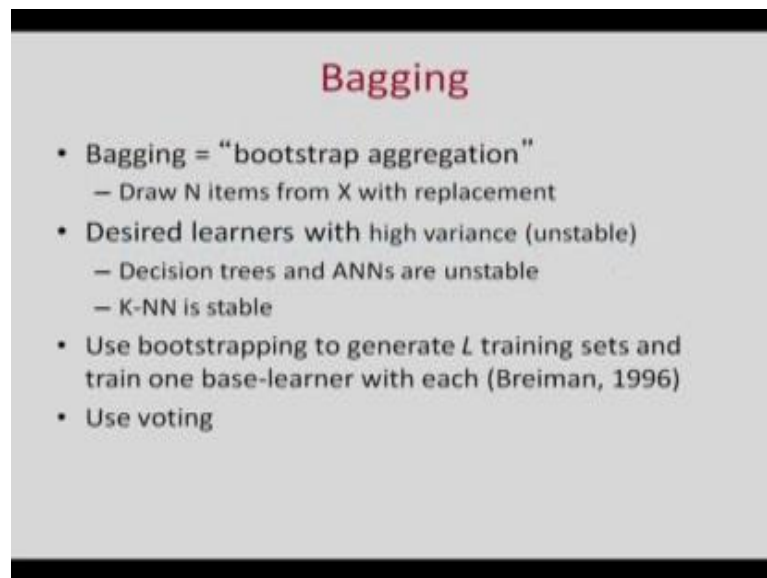


**Introduction to Machine Learning**  
**Prof. Sudeshna Sarkar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Module - 8**  
**Lecture - 36**  
**Bagging and Boosting**

Good morning, today we will have the second part of the lecture on Ensemble Learning. In the last class, we gave a big introduction to the different ensemble learning methods. Today, I will talk about two specific methods, bagging and boosting which are methods for ensemble learning.

(Refer Slide Time: 00:41)

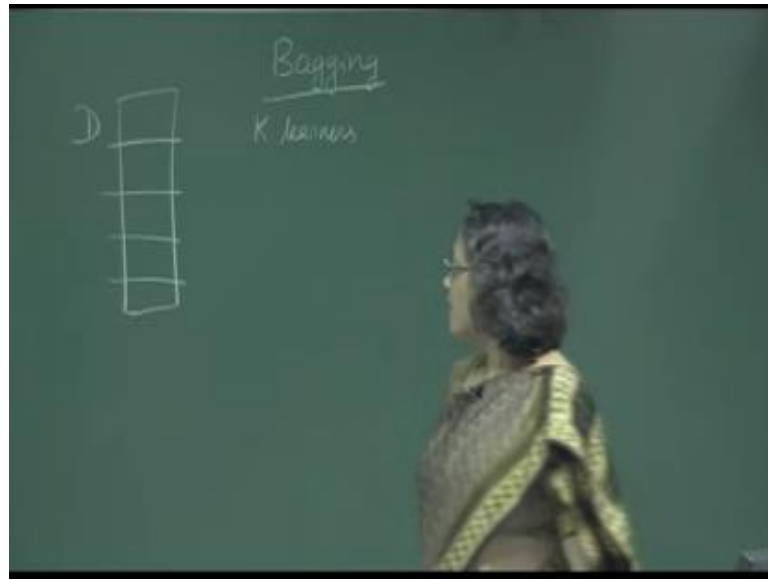


**Bagging**

- Bagging = “bootstrap aggregation”
  - Draw  $N$  items from  $X$  with replacement
- Desired learners with high variance (unstable)
  - Decision trees and ANNs are unstable
  - K-NN is stable
- Use bootstrapping to generate  $L$  training sets and train one base-learner with each (Breiman, 1996)
- Use voting

Bagging is a method for ensemble learning bagging stands for bootstrap aggregation.

(Refer Slide Time: 00:49)

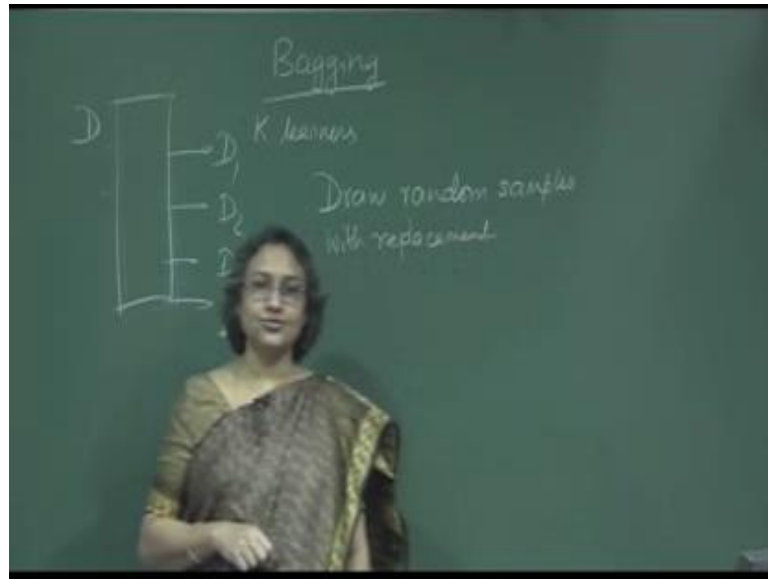


So, we saw that in order to have in order to use an ensemble of learners and be able to combine the learners and get better accuracy, what is needed is a set of learners who which make independent errors you want to set of learners which make independent errors. And we discussed, in the last class, in order to force the learners to make independent errors, you can make them use different samples - data samples, you can make them use different parameters of the algorithm, or you can make them use different algorithms and the output can be combined by some form of voting.

Now, in bootstrap aggregation, what is done is that; first of all the samples are generated such that the samples are different from each other. However, suppose, you have a training set  $S$  or let us call it  $D$ , and if you are going to have  $n$  learners or let us say  $k$  learners, you could split the training set into  $k$  partitions and use one for each learner.

However, often it is the case that the size of the training set is not large, so if you make it smaller by splitting it into  $k$  different sets, the individual training sets will be small. And as we have observed while talking about the foundations of machine learning is that if you use a learner on a small training example, the learner can over fit, it can have high variance and it will not generalize well. So, we want to be able to use you know, we do not we will not be able to afford to have fully independent data, but we can sample randomly from the data, so that you know there may be some overlaps. And we do that by a method of sampling instead of dividing it into  $k$  disjoint groups.

(Refer Slide Time: 03:19)



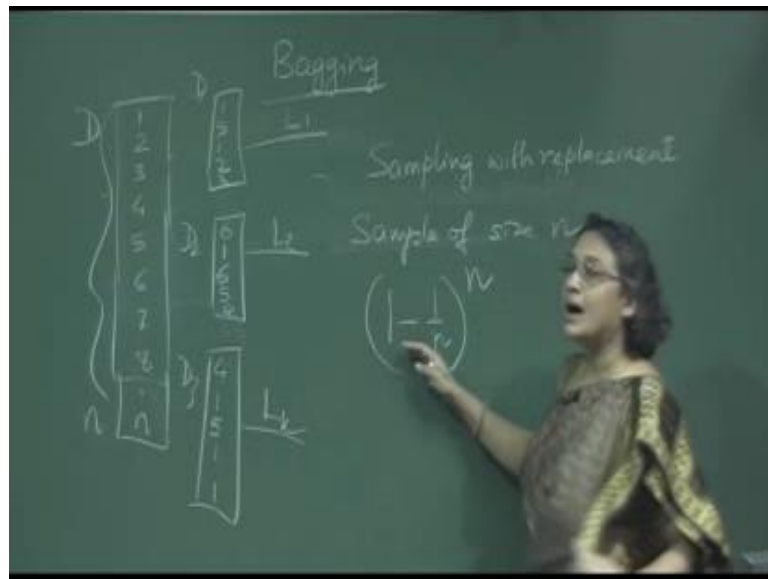
We treat this as the pool; and from this pool, we sample  $D_1, D_2, D_3, D_k$ , so these are the different data subset which is sampled from the original data. They may have some in common, but they also will have different values. Now, as we have discussed that what we do is that to get  $D_i$  draw random samples with replacement. For example, if we desire that each of these  $D_i$ 's, we will have  $m$  examples we randomly draw  $m$  samples from  $D$  with replacement that is we can have the same instance repeated several times and some instances may not appear in a particular  $D_i$ , and so these  $D_i$ 's can be different.

We have discussed in the last class that in order that ensembles will work we desire that these learners may have high variance. And by combining these  $k$  learners, we can reduce the variance you know in certain cases by  $1$  by  $k$ . So, we can start with learners with high variance. And we also discussed that in order that the different learners produce different hypothesis, we want a learners to be unstable, stable learners you know the respective of what data there you get, the output will be similar that is they have low variance. Learners with high variance are unstable.

Among the algorithms that we have studied decision tree, then neural network they are unstable algorithms; decision tree neural network are unstable algorithms given different data, they may give different models. Whereas algorithms like  $k$  nearest neighbor is more stable. So, when we want to use ensembles or specifically bagging, we wish to use unstable learner. And we use bootstrapping to generate the training sets; we generate

these  $k$  training sets. And we train.

(Refer Slide Time: 06:05)



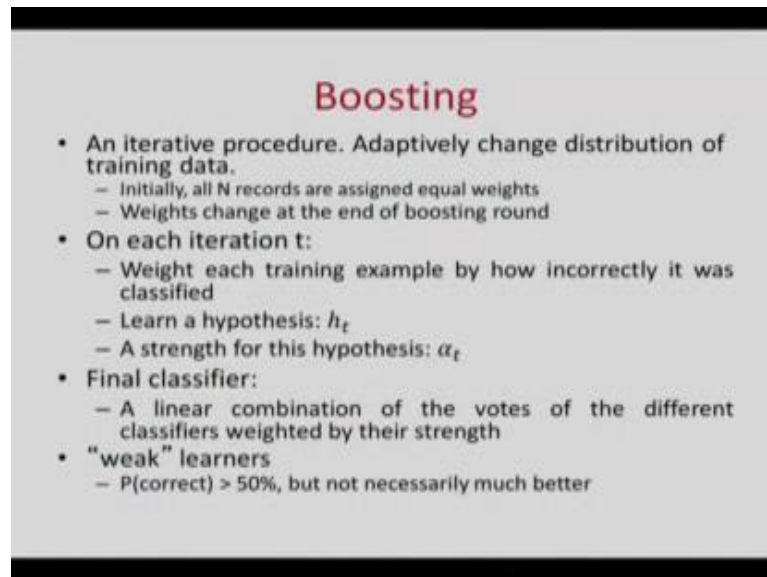
So, once we get these training sets so suppose in this  $D$ , we have the instances 1, 2, 3, 4, 5, 6, 7, 8. And we sample and we get  $D_1$  and  $D_1$  may have 1, 5, 1, 2, 3 and  $D_2$  may have 6, 1, 6, 5, 4;  $D_3$  may have 4, 1, 5, 1, 1. So, we get these different learners different data sets. And we train a learner  $L_1, L_2, \dots, L_k$  on these different data sets and we get a model. And these models have to be combined by voting. So, we use bootstrapping to generate these  $k$  learners, and train a base learner with each and combine the output by voting. So, in bagging, what we do is that we do sampling in order to get these individual data sets, we do sampling with replacement. And from each sample, we build a bootstrap sample. And if you look at each instance in the original data set, what is the probability that it is included in the sample.

Suppose, there is a sample of size  $m$  sample of size  $n$  right and suppose this data set is also of size  $n$ . So, if you take any instance what is the probability that when you draw randomly from this data set from the sample, what is the probability that a particular instance will get included in  $D_i$ ? It is  $1 - \frac{1}{n}$ . And if you do  $n$  such draws, so if this is  $n$  size  $n$  and each of the subsamples are also of size  $n$ , then this is the probability that a particular instance will get included in the sample.

And so this particular sample will get selected, and we can see that you know this can be approximated by a  $e^{-1}$  and we can see that if we use  $n$  random samples about 63

percent of the instances about 63 percent of the instances will occur in each of these samples; 63 percent of the unique instances. And other instances will be repeated this you can find out. Now, So, this is about bagging.

(Refer Slide Time: 09:20)



### Boosting

- An iterative procedure. Adaptively change distribution of training data.
  - Initially, all  $N$  records are assigned equal weights
  - Weights change at the end of boosting round
- On each iteration  $t$ :
  - Weight each training example by how incorrectly it was classified
  - Learn a hypothesis:  $h_t$
  - A strength for this hypothesis:  $\alpha_t$
- Final classifier:
  - A linear combination of the votes of the different classifiers weighted by their strength
- “weak” learners
  - $P(\text{correct}) > 50\%$ , but not necessarily much better

Next we come to a second ensemble algorithm which we call boosting. Now, in bagging we are generating the data samples  $D_1, D_2$  from the original data set.

(Refer Slide Time: 09:34)



Boosting

$D_1 - L_1 \alpha_1$   
 $D_2 - L_2 \alpha_2$   
 $D_3 - L_3 \alpha_3$

1  
2  
3  
4  
5  
6  
7  
8

And they are all getting the same probability of getting included in any of these subsets. Now in boosting, what we do is that again we have a sample. Now when we try to get  $D$

i, we have to sample from  $D$ . But now what we do is that we assign a probability with each instance, so every instance is assigned a probability. Suppose, to start with there are  $n$  instances and initially we may have we may give equal probability to all the instances. So, each of none of them has a probability of  $1/n$ .

In the first iteration, you choose  $D_1$  based on this uniform probability distribution, and you get  $D_1$  by sampling. Now on  $D_1$  you train a learner  $L_1$ ; and this learner, you apply to all these instances. Now this learner may have 100 percent accuracy on these instances you know if that is the case then it is a it is a very good learner, you no need to really proceed, so, it will have you know all of them it will do it correctly. But in most of what will happen is that this learner will label sum of these instances in  $D$  correctly and some wrongly.

Now, suppose,  $D_1$  labels this correctly, labels this wrongly, labels this correctly, labels this correctly, labels this wrongly, labels this wrongly, labels this correctly, and labels this correctly. Now what we want is we want the next learner to have independent errors, and we want the next learner to do well on those instances where the current learner has not done well. So, now what we will do is we will change the probability distribution. And more specifically, we identify those instances which were wrongly labeled by learner one, and we will increase the probability of these instances. And those which were correctly labeled we will proportionately reduce the probability.

So, what we are going to do is that we want to hide the probability of those instances which we were not correctly classified by learner  $L_1$ . And So, we will get we will have the same dataset, but now the probability distribution will change. Now we will now sample  $D_2$  according to this probability distribution and we will get a sub sample and this sub sample is more likely to contain 2 5 6 then 1 3 4 7 8, so we get  $D_2$ . And on  $D_2$  we apply learner we apply the learning algorithm and get the learner  $L_2$ .

Now, again we apply  $L_2$  on these training instances, and find out which ones  $D_2$  does error on. Suppose,  $D_2$  wrongly classifies 2 7 and three, and the rest it does correctly. So, what we are going to do now is to increase the probability of 2 3 and 7 beyond what was the current probability, and reduce the probability of the rest to keep the sum of the probabilities equal to 1.

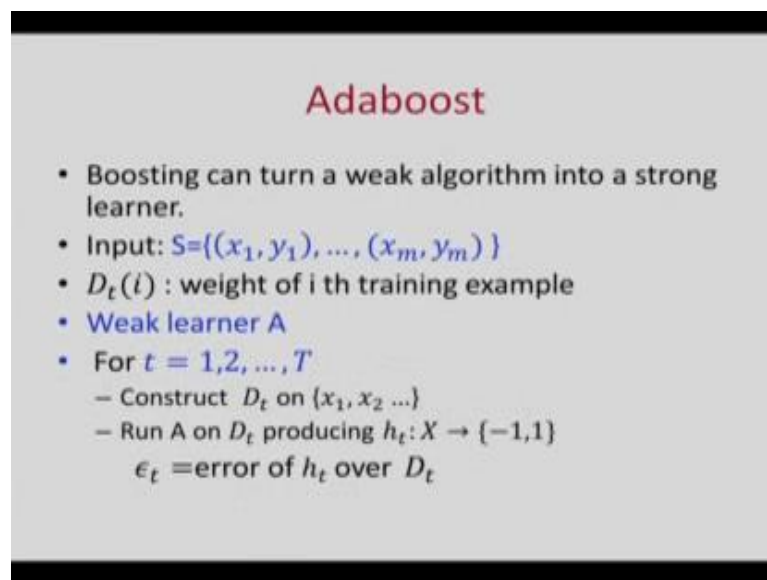
So, based on that we are going to find  $D_3$ , on which we are going to train learner  $L_3$

like this we will go on, we will get the different learners a different data and the different learners. And then we will combine these different outputs of the different learners by voting, and while voting we will assign a weight to each learner and the weight will be somewhat related to how good the learner was. So, this is the basic boosting algorithm.

So, in boosting is an iterative procedure, we start with a uniform probability distribution on the given training instances. And we adaptively change the distribution of the training data. So, initially all the training instances have equal weight after each round of boosting the weights get changed. So, we learn a hypothesis or a learner and we assign a strength to each learner we assign a strength to each learner.

And this strength is used to decide the weight of the voting, and the final classifier is a linear combination of this different learning hypothesis you know weighted by these weights right. So, what we require is that the individual what we have said earlier also the individual learners may be weak learners they must be unstable they must be weak learners. In fact, it is better if they are weak learners, but they must have more than 50 percent accuracy for a 2-class problem.

(Refer Slide Time: 15:40)

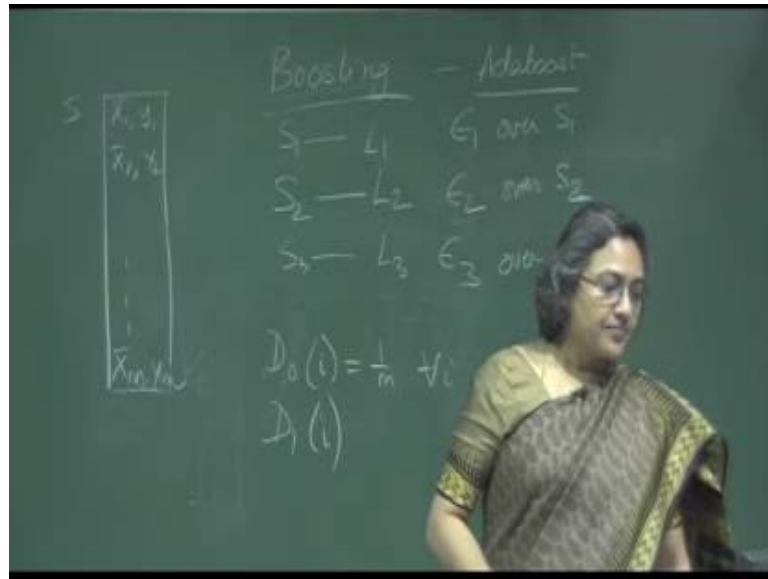


**Adaboost**

- Boosting can turn a weak algorithm into a strong learner.
- Input:  $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$
- $D_t(i)$  : weight of  $i$  th training example
- Weak learner  $A$
- For  $t = 1, 2, \dots, T$ 
  - Construct  $D_t$  on  $\{x_1, x_2 \dots\}$
  - Run  $A$  on  $D_t$  producing  $h_t: X \rightarrow \{-1, 1\}$
  - $\epsilon_t = \text{error of } h_t \text{ over } D_t$

Now, there are several boosting algorithms.

(Refer Slide Time: 15:48)



One most common algorithm for boosting is Adaboost which I will talk about in today's class. So, we have seen that boosting can turn a weak algorithm into a strong learner. So, Adaboost takes a sample as input, and let us say we call this sample  $S$ . And it contains  $x_1, y_1, x_2, y_2, \dots, x_m, y_m$ . This is the training example  $x$  is a vector of the input attributes and  $y$  is the output attribute. And we have a weight for each training example initially  $D_0(i)$  is the weight of  $i$ th training example; initially  $D_0(i)$  equal to  $1/m$  for all  $i$  between 1 to  $m$ . Initially at time 0, all the learners have equal weight which is equal to  $1/m$  and we use a algorithm.

Now, we do capital  $T$  number of iterations. In the first iteration we construct  $D_1$  on  $x_1 \times 2 \times m$ , we construct  $D_1$ ; and by sampling from this  $S$ , using the distribution  $D_0$ . So, let us write it as script  $D$ , so this is  $D_0$ . So, we get  $D_1$  or let us me call this  $S_1$ . Then what we do is that we update  $D_1$ . So, for those  $i$ , for which from  $S_1$  we get a particular learner  $L_1$ . So, we find out which training examples  $L_1$  classifies correctly, for those training examples that  $L_1$  classifies wrongly, we increase  $D_1(i)$ ,  $D_1(i)$  is increased for those examples on which  $L_1$  wrongly labels.

So, we update  $D_1(i)$ ; and from  $D_1(i)$  we sample  $S_2$ . And from  $S_2$  by running our algorithm we produce the learner  $L_2$ . And let us say when we apply the learner  $L_1$  on  $S$  the error is  $\epsilon_1$ . When I learn  $L_2$  on initial  $S$  error is  $\epsilon_2$ ; similarly  $S_3$  produces  $L_3$  with error  $\epsilon_3$ . So,  $\epsilon_t$  is the error over the current sample; this is



the error not over the original data, but over the current sample  $S_t$  is the error over  $S_t$ .

Now, with this, let us just look at please look at the slide to the boosting algorithm. So, we are given  $S$  equal to  $(x_1, y_1, x_2, y_2, \dots, x_m, y_m)$ , where  $x$  is the set of input attributes,  $y$  is the output attribute, and initialize the distribution to  $1/m$ . We initialize the distribution to  $1/m$ .

(Refer Slide Time: 19:57)



And then we run this iteration for  $t$  equal to one dot dot  $T$ . So, we trained the weak learner using  $D_t$ , and get a weak classifier, let us call it  $h_t$ . Then we choose  $\alpha_t$  as the weight associated with  $h_t$ , we will see later how  $\alpha_t$  is chosen. And we update the distribution, so we get  $D_{t+1}$  as the updated distribution which is obtained from  $D_t$  into exponential of minus  $\alpha_t y_i h_t(x_i)$ . So,  $y_i$  is the target output of the particular training example  $i$ , and  $h_t(x_i)$  is what is output by the learner  $h_t$ . So, if they agree this is 1; if they do not agree, it is minus 1

So, divided by a normalization factor this normalization factor is chosen such that the sum of the probabilities sum of this overall  $i$  is equal to 1. So,  $Z_t$  is a normalization factor. Now, we do this capital  $T$  number of times and final classifier that we output is given by  $h(x)$  equal to the sign of  $\sum \alpha_t h_t(x)$ . So,  $h_1, h_2, \dots, h_T$  are the different classifiers that are output and  $h_t$  are the different classifiers they are weighted by  $\alpha_t$ . And the total is computed if the total is positive then the class is positive if the total is

negative the class is negative. So, this is the basic Adaboost algorithm.

(Refer Slide Time: 22:57)

Given:  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in X, y_i \in Y = \{-1, +1\}$   
Initialize  $D_1(i) = 1/m$ .  
For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ .
- Get weak classifier  $h_t: X \rightarrow \mathbb{R}$ .
- Choose  $\alpha_t \in \mathbb{R}$ .
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Where  $Z_t$  is a normalization factor

$$Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Output the final classifier:

$$H(x) = \text{sign} \left( \sum_{i=1}^T \alpha_i h_i(x) \right).$$

Choose  $\alpha_t$  to minimize training error

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

where

$$\epsilon_t = \sum_{i=1}^m D_t(i) \delta(h_t(x_i) \neq y_i)$$

You can see the slide here, where the algorithm is given. So, this is our sample;  $y_i$  is either minus 1 or plus 1, this is the initialization of the probability distribution. And we run this algorithm for capital  $T$  number of iterations. In each iteration, we use the current probability distribution to generate the data; and on that data we train a classifier  $h_t$ , we find  $\alpha_t$  it will show you how  $\alpha_t$  is computed. And based on that the new probability distribution is computed which is  $D_{t+1}$ . And  $Z_t$  is the normalization factor so to make sure that the sum of this is equal to 1, and the final classifier is taken as a vote of the different  $h_t$ 's weighted by  $\alpha_t$ .

Now, let us see how  $\alpha_t$  is computed. First, we need to compute  $\epsilon_t$  which is the error of the hypothesis  $h_t$  with the current sample. So,  $\epsilon_t$  is summation over the training examples  $D_t(i)$  is the probability of the  $i$ th training example in the sample. And  $\delta(h_t(x_i) \neq y_i)$  is you know if  $i$  is misclassified this is 1, otherwise this is 0. So, what we are doing is that we are taking the number of misclassifications each you know with a taking the weighted sum of the number of misclassifications so weighted by the probability so that is  $\epsilon_t$ .

And  $\alpha_t$  is computed as half log of 1 minus  $\epsilon_t$  by  $\epsilon_t$ , so this  $\epsilon_t$  is a measure of the error and  $\alpha_t$  is a measure of the accuracy of the particular hypothesis  $h_t$ . And in the ensemble when we are combining the votes of the different hypothesis we

are combining based on alpha t. So, this is the mechanism which is followed by the Adaboost algorithm which is a very popular ensemble learning algorithm.

(Refer Slide Time: 25:18)

**Strong weak classifiers**

- If each classifiers is (at least slightly) better than random  
 $\epsilon_t < 0.5$
- It can be shown that AdaBoost will achieve zero training error (exponentially fast):

$$\frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \prod_t Z_t \leq \exp\left(-2 \sum_{t=1}^T (1/2 - \epsilon_t)^2\right)$$

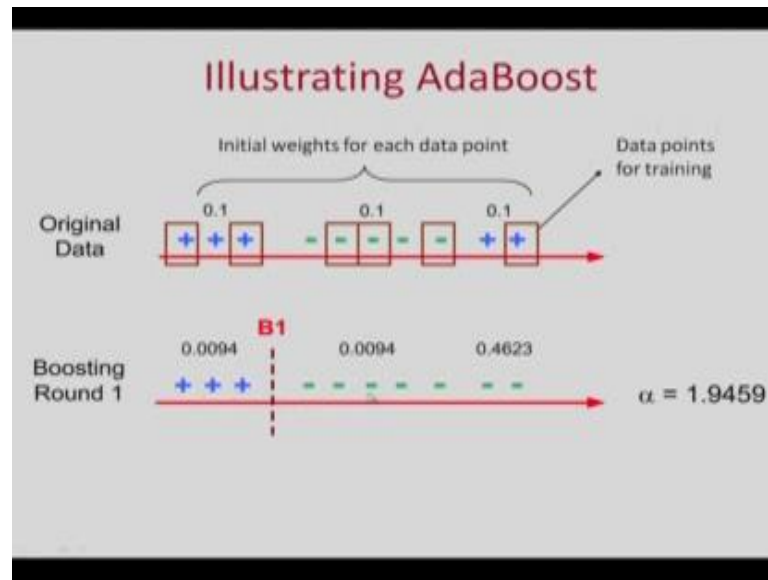
Now we will talk about a little bit more about why this ensembling works. So, if you are given a number of weak classifiers, they can be combined to get a strong classifier. These weak classifiers must have error better than random that is epsilon t must be less than 0.5 for a 2-class problem it can be shown that this algorithm adaboost will achieve zero training error exponentially fast.

Under the assumption that the errors are independent errors of the different hypothesis are independent. It is shown that  $\frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i)$  this is the average error, it can be shown the average error of this ensemble produce by a Adaboost is less than equal to the product of over all the members of the ensemble. This is the product of all the learners that we are using in the ensemble times  $Z_t$ , which is less than equal to exponential of minus 2 small t equal to 1 to capital T, this is the total number of iterations in the Adaboost half minus epsilon t whole square.

So, epsilon t, as we said has to be less than 0.5, but even if it is a small number you know it may not need not be very much smaller, but whatever it is so this is the half minus epsilon t is a fraction less than 1. And as you take this product, this you know this will become as t increases this will drop to zero exponentially fast, and therefore,

Adaboost can this algorithm if the learners are independent it can become very good as we increase the number of iterations. So, we are not going into the theory, but this is the result which has been proved.

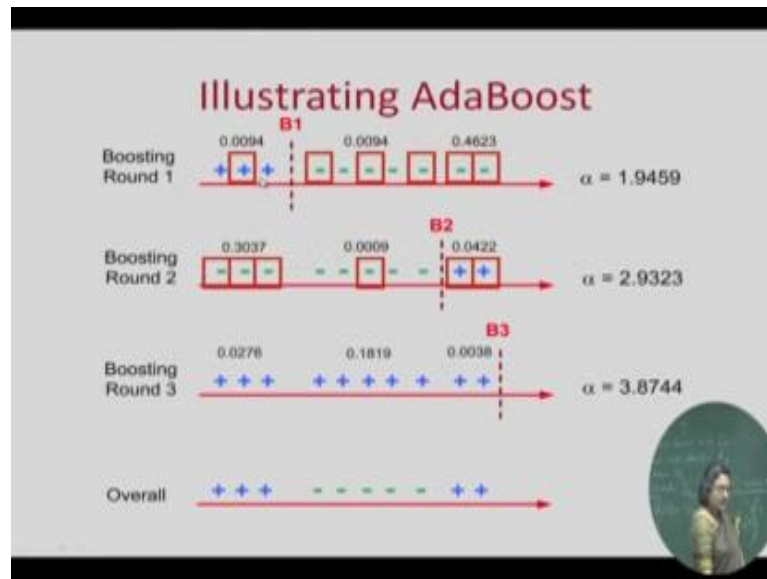
(Refer Slide Time: 27:45)



Now, we will show please look at this slide we will illustrate AdaBoost, how it works So, suppose this is the initial data. We have 10 data points 5 of them are positive which is given by green and given by blue are positive. And the 5 green ones are negative, and all of them each one of them has probability 0.1. So, sum of the probabilities is 1. Now when for boosting you select the first set of training example and the ones within squares get selected. So, these are the data points that you use for training. And based on this you get a classifier and this classifier, let us say predicts these 3 as positive and these 7 as negative.

So, we see that these three are correctly labeled as positive, but here these two are wrongly labeled as negative. So, now this probability distribution is updated. So, these two where AdaBoost makes an error their probability is boosted using the formula of boosting and they get boosted to 0.4623 and to balance the probability of these 8 examples are reduced, and they become 0.0094. Based on these values, the value of epsilon and alpha can be calculated and alpha happens to be 1.9459.

(Refer Slide Time: 29:31)



Now, what happens is the next round takes place. In the next round, we have this new distribution from which the data points are sampled. And let us say now these 6 points which are in square are sampled, and suppose the learner that to come up with this B 2, now B 2 again makes some error and based on that error you know B 2 makes error on these three examples.

And based on this the probabilities of these examples are increased, and the others are reduced, and alpha in this case is computed to be 2.9323. Then the third round takes place on this new distribution. And let us say third round chooses some samples and B 3 is this classifier. And there is an error on these values this middle values and so the error on this values based on that the probabilities are recomputed, and alpha is now computed to be 3.8744.

Finally, these 3 classifiers are combined with alpha 1, alpha 2, alpha 3 and the overall classifier now is able to label these as positive, these as negative, these as positive. So, this is in this particular case, boosting is able to combine these three classifiers into a classifier which is fully accurate it may not always happen that you get it something fully accurate.

But what is the interesting you see that these classifiers were linear classifiers they just made a division of this interval; whereas the ensemble classifier does not belong to the same type of classifier, it is able to identify that there are positive examples here and negative examples here. So, this is an illustration of the AdaBoost algorithm.

With this, I stop today's lecture on boosting.

Thank you.