to y_i , i such that $h_m(X_i)$ is equal to y_i . So, this sum over all i of $w_i^{\text{prime } m}$ split into two sums, those i 's such that $h_m(X_i)$ not equal to y_i and the remaining i . Now, for those i such that $h_m(X_i)$ not equal to y_i , I know what the value of $w_i^{\text{prime } m+1}$ is w_i^m not no prime here w_i^m into $1 - \xi_m$ by ξ_m .

For the once where h_m is correctly classified $w_i^{\text{prime } m}$ is simply w_i^m . Now, this is not dependent on i and sum of w_i^m such that $h_m(X_i)$ is not equal to y_i . We just now calculated, that is ξ_m , ξ_m by definition is sum of w_i^m such that $h_m(X_i)$ not equal to y_i . This we already seen is equal to $1 - \xi_m$, right? So, which means, the sum over i of $w_i^{\text{prime } m+1}$ is always $2(1 - \xi_m)$.

(Refer Slide Time: 57:37)

Using this, we get

$$\begin{aligned}\sum_{h_m(X_i) \neq y_i} w_i(m+1) &= \sum_{h_m(X_i) \neq y_i} \frac{w_i(m) \frac{1-\xi_m}{\xi_m}}{2(1-\xi_m)} \\ &= \frac{1}{2\xi_m} \sum_{h_m(X_i) \neq y_i} w_i(m) \\ &= \frac{1}{2\xi_m} \xi_m \\ &= \frac{1}{2}\end{aligned}$$


PR NPTEL course - p 130142


Given this, let us say I want sum of $w_i(m+1)$ summed over all, i which are misclassified by h_m right that is for the once, which classifier we just now seen $w_i(m+1)$ is $w_i(m)$ into m minus x_i by x_i . So, $w_i(m+1)$ is $w_i(m)$ plus 1 by the normalizing constant, the normalizing constant we have just now seen is $2(1-\xi_m)$. So, this $1-\xi_m$ will cancel with this $1-\xi_m$ this gives me 1 by $2\xi_m$ into sum over i such that $x_i \neq y_i$ of $w_i(m)$ right. This we already know ξ_m this is half.

(Refer Slide Time: 58:23)

- Thus we have shown that

$$\sum_{i: h_m(X_i) \neq y_i} w_i(m+1) = \frac{1}{2}$$

- The weight update is such that at the next iteration, half the total weight is for patterns misclassified by the current classifier and the remaining half is for patterns correctly classified by the current classifier.



PR NPTEL course - p 130142

So, we have shown that sum over i such that X_i, y_i is misclassified by h_m , the weights at the next iteration is half. So, the weight update is such that at the next iteration half the weight of the pattern, misclassified by the current classifier and the remaining half of half is for patterns correctly classified by the current classifier. So, this is very interesting weight update in the next class, we will look at few more interesting facts about this algorithm, which is what it makes it a very great boosting algorithm. Then show that we can actually, look at it as a very interesting risk minimization under a special kind of loss function. So, next class we finish we will look at adaboost algorithm again, and give you an overview of a view of adaboost algorithm as a risk minimize.

Thank you.