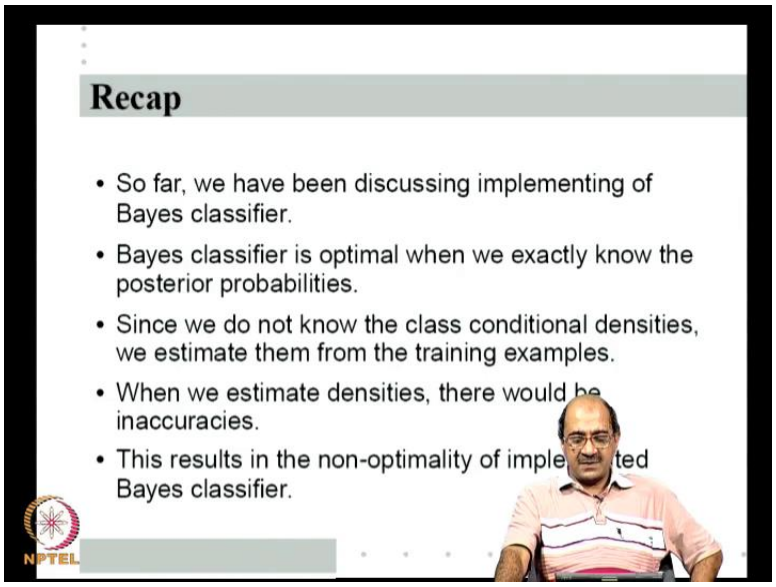
Pattern Recognition

Prof. P. S. Sastry
Department of Electronics and Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 14
Linear Discriminant Functions; Perceptron-Learning Algorithm and Convergence Proof

Hello, welcome to the next lecture in pattern recognition. In this course so far in all the lectures we had, about twelve of them so far, we have mainly concentrated on implementing of Bayes classifier. We started with the standard statistical way of formulating the problem, talked about class conditional densities, priors, posterior probabilities, looked at the Bayes classifier mainly, and then we were discussing how to implement the Bayes classifier.

(Refer Slide Time: 00:57)



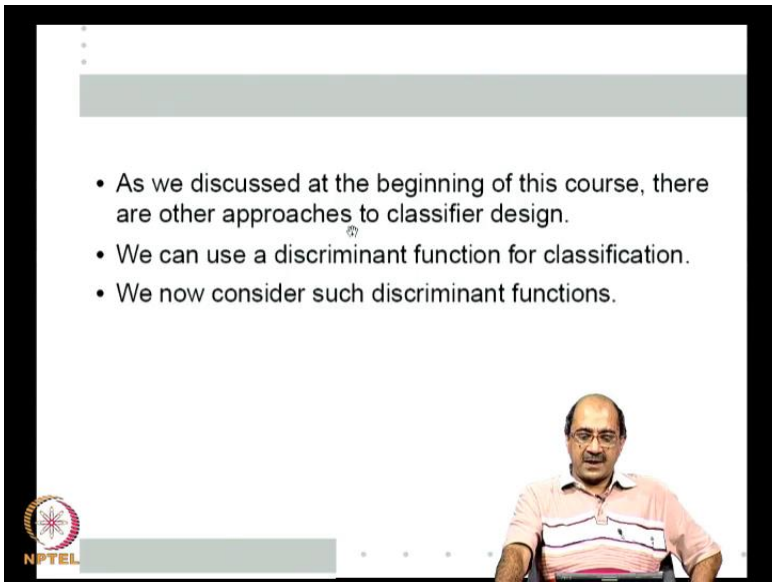
Recap

- So far, we have been discussing implementing of Bayes classifier.
- Bayes classifier is optimal when we exactly know the posterior probabilities.
- Since we do not know the class conditional densities, we estimate them from the training examples.
- When we estimate densities, there would be inaccuracies.
- This results in the non-optimality of implemented Bayes classifier.

Essentially Bayes classifier is optimal, when we exactly know the posterior probabilities. Since we do not know the posterior probabilities we use the training set of examples to estimate the class conditional densities and hence the posterior probabilities, which means, when we estimate densities, there would be inaccuracies in the densities of course. Bayes classifier is optimal only when we exactly know the posterior probabilities.

But when we use estimated densities the inaccuracies in the density estimation will translate into non-optimality of the implemented Bayes classifier. So, we have discussed this earlier so, but it is we have been looking at how to implement Bayes classifier, how to estimate densities. We considered various methods of estimating densities and how we can implement Bayes classifier with that but, since this is a problem in general, to relate how inaccuracies in density estimation affects final classification accuracy accuracies of the classifier. It is you know, it is often desirable to look at techniques other than the Bayes classifier for classification.

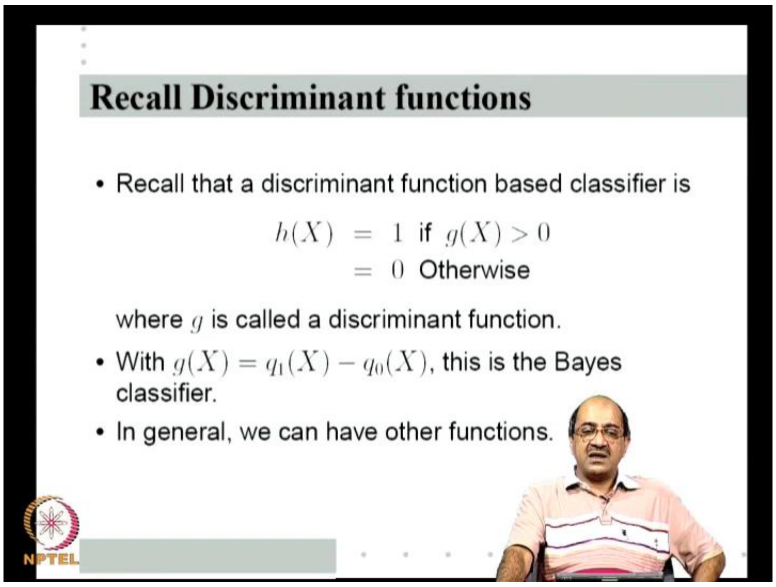
(Refer Slide Time: 02:07)



- As we discussed at the beginning of this course, there are other approaches to classifier design.
- We can use a discriminant function for classification.
- We now consider such discriminant functions.

So, as we seen in the beginning of the course when we gave a overview, there are methods other than Bayes classifier. One of them that we specifically mentioned is the discriminant function based classification. So, that is what we are going to look at next. So we will start in this class basically with linear discriminant function based classifiers what are called linear models for classification.

(Refer Slide Time: 02:33)



Recall Discriminant functions

- Recall that a discriminant function based classifier is

$$h(X) = 1 \text{ if } g(X) > 0$$
$$= 0 \text{ Otherwise}$$

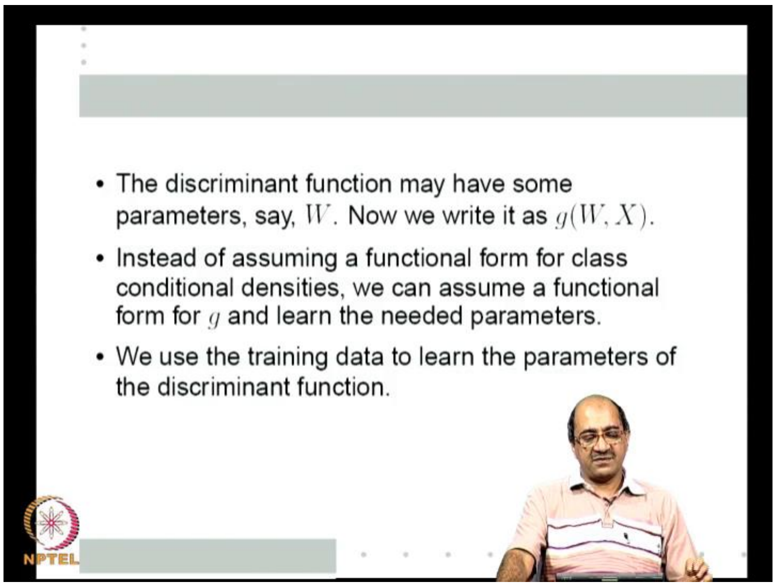
where g is called a discriminant function.

- With $g(X) = q_1(X) - q_0(X)$, this is the Bayes classifier.
- In general, we can have other functions.

So, let us recall what a discriminant based sort of classifier is. In a discriminant based classifier we like we have been doing so far we will stick to two class classification we look at multiclass discriminant functions a little later. So, for this class and the next few classes, we are essentially looking at two class classification and also regression problems we look at. So, in a two class classification problem, a discriminant function based classifier has the following structure given a feature vector x, the classifier output h of X is 1, if g x is greater than 0, is equal to 0 otherwise where g is some pre given function which is called the discriminant function.

That is how the discriminant based classifier is formulated once again, which should be one, which should be 0 is matter of notation, but in this course we will take if g X is greater than 0 then the class is 1 and now, a classifier based on that is equal to discriminant function based classifier. Sometimes, the classifier itself is called a discriminant function. For any case the function g is called the discriminant function. The Bayes classifier results if we take g (X) to be q 1 (X) minus q 0 (X) where q 1 and q 0 are the posterior probabilities of class one and class 0 respectively. So, g X greater than 0 means q 1 (X) greater than q 0 (X) in which case I will put in 1 that is the bayes classifier in the 0 1 loss function model.

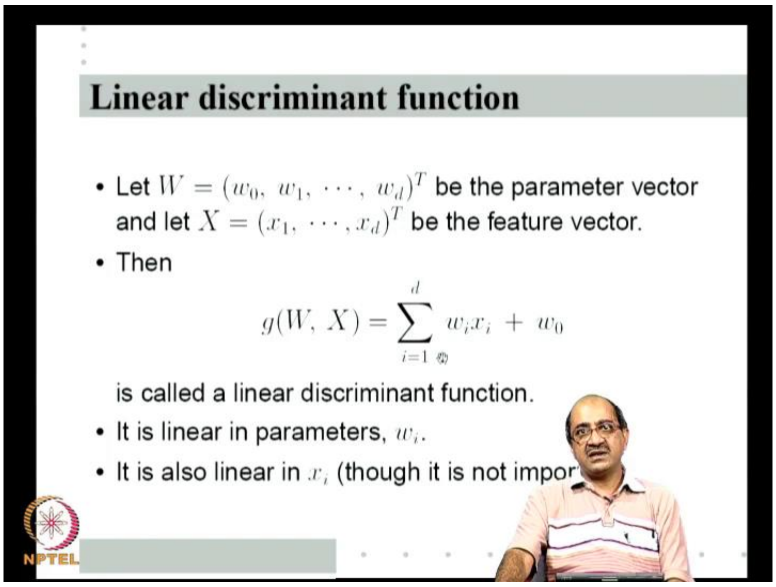
(Refer Slide Time: 04:20)



- The discriminant function may have some parameters, say, W. Now we write it as $g(W, X)$.
- Instead of assuming a functional form for class conditional densities, we can assume a functional form for g and learn the needed parameters.
- We use the training data to learn the parameters of the discriminant function.

So, even the Bayes classifier is essentially discriminant function based classifier for the issue of that we do not have to take g X to be q 1 (X) minus q 0 (X). We can take many other functions for g (x). That is how we get a richer class of classifiers. When we look at discriminant based classifiers in general, the discriminant function may have some parameters say, a vector parameter denoted by W then we write the discriminant function as g of W comma X. The idea is that so far we have been assuming some functional form for the class conditional densities and then using the training samples to estimate the densities and then using the estimated densities to implement a classifier. An equivalent method is to think of some particular functional form for g and then need the needed parameters can be estimated directly from the training samples.

(Refer Slide Time: 05:08)



Linear discriminant function

- Let $W = (w_0, w_1, \cdots, w_d)^T$ be the parameter vector and let $X = (x_1, \cdots, x_d)^T$ be the feature vector.
- Then

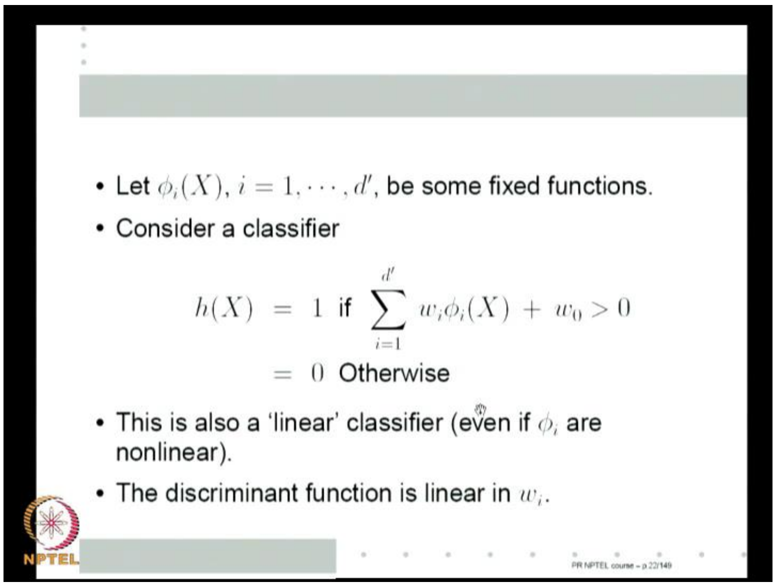
$$g(W, X) = \sum_{i=1}^{d} w_i x_i + w_0$$

is called a linear discriminant function.
- It is linear in parameters, w_i.
- It is also linear in x_i (though it is not impor

So, that is basic idea of discriminant function based classifiers we use the training data to learn the parameters in the discriminant function. Specifically, in this class we look at what are called linear discriminant functions. So let us consider discriminant function which has let us say d plus 1 parameters, the parameter vector w has components w 0, w1 , w d and the feature vector is d dimensional with components x 1 to x d. Then a discriminant function which has the following structure summation i is equal to 1 to d w i x i plus w naught is called a discriminant function. It is called a linear discriminant function. So, a linear classifier or a linear discriminant function based classifier, both of them are often used as synonymously is one where I put x in class 1 if summation w i x i plus w naught is greater than 0 otherwise, I put in class 0 .So, that is why this is a linear discriminant function.

Basically, this is called linear mainly because it is linear in the parameters w this discriminant function has d plus one parameters if the feature vector has dimension d. Those are w 0 w 1 up to w d and it is essentially a linear some with respect to w i the w i's are the unknown parameters that is what we need to learn and this function is linear in w 1's and that is the main reason it is called linear discriminant function. All the various techniques we consider for implementing such linear classifier, essentially make use of the fact that this function is linear in its parameters w i. This particular function is also linear in x i right very often one may confuse that linearity is with respect to x i while we normally write the linear discriminant function like this. The fact that it is linear in x i is not particularly important for the linear techniques of learning classifiers. So, basically it being linear in w i is what makes it a linear discriminant function.

(Refer Slide Time: 07:02)



- Let $\phi_i(X)$, $i = 1, \cdots, d'$, be some fixed functions.
- Consider a classifier

$$h(X) = 1 \text{ if } \sum_{i=1}^{d'} w_i \phi_i(X) + w_0 > 0$$
$$= 0 \text{ Otherwise}$$

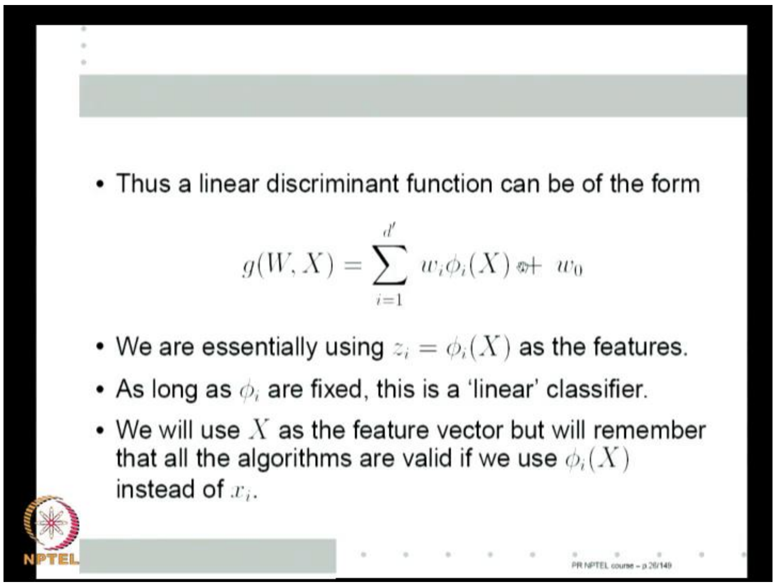
- This is also a 'linear' classifier (even if ϕ_i are nonlinear).
- The discriminant function is linear in w_i.

PR NPTEL course – p.22/149

To make this very clear let us consider generalization, let us say phi i of X i running from 1 to d prime, where d prime is some number, may be greater than d. This some fixed functions so, given any feature vector x I can compute phi 1 X phi 2 X phi d prime X each of these phi's are function of all components of X that is why I wrote it as phi i of X.

Let us say this phi i are some prefixed functions. There cannot be changes there, they are not learnt using the training sample. They are fixed and then consider a classifier like this: h X is 1 if summation i is equal to 1 to d prime w i phi i x plus w naught greater than 0. So, essentially instead of using x i the i th component of the feature vector, we were using phi i of x the idea is that this is also a linear classifier even if phi are non-linear. We would still think of this as the linear classifier, because its linear in the adoptable parameters w i the parameters w i are what are to be learnt and with respect to those parameters this is still a linear function. Hence, this is called linear discriminant function and the classifier is called a linear classifier. For any fixed phi i, because phi i are not learnt once again is it is a linear discriminant function because it is linear in the parameters w i.

(Refer Slide Time: 08:23)



- Thus a linear discriminant function can be of the form

$$g(W, X) = \sum_{i=1}^{d'} w_i \phi_i(X) + w_0$$

- We are essentially using $z_i = \phi_i(X)$ as the features.
- As long as ϕ_i are fixed, this is a 'linear' classifier.
- We will use X as the feature vector but will remember that all the algorithms are valid if we use $\phi_i(X)$ instead of x_i.

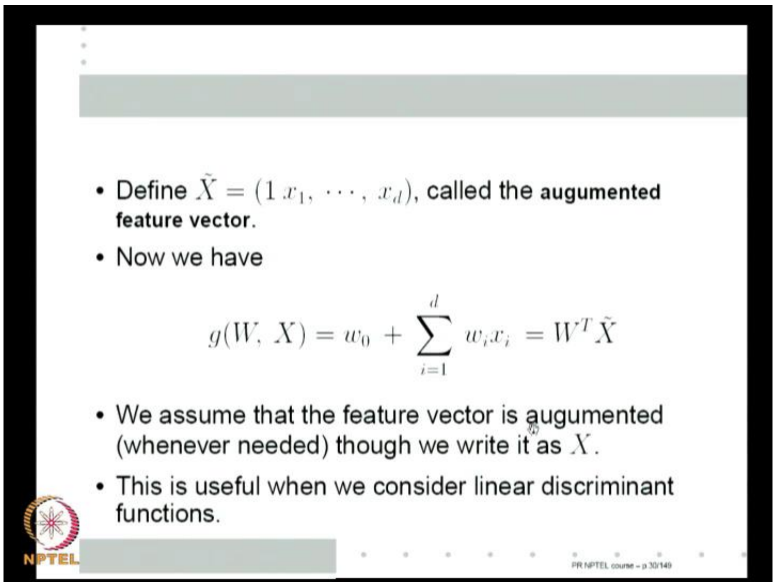
PR NPTEL course – p 26/149

This means that I can write a general linear discriminant function in the following form g W comma X is i is equal to one to d prime w i phi i X plus w naught where phi i's are some fixed functions. It essentially means that because phi i is a prefixed instead of using X i at the i-th feature I am using z i is equal to phi i of X at the i-th feature. So, if you think of z i as phi i of x then the vector z, z 1, z 2, z d prime is a new feature vector and with respect to that it is same as the old linear discriminant function.

So, that is the reason why we will still like to call all such things as linear discriminant functions and the resulting classifier structures as linear classifiers. As long as phi is fixed all these are linear classifiers. It means that all the techniques that we are going to present for linear classifiers will be valid if we, instead of using x i as the features if we use phi i of x as the features where phi 1 phi 2 and so on are some prefixed functions. For the rest of the lecture and for many lectures to follow most of the time we will use X itself for the feature vectors.

So, we will write the discriminant function as w i x i summation w i x i plus w naught, but we will remember that all the algorithms are valid if we change x i by phi of X because this will revert back to our usual notation of summation w i x i plus w naught. Sometimes, this phi i are called fixed basis functions. We will see later on in the course why that basis function name comes from.

(Refer Slide Time: 09:59)



- Define $\tilde{X} = (1\ x_1,\ \cdots,\ x_d)$, called the **augumented feature vector**.
- Now we have

$$g(W,\ X) = w_0 + \sum_{i=1}^{d} w_i x_i = W^T \tilde{X}$$

- We assume that the feature vector is augumented (whenever needed) though we write it as X.
- This is useful when we consider linear discriminant functions.

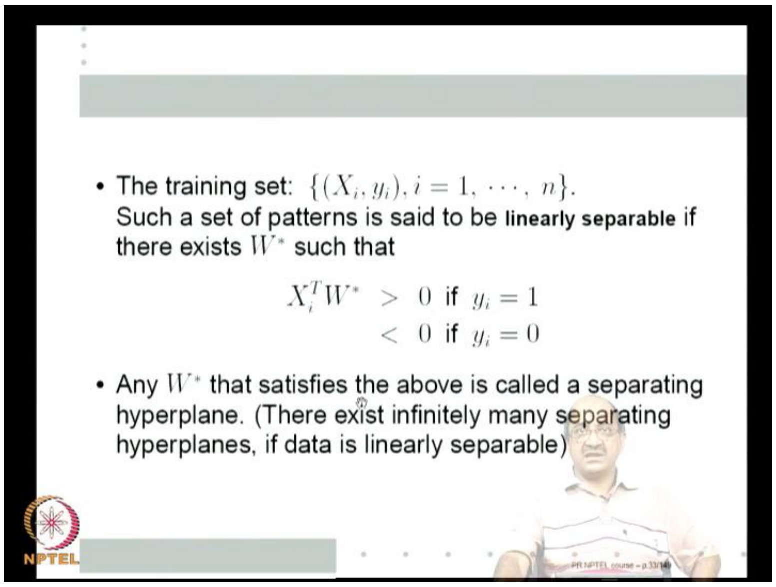
NPTEL
PR NPTEL course – p.30/149

One more small notation which is convenient when we are describing linear classifiers. Let us define a new feature vector which we call x tilde which is same as the old one except that I put an extra feature 1 extra feature whose value is 1 as the first feature. So, x tilde is 1 x 1 x 2 x d is a d plus one dimensional vector. It is called the augumented feature vector. Recall that w is the d plus one dimensional vector with components w 0 w 1 w d so what I get by this is the earlier linear discriminant function g of W comma X can be written as w 0 plus i is equal to 1 to d w x i. This is how we define the linear discriminant function can now be written as; W transpose X i this can be written as simple inner product between the parameter vector w and the augmented feature vector x tilde.

So, essentially we can think of this as simply as an inner product without worrying about the constant. So, by considering the feature vector to be augmented with a extra component which is always 1, we can write the linear discriminant function simply as W transpose X tilde. What we will do in this class as well as for the rest of our discussion on linear classifiers that we always assume the feature vector to be augmented of course.

But whenever, we need we simply assume that the feature vector is augmented so that we can write linear discriminant function as W transpose X this is useful mainly in linear discriminant functions and linear classifiers and also linear models for regression. So, we simply write this as W transpose X even though we should write it as X tilde under the implicit understanding that whenever we need we will augment the feature vector like this.

(Refer Slide Time: 11:55)



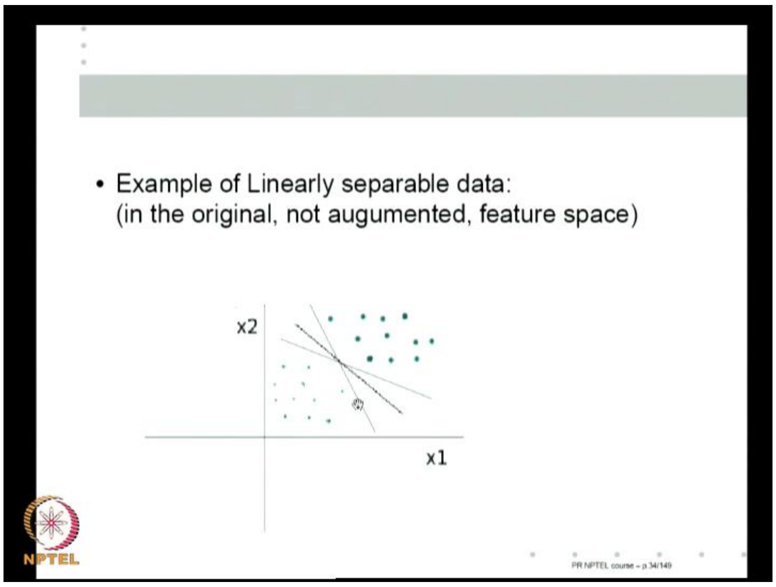
- The training set: $\{(X_i, y_i), i = 1, \cdots, n\}$.
  Such a set of patterns is said to be **linearly separable** if there exists W^* such that

$$X_i^T W^* \begin{array}{l} > 0 \text{ if } y_i = 1 \\ < 0 \text{ if } y_i = 0 \end{array}$$

- Any W^* that satisfies the above is called a separating hyperplane. (There exist infinitely many separating hyperplanes, if data is linearly separable)

Alright now, what is learning of linear classifiers as usual the training set is x i y, i going from 1 to n. So, such a set of patterns are feature vectors is said to be linearly separable if there exists a W star such that X i transpose W star is greater than 0 if y i is equal to 1 and X i transpose W star less than 0 if y i is equal to 0. It means there does exist a linear discriminant function with parameters W star, which correctly classifies all the training patterns. I am not using the augmented feature vector notation here. When I write X i transpose W star I am implicitly assuming that X i is augmented any W star that satisfies this is called a separating hyperplane. There exists infinitely many separating hyperplanes if the data is linearly separable right here is a simple example.

(Refer Slide Time: 12:58)

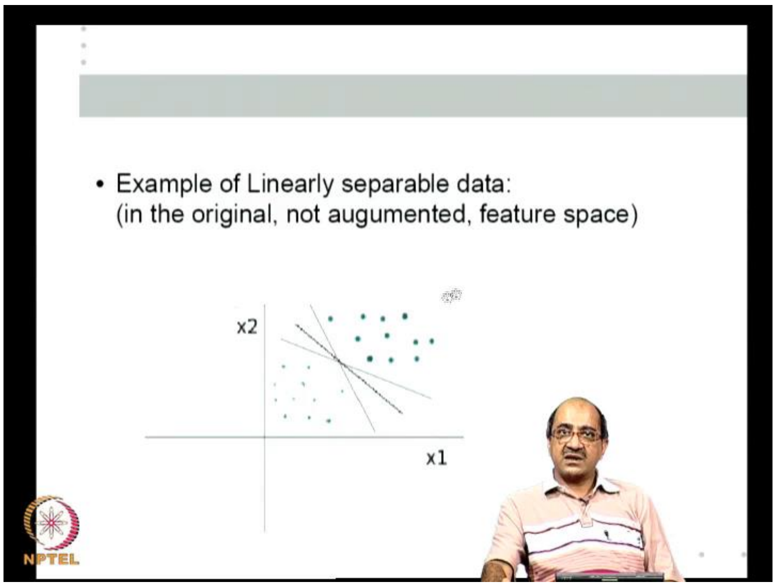


- Example of Linearly separable data:
  (in the original, not augumented, feature space)

So, it is a two dimensional data note that I am showing it in the original space not in the augmented feature space. So, this is one class, this is another class. So, they are linearly separable because I can put a line and you know any line that passes through this cone that I showed will be a separating hyperplane. Essentially, if you consider the middle line here the w will be a vector perpendicular to this hyperplane. Let us say w is in point in this direction, the direction of w is arbitrary, but suppose W's point in this direction then the dot product between w and any of these patterns. Pattern vectors would be positive and the dot product between w which is directly like this and any of these pattern vectors will be negative. That is why this pattern set is linearly separable and that is what is given by this equation.

So, essentially when you say w is a separating hyperplane representationally W star is actually the normal vector to the hyperplane. So, W transpose X is equal to 0 is the solution of the hyperplane. All points on the hyperplane are perpendicular to w. W transpose X is equal to 0 is the hyperplane. So, W transpose X greater than 0 is one side of the hyperplane. W transpose X less than 0 is the other side of the hyperplane.

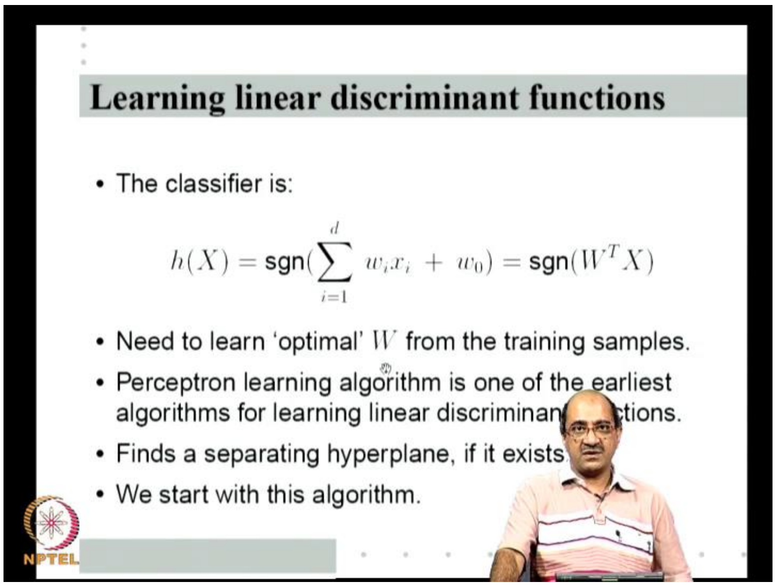
(Refer Slide Time: 14:32)


- Example of Linearly separable data:
  (in the original, not augumented, feature space)

So, when we said that patterns are linearly separable we used a strict inequality on either side which means like in this example no pattern need to be on the separating hyperplane patterns are strictly away from the separating hyperplane. It is because you have only finitely many patterns that is separable. I can always make sure that a separable hyperplane is such that no pattern is on the separating hyperplane.

So, that is the reason why we have used strictly inequalities in both sides, I hope the notation is clear. Now, the separating hyperplane is this line that separates the two classes w is actually a vector perpendicular to this hyperplane. Thus, the equation of the hyperplane is W transpose X is equal to 0. So, W transpose X greater than 0 is one side of the hyperplane, the positive side of the hyperplane that is all the vectors which have a positive inner product with the vector w which is perpendicular to the hyperplane.

(Refer Slide Time: 15:42)



Learning linear discriminant functions

- The classifier is:

$$h(X) = \text{sgn}(\sum_{i=1}^{d} w_i x_i + w_0) = \text{sgn}(W^T X)$$

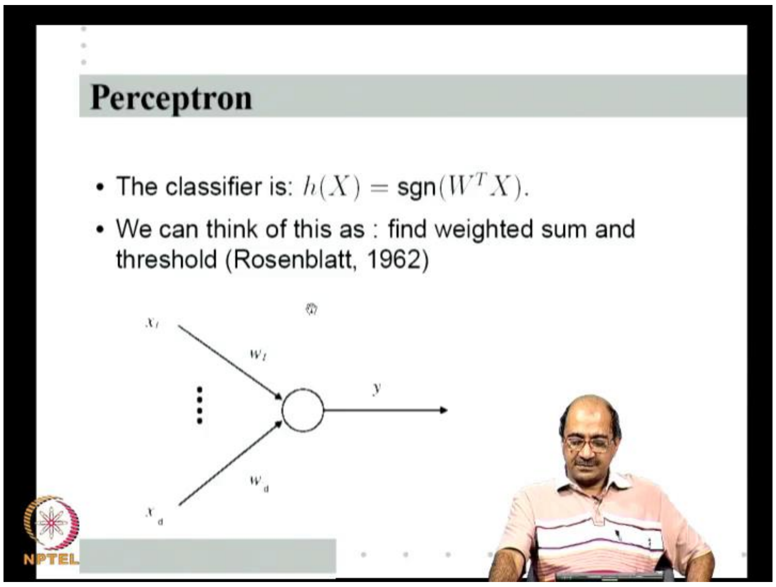
- Need to learn 'optimal' W from the training samples.
- Perceptron learning algorithm is one of the earliest algorithms for learning linear discriminant functions.
- Finds a separating hyperplane, if it exists.
- We start with this algorithm.

So, the hyperplane separate these two, the w vector is the perpendicular to the hyperplane and we also noted that when it is separable no pattern is on the hyperplane that is why both the inequalities are strict so what is learning of linear discriminant functions. So this is our classifier h (X) is 1 if the sum is greater than 0 otherwise. So, we call is the sign function s g n we will write this as s g n for this lecture.

So, I can also write as W transpose X implicitly assuming that X is augmented. So, sign of W transpose X is my classifier so, what I have to learn I have to learn the optimal values of W from the training samples there are many algorithms. We will start with the one of the oldest algorithms for this problem which is arguably the first ever pattern recognition or machine learning algorithm that is ever proposed that is called the perceptron algorithm.

The perceptron learning algorithm is one of the earliest algorithms for learning linear discriminant functions. It can find a separating hyperplane if it exists so, it works only if the pattern set is linearly separable. Later on we will see how to handle pattern sets that are not linearly separable. But to start with we look at the perceptron algorithm, which works only when the pattern set is linear probe that is what I mean by it finds a separating hyperplane, if a separating hyperplane exists so this what we are going to start with in this class.

(Refer Slide Time: 17:00)



Perceptron

- The classifier is: $h(X) = \text{sgn}(W^T X)$.
- We can think of this as : find weighted sum and threshold (Rosenblatt, 1962)

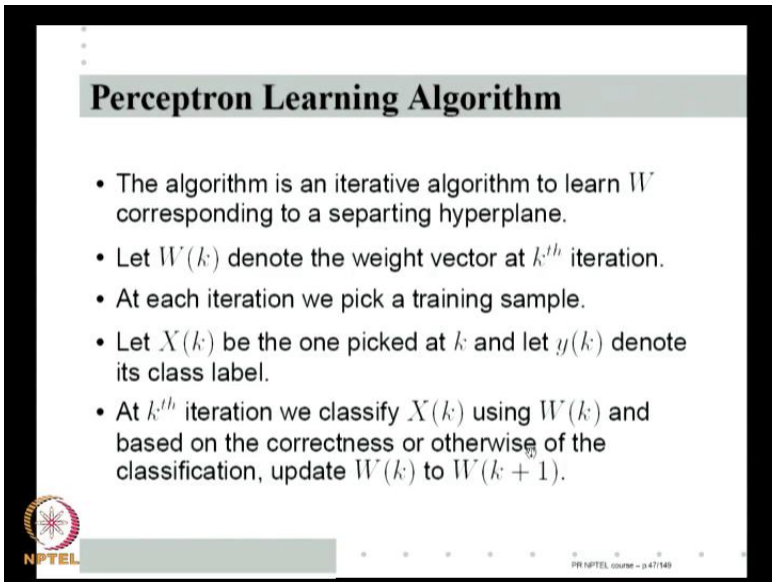
So, once again for a linear discriminant function this is the classifier sign of W transpose X. So, we can think of this as what is W transpose X. It is a kind of a weighted sum so w's are the weights, x's are the components of X are the features so you find a weighted sum of the features. The W transpose X is simply a weighted sum of the features so this is a weighted sum of the features and then the sign simply thresholds at 0 here.

So, if W transpose X is greater than 0, it has one value less that equal to 0 another value. So, we can pictorially think of it as a unit some unit into which there are various inputs coming X 1 to X t and each input line there is a weight W 1 W d. What this unit does is it finds a weighted sum of all its input W 1 X 1 plus W 2 X 2 plus W n X n and then thresholds it at W 0 because there is a constant W 0 in this.

So, the sign of this is same as finding the weighted sum W 1 X 1, W 2 X 2, W d X d and threshold you get it W 0 that is its output such a unit is what is called perceptron.

It is originally proposed by Rosenblatt in late 1950s. The particular algorithm with its convergence proof first came out in 1962 that is also due to Rosenblatt. Actually the perceptron was used as a model for neurons model for how neurons in our brain can learn which is not particularly important for us for this class. But for historical purpose perspective is this interesting to note that Rosenblatt was actually investigating how we learn to recognise mainly visual categories. This is one mathematical model he came up with. At the end of this lecture I will come back to this figure and explain a little more on that.

(Refer Slide Time: 18:58)



Perceptron Learning Algorithm

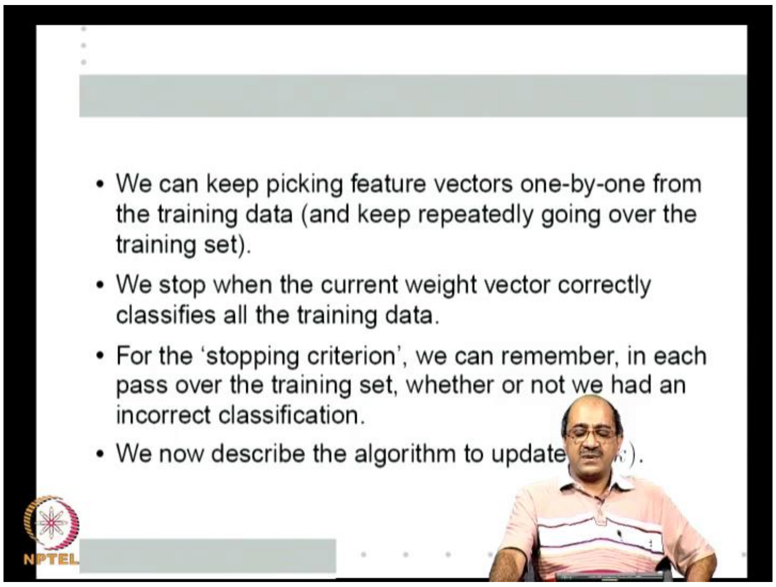
- The algorithm is an iterative algorithm to learn W corresponding to a separting hyperplane.
- Let $W(k)$ denote the weight vector at k^{th} iteration.
- At each iteration we pick a training sample.
- Let $X(k)$ be the one picked at k and let $y(k)$ denote its class label.
- At k^{th} iteration we classify $X(k)$ using $W(k)$ and based on the correctness or otherwise of the classification, update $W(k)$ to $W(k+1)$.

PR NPTEL course – p.47/149

The perceptron learning algorithm which learns the weight vector W corresponding to a separating hyperplane is an iterative algorithm iterative means it is over iterations it keeps updating. So, at k th iteration W of k W within brackets k is the value of the current weight vector. From now on we will call the parameter as weight vector because as we seen essentially what the linear discriminant function does is a weighted sum.

So, the let W(k) denote the weight vector at the k th iteration so at each iteration I update W k. So, W(k) is updated into W(k plus 1) so to that what I do is at each iteration I pick a training sample. We currently see how to, but is this almost any way of picking we will work. So, let us say X k is the one that is picked at k, so this could be one of the X 1, X subscript 1 X subscript 2 X subscript n the n pattern vectors have x subscript one tox subscript n.

(Refer Slide Time: 20:49)



- We can keep picking feature vectors one-by-one from the training data (and keep repeatedly going over the training set).
- We stop when the current weight vector correctly classifies all the training data.
- For the 'stopping criterion', we can remember, in each pass over the training set, whether or not we had an incorrect classification.
- We now describe the algorithm to update [illegible]).

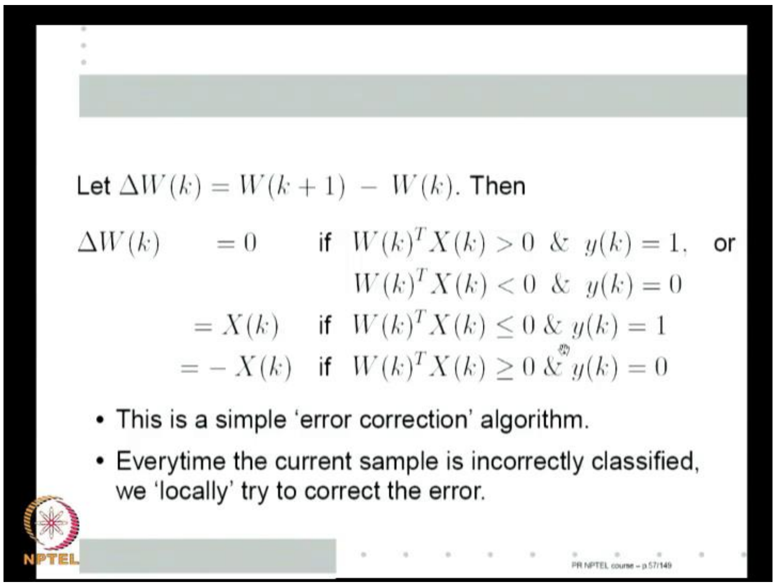
So, X of k let that denote the training sample that is picked at iteration k and let y of k be the corresponding class label as given in the training set for that pattern. What the perceptron algorithm does is, it uses the current weight vector namely W(k) to classify the current sample namely X k. And depending on whether this classification turned out to be right or wrong it updates W(k) into W(k) plus 1. That is how the perceptron algorithm works is an iterative algorithm like this you start with any arbitrary W 0. For example, I can take W 0 to be 0, then at each iteration I have current weight vector W k.

I pick one of the patterns I call that X k .Then I classify X k using W(k) and based on the correctness or otherwise of that I update W(k) to W(k) plus 1. How do I pick? I can pick training patterns sequentially. I can first X 1 then X 2 then X 3 then X n and then once again go over the training set X 1 X 1 and so on.

So, I repeatedly keep going over the training set picking feature vectors one by one till the algorithm converges. We will see what convergence means we stop when the current weight vector correctly classifies all the training data. So, at any time I am holding a weight vector in hand I pick the next training sample as I said for concreteness. Let us say I keep going over the training set repeatedly so I pick training samples as first X 1 then X 1 then X n minus and so on. Then X n minus 1 then X n then once again X 1 then once X 2 and so on. So, anytime the current weight vector correctly classifies the current sample I do nothing I we will see the algorithm correctly otherwise I change W. I keep doing this and I stop when the current weight vector correctly classifies all the training data. How would I know this we need a stopping criterion. For example, we can remember in each pass over the data whether or not we had an incorrect classification and I can remember when I had last incorrect classification.

If the last time I had incorrect classification is more than you know n iterations away M being the training sample because I know I am picking them one by one. I know the current weight vector correctly classifies. So, using some such simple technique of programming I can stop by knowing that the current weight vector correctly classifies all the training samples.

(Refer Slide Time: 22:40)



Let $\Delta W(k) = W(k+1) - W(k)$. Then

$$\begin{aligned}\Delta W(k) &= 0 && \text{if } W(k)^T X(k) > 0 \;\&\; y(k) = 1, \text{ or}\\ & && \phantom{\text{if }} W(k)^T X(k) < 0 \;\&\; y(k) = 0\\ &= X(k) && \text{if } W(k)^T X(k) \leq 0 \;\&\; y(k) = 1\\ &= -X(k) && \text{if } W(k)^T X(k) \geq 0 \;\&\; y(k) = 0\end{aligned}$$

- This is a simple 'error correction' algorithm.
- Everytime the current sample is incorrectly classified, we 'locally' try to correct the error.

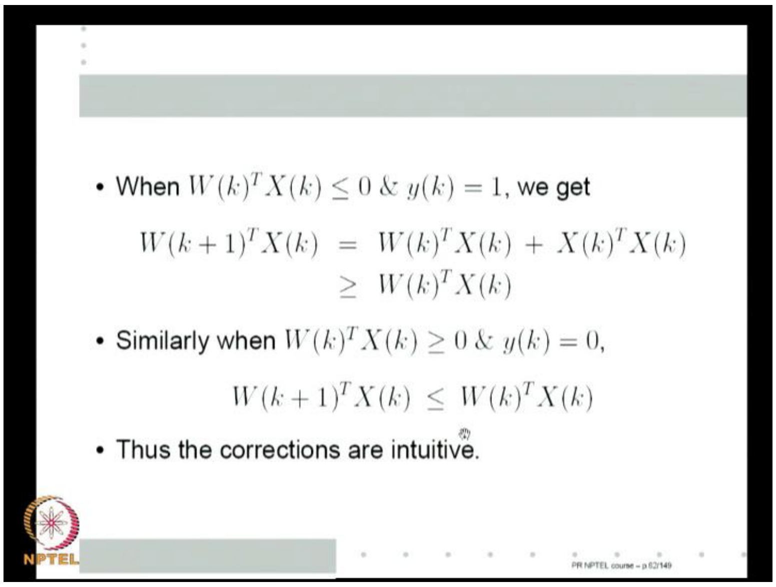
PR NPTEL course – p.57/149

So, if we have a sufficiently long run of no classification errors sufficiently long depends on how many training samples I have then I can stop. So, that is the overall view of the algorithms the only thing that is not specified is how to update W k. Now, let us see how to update w k. To update W(k) so say W(k plus 1) minus W(k) is delta W(k) that means W(k plus 1) is W(k) plus delta W k. So, I have to specify delta w k. So we specify delta W(k) as follows, delta W(k) is 0 meaning I do nothing I make W(k plus 1) is equal to W(k) if W(k) transpose X k greater than 0 and y k is 1 or W(k) transpose X y is less than 0 and y k is equal to 0. So, if the current weight vector correctly classifies the current sample I have to do nothing that is reasonable because this particular W correctly classified X. How do I know if y k is one I want W(k) transpose X k to greater than 0 and if y k is equal to 0 I want W(k) transpose X k to be less than 0.

So, this simply tells you whether or not W(k) correctly classifies X k. So, W(k) correctly classified X k I do not update I keep W(k plus 1) is equal to W(k) all right. Otherwise I have two kinds of errors. These two kinds of errors could be, y k is equal to 1 I get W(k) transpose X k less than 0 or y k is equal to 0 I get W(k) transpose X k greater than 0. So, in each of the errors we have to say what to do for W(k) here is what I do. If W(k) transpose X k is less than equal to 0 instead of being greater than 0 and y k is 1 then delta W(k) is X k. That means W(k plus 1) is W(k) plus X k which means if I make this kind of error W(k) transpose X k less than equal to 0 or y k is equal to 1. I simply add that X k to current W(k) and if I make the other kind of error W(k) transpose X k greater than 0 but, y k is equal to 0 I subtract that X k from W(k) so delta W(k) is minus X k.

So, this is the perceptron algorithm it is a kind of simple error correction algorithm. If I make no error I keep W(k plus 1) equal to W k. If I make error depending on the kind of error I either add or subtract X k. Of course, we will currently see why adding or subtracting can be called error correction, we will see it in a couple of minutes. So, it is a simple error correcting algorithm actually as we will see every time the current sample is incorrectly classified we are locally trying to correct the error. How are we locally trying to correct the error? Let us look at the error correction there are two cases one is W(k) transpose x k less than equal to 0 y k is equal to one then we add x k.

(Refer Slide Time: 25:09)



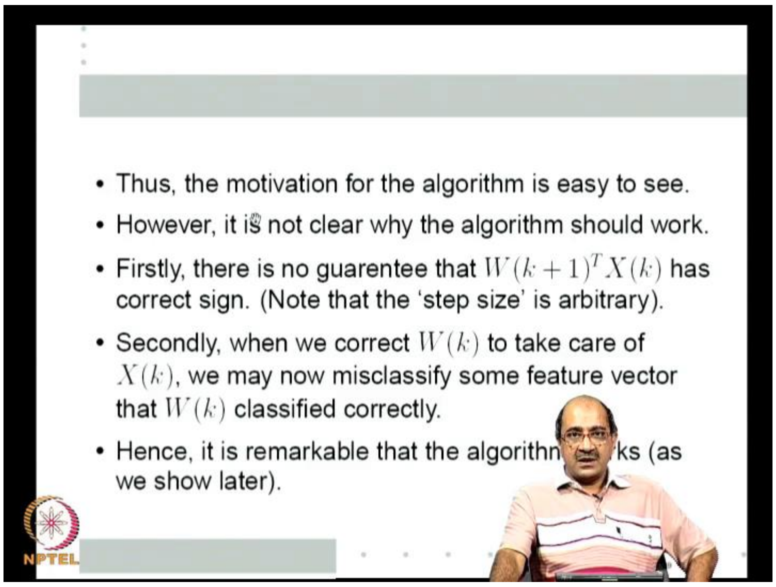
- When $W(k)^T X(k) \leq 0$ & $y(k) = 1$, we get
$$W(k+1)^T X(k) = W(k)^T X(k) + X(k)^T X(k)$$
$$\geq W(k)^T X(k)$$
- Similarly when $W(k)^T X(k) \geq 0$ & $y(k) = 0$,
$$W(k+1)^T X(k) \leq W(k)^T X(k)$$
- Thus the corrections are intuitive.

PR NPTEL course – p.62/149

Let us look at what it does. So, when W(k) transpose X k less than 0 y k is equal to 1. What is the error? When y k is equal to 1, to correctly classify X k I should have had W(k) transpose X k to be positive. Somehow I have to change W(k) so that the if I taken W(k plus 1) transpose X k it is possibly would become positive. So, what have I done now W(k plus 1) transpose X k in this particular case W(k plus 1) is W(k) plus X k. In this case, the correction is to add x k. So, W(k plus 1) transpose X k is W(k) transpose X k plus X k transpose X k so, actually I had W(k) transpose X k to be less than 0 while I wanted it to be greater than 0. So, I am ensuring that W(k plus 1) transpose X k is the old W(k) transpose X k plus some positive quantity. This means I have ensured that W(k plus 1) transpose X k is greater than W(k) transpose X k. I have made an error because W(k) transpose X k is less than 0.

So, the right direction of changing W is such that W(k plus 1) transpose X k should increase relative to W(k) transpose X k that is what adding X k does. In a similar way when W(k) transpose X k is greater than 0 and y k is equal to 0 essentially we have to decrease this inner product that is why I am subtracting X k from W k. So, W(k plus 1) transpose X k by the same token will be less than W(k) transpose X k. So, this is how the corrections are. So, in both lines I am correcting W(k) so that intuitively next time around at least I moved this W transpose X in the right direction. The corrections are intuitive is very simple as I said locally try to correct the error just for that particular X k and essentially adding and subtracting X k simply ensures that I am correcting in the right direction.

(Refer Slide Time: 27:14)

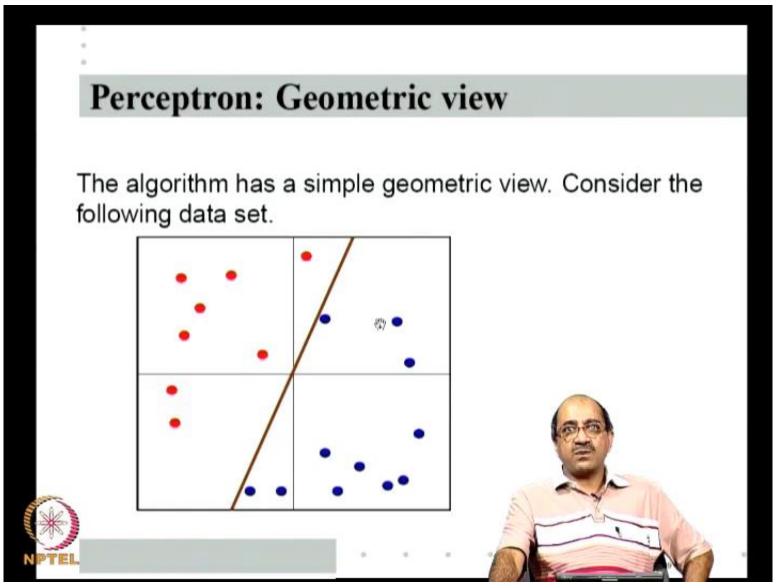


- Thus, the motivation for the algorithm is easy to see.
- However, it is not clear why the algorithm should work.
- Firstly, there is no guarentee that $W(k+1)^T X(k)$ has correct sign. (Note that the 'step size' is arbitrary).
- Secondly, when we correct $W(k)$ to take care of $X(k)$, we may now misclassify some feature vector that $W(k)$ classified correctly.
- Hence, it is remarkable that the algorithm ks (as we show later).

That is the motivation for the algorithm is easy to see is very simple minded so to say just locally try to correct the error. However, it is not clear why such a simple minded algorithm should work. I hope you people see it is very simple and what is remarkable about it is that it always finds a separating hyperplane. What do you means by its not clear why the algorithms should work? Firstly, while W(k plus 1) transpose X k is more than W(k) transpose X k when it should increase W(k) transpose X y is negative. So, I added X k so that W(k plus 1) transpose X k is more than W(k) transpose X k but, does not mean that W(k plus 1) transpose x i is become positive because so to say this step size I am just adding x and the magnitude of X I do not know.

So, I have not even ensured that at least W(k) plus one correctly classifies X. I am just moving in the right direction so there is not even a guarantee that W(k plus 1) correctly classifies X k. More importantly when I correct W(k) to take care of X k then corrected W(k) may now misclassify some other feature vectors which are classified correctly. Earlier, I might have looked at some other X which is classified correctly so, I did not do anything to W k. Now, when I modify W(k) I have no way of knowing whether what all earlier corrections I have undone. So, really there is I just do not know why this should work at all. So, given that it is really remarkable that this algorithm works we will prove it formally. I just want you to see that while its intuitive and very simple it is also a very remarkable result that such a simple algorithm always gives you a separating hyperplane.

(Refer Slide Time: 29:04)

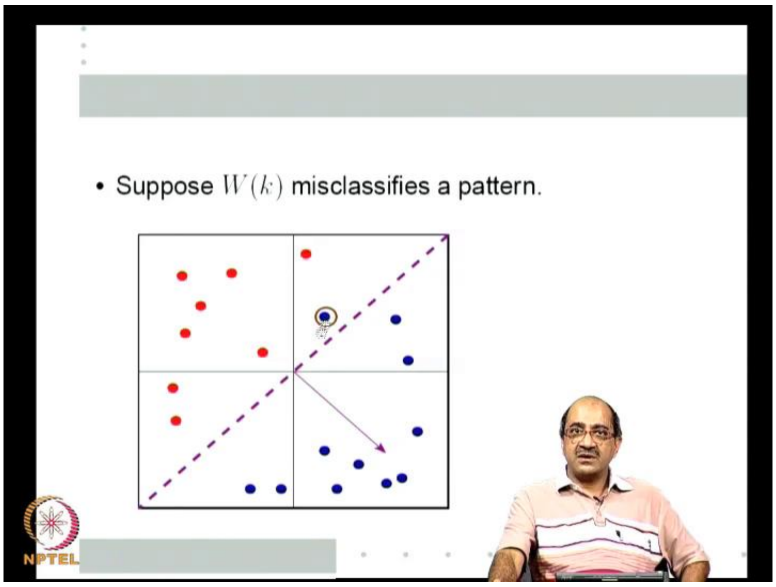


Perceptron: Geometric view

The algorithm has a simple geometric view. Consider the following data set.

There is also a geometric way of looking at the problem. So, let us look at it like this. Let us say this is my data set, all the blues are one class all the greens are other class and these are linearly separable. Note that in the augmented feature space because I am talking about W transpose X greater than 0 or less than 0 and W transpose X equal to 0 is the hyperplane. The hyperplane always passes through the origin in the augmented feature space ok. So of course, the feature vectors are closed but, not on the hyperplane. I have taken a slightly tough problem but, the two classes are still linearly separable now.

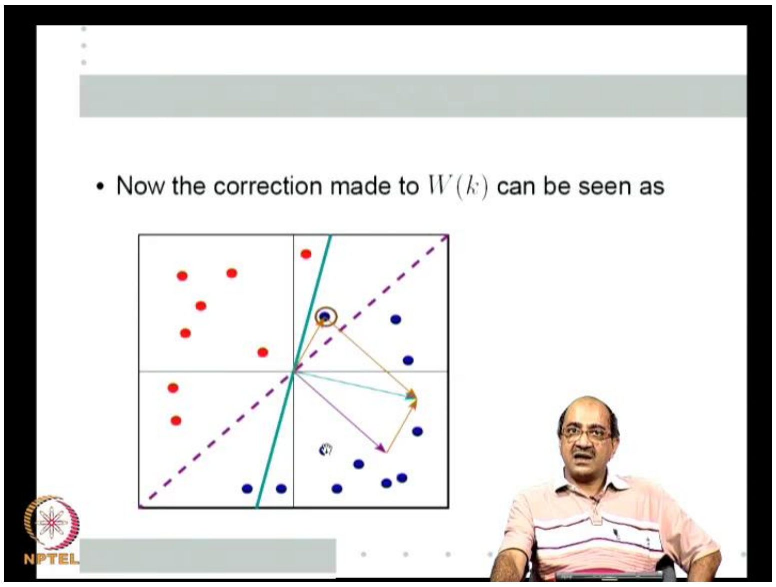
(Refer Slide Time: 29:49)



- Suppose $W(k)$ misclassifies a pattern.

Let us say on this problem at some iteration we had one particular W vector. So, let us say this is the hyperplane I am currently considering. This is the normal to the hyperplane and let us says this is the positive direction. So, this is my W vector, the vector that is shown is the W vector and the line shown in dashes the magenta line is the current hyperplane. So, all my blues were on the positive class all my reds are in negative class. So, there is one blue thing which is circled which is incorrectly classified by this particular hyperplane. So, what is that I do for correction I take this W vector to that I add the X vector. X vector is the line that joins origin to this incorrectly classified point and that gives me new W(k) plus 1. Let us do that computation.

(Refer Slide Time: 30:34)

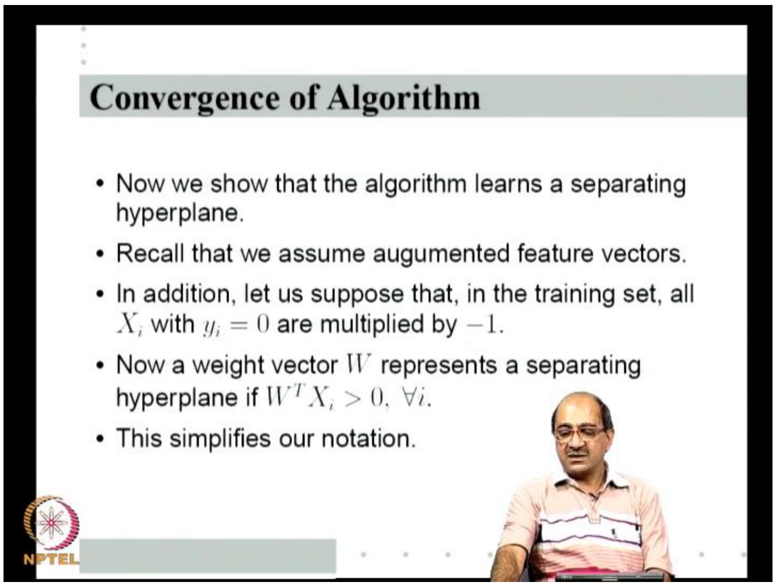

- Now the correction made to $W(k)$ can be seen as

So, as earlier that is my earlier W vector and this line is the X k that is incorrectly classified. Now, I have shown you the parallelogram law these are vector addition so, if I add this vector to that vector I get this green vector. So, that is the new W(k plus 1) which means this is the new hyperplane. So, I have effectively adding this incorrectly classified X to this W means I am rotating this hyperplane around the origin in the direction in which this particular sample is in this case is in the counter clockwise direction.

So, adding this X to W is simply rotating so adding and similarly, subtracting will be rotating in the other direction because subtracting is simply reversing the direction of that vector and adding. So, I am just rotating the hyperplane so each correction is essentially rotating the hyperplane. So, that is a direction so that the incorrectly classified sample comes on the hopefully comes on the right side of the hyperplane. In this particular example by this amount of rotation the incorrectly classified sample is now come on the right side.

However, earlier this hyperplane was correctly classifying that sample. That sample was on the positive side of the dashed hyperplane. Now, when I rotated this to take care of this. This has come on the wrong side of the hyperplane. Now, it has come on the negative side hyperplane right. So, this shows us both that the adding X and subtracting X to W is simply rotating the hyperplane in the correct direction as far as that particular sample is concerned. Now, the rotation of course in this particular example rotation action is sufficient to correctly classify that sample that is one problem. The second problem is when I rotate like that something else that was correctly classified earlier may now become incorrectly classified. As I said that keeps on going I will keep going on the training samples repeatedly. I keep rotating the hyperplane this way, that way every time. I see the example and somehow at the end I come to the. So, let us get over this again. So, that is the original hyperplane that is the wrongly classified sample so I have to rotate the hyperplane counter clockwise to take care of it and that is what my perceptron algorithm does.

(Refer Slide Time: 33:10)



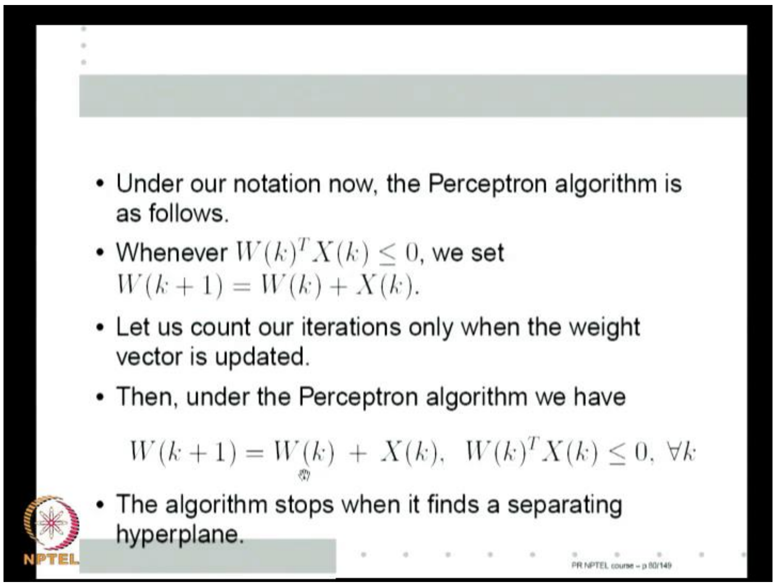
Convergence of Algorithm

- Now we show that the algorithm learns a separating hyperplane.
- Recall that we assume augumented feature vectors.
- In addition, let us suppose that, in the training set, all X_i with $y_i = 0$ are multiplied by -1.
- Now a weight vector W represents a separating hyperplane if $W^T X_i > 0, \forall i$.
- This simplifies our notation.

Now, given that we got the algorithm, let us look at the convergence. So, we now show that the algorithm learns a separating hyperplane. How do I show that? We first before we go there, Let us put some notation. First recall that all feature vectors are augmented in this we assuming all feature vectors augmented that is why as I said the hyperplane always passes through the origin. In addition, let us suppose that in the training set we will first take the training set and then multiply all x i whose corresponding y i are 0 with minus 1. What does this do to me this means that a weight vector W represents a separating hyperplane if W transpose X i greater than 0 for all i. I do not have to look at y i anymore if I simply multiplied all X i whose corresponding y i is equal to 0. Then I do not need y i for implementing my algorithm.

Now, I am working towards finding a W so that W transpose X i greater than 0 for all i because if it is a y i is equal to plus 1 vector then W transpose X i greater than 0. If y i is equal to 0 class then the original X i has been multiplied by minus 1 to make this X i. So, if this W transpose X i is greater than 0 for the original feature vector will be negative right so by this simple notational device of assuming that in the training set is equal to 0 class feature vectors are multiplied by minus 1.

(Refer Slide Time: 34:56)



- Under our notation now, the Perceptron algorithm is as follows.
- Whenever $W(k)^T X(k) \leq 0$, we set $W(k+1) = W(k) + X(k)$.
- Let us count our iterations only when the weight vector is updated.
- Then, under the Perceptron algorithm we have

$$W(k+1) = W(k) \; + \; X(k), \;\; W(k)^T X(k) \leq 0, \; \forall k$$

- The algorithm stops when it finds a separating hyperplane.

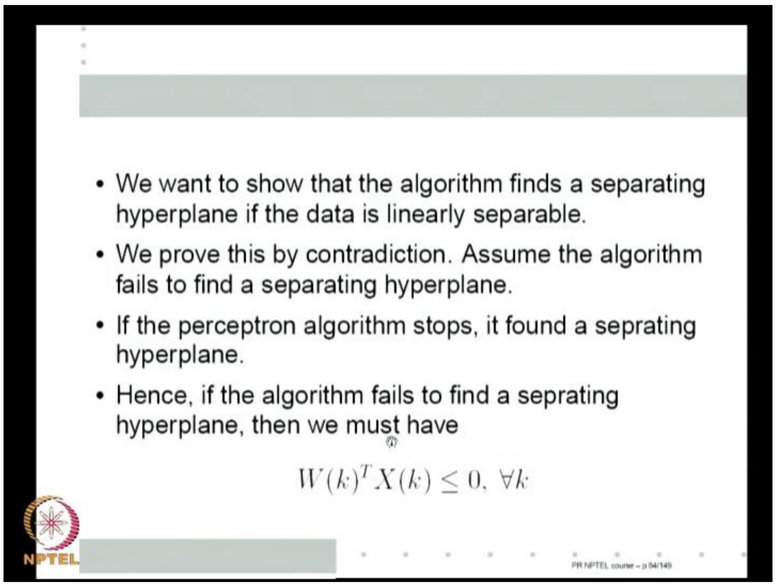
PR NPTEL course – p.80/149

Now, a separating hyperplane is simple defined by W transpose X i greater than 0 for all i and similarly, this simplifiers or notation as follows and I have notation the perceptron algorithm. Simply, if W transpose x, W(k) transpose X k is less than equal to 0, we add x there is no adding or subtracting anymore because we are adding y, y is equal to 1 patterns and subtracting y is equal to 0 patterns, but all the y is equal to 0 patterns are multiplied by minus 1.

So, for all patterns is only addition now my algorithm is very simple. If W(k) transpose X k greater than 0 do nothing if W(k) transpose X k less than equal to 0 add X k to W k. So, by the simple device of assuming that all the 0 class patterns, I have multiplied by minus 1. I can notationally simplify my perceptron algorithm. The second simplification we make is let us say we count our iterations only when the weight vector is updated. So, as k goes every iteration picking up an X k if W(k) transpose X k greater than 0 we are saying W(k) plus one is equal to W(k) if because W(k) plusone is equal to W(k) i can simply assume k is not incremented.

So, I increment k only whenever I have a correction so that i get successively different W's. So, I consider a sequence updated like that where k increments only when there is a correction sounder. This is the perceptron algorithm till it actually converges. Thus what this for all k means not really for all k till it actually converges is like this is always W(k) transpose X k less than equal to 0 and W(k plus 1) is equal to W(k) plus X k because that is how am I am counting my k's. The algorithm stops when it finds a separating hyperplane. So, it keeps on going like that after some k it would not update at all if the k would not increase because everything is correctly classified. So, if it stops then because everything is correctly classified that is the final W(k) so there will be a maximum k at which kit will stop or till that time. This is the algorithm because I am incrementing my iteration count only when I had an error.

(Refer Slide Time: 37:10)



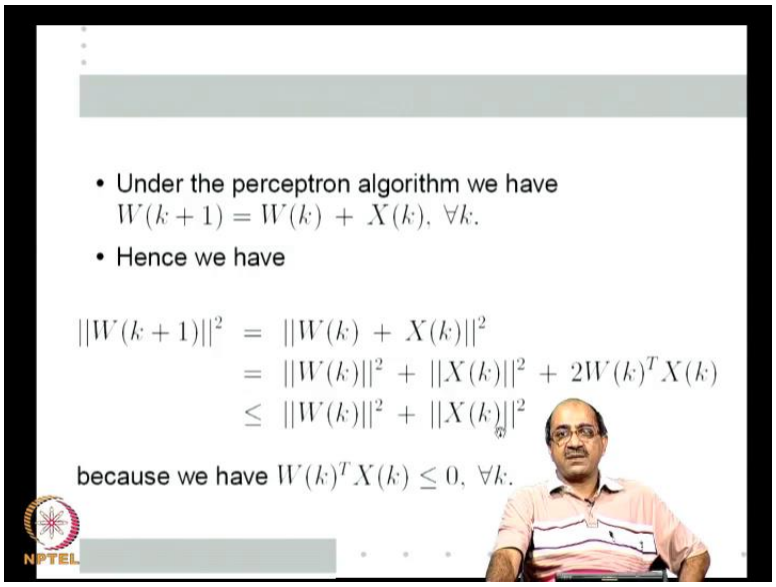
- We want to show that the algorithm finds a separating hyperplane if the data is linearly separable.
- We prove this by contradiction. Assume the algorithm fails to find a separating hyperplane.
- If the perceptron algorithm stops, it found a seprating hyperplane.
- Hence, if the algorithm fails to find a seprating hyperplane, then we must have

$$W(k)^T X(k) \leq 0, \ \forall k$$

PR NPTEL course – p.94/149

Now, we want to show that the algorithm finds a separating hyperplane if the data is linearly separable. We prove this by contradiction that is, we first assume that the algorithm fails to find a separating hyperplane even though 1 exists and then show that it cannot be. If the perceptron algorithm ever stops that it W(k) is no longer updated and W(k) transpose X k but, greater than 0, it remains greater than 0 for all k then it has found a separating hyperplane. So, what it means is that if the algorithm fails. We are assuming that the algorithm fails to find a separating hyperplane that means, this keeps on going but, at every k W(k) transpose X k less than or equal to 0. Hence, W(k plus 1) is equal to W(k) plus X k that keeps on going for all k. The k does not stop ever k keeps going. So, if perceptron algorithm fails to find a separating hyperplane then we must have this for all k.

(Refer Slide Time: 38:07)



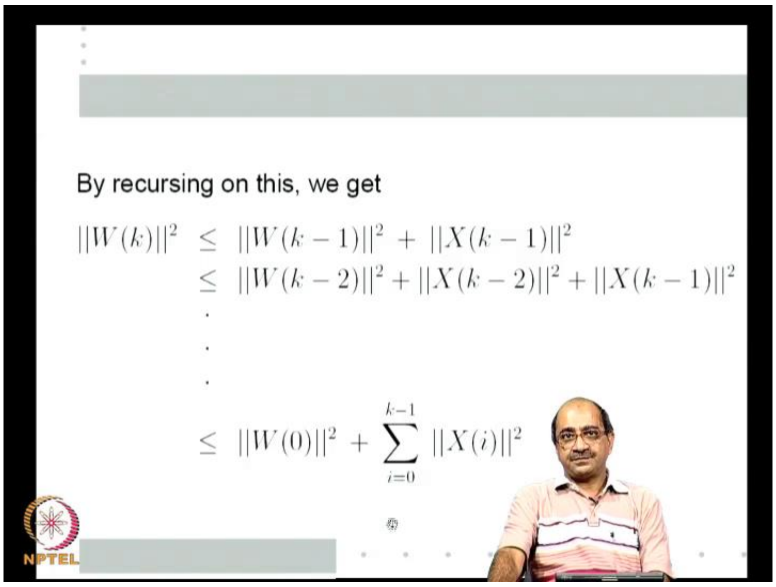
- Under the perceptron algorithm we have $W(k+1) = W(k) + X(k), \forall k.$
- Hence we have

$$\begin{aligned} \|W(k+1)\|^2 &= \|W(k) + X(k)\|^2 \\ &= \|W(k)\|^2 + \|X(k)\|^2 + 2W(k)^T X(k) \\ &\leq \|W(k)\|^2 + \|X(k)\|^2 \end{aligned}$$

because we have $W(k)^T X(k) \leq 0, \forall k.$

So, now under that let us see a contradiction under the perceptron algorithm. We have W(k plus 1) is equal to W(k) plus X k now I am saying that this keeps on going without a stop. The iteration count keeps increasing. Now, using this we know W(k plus 1) norm square, these two lines denote norm. So, W(k plus 1) norm square because W(k plus 1) is this is W(k) plus X k norm square. Now, let us expand this norm square to get W(k) norm square plus X k norm square plus 2 W(k) transpose X k. Now, our iteration count or iterations are such that for each k W(k) transpose X k is always less than or equal to 0. So, which means W norm of W(k plus 1) whole square is less than equal to norm of W(k) the whole square plus norm of X k whole square because this is always less than or equal to 0.

(Refer Slide Time: 39:11)

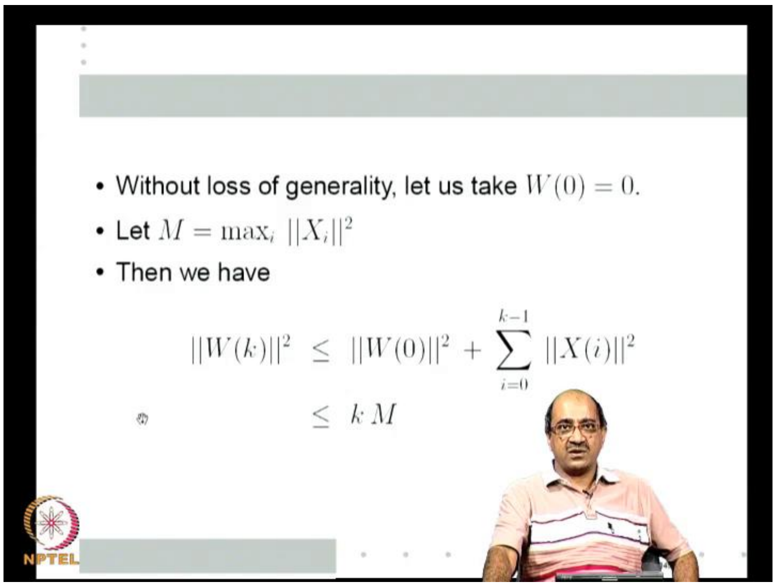


By recursing on this, we get

$$\begin{aligned} ||W(k)||^2 &\leq ||W(k-1)||^2 + ||X(k-1)||^2 \\ &\leq ||W(k-2)||^2 + ||X(k-2)||^2 + ||X(k-1)||^2 \\ &\cdot \\ &\cdot \\ &\cdot \\ &\leq ||W(0)||^2 + \sum_{i=0}^{k-1} ||X(i)||^2 \end{aligned}$$

So, now I can re curs on this so I had W(k plus 1) so I have put k so W(k) norm square is W(k) minus 1 norm square plus X k minus 1 norm square. Now, the same thing is true for W(k) minus 1 norm square that is W(k) minus 2 norm square plus X k minus 2 norm square plus X k minus 1 norm square and so on. So, if I recurs on that ultimately I get W 0 norm square plus X i norm square I is equal to 0 to k minus 1. Note that this is X brackets i. This is the i th x I picked while going down the things which has the i th time I had to correct right. So, this X within brackets i would be one of the X 1 to X n. I do not know which one but, that happened to be the i th time I corrected the W as I kept going again and again over the features. So, I get W(k) norm square given by this.

(Refer Slide Time: 40:05)

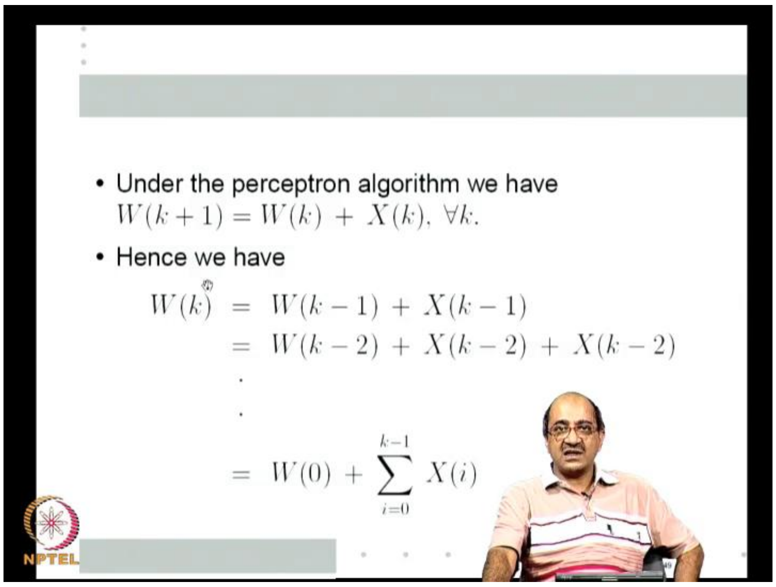


- Without loss of generality, let us take $W(0) = 0$.
- Let $M = \max_i \ ||X_i||^2$
- Then we have

$$\begin{aligned} ||W(k)||^2 &\leq ||W(0)||^2 + \sum_{i=0}^{k-1} ||X(i)||^2 \\ &\leq k\,M \end{aligned}$$

Without loss of generality as assumed W 0 is equal to 0. We can do the same proof without that, but there are only finitely many vectors X i so each of them have some length some norm. So, let M denote the maximum norm of all the X i's now, we have W(k) norm square is W 0 norm square plus this. So, because the W 0 is 0 and maximum of X i norm square is M. This simply becomes k times M. So, norm of W(k) whole square is bounded above by k times a constant so it increases linearly with k. So, if the perceptron algorithm keeps making error sometime or the other as I keep going again and again on the samples so that the iteration counts keeps increasing then the norm of W(k) keeps increasing linearly with k. This is one relation that we got.

(Refer Slide Time: 41:01)



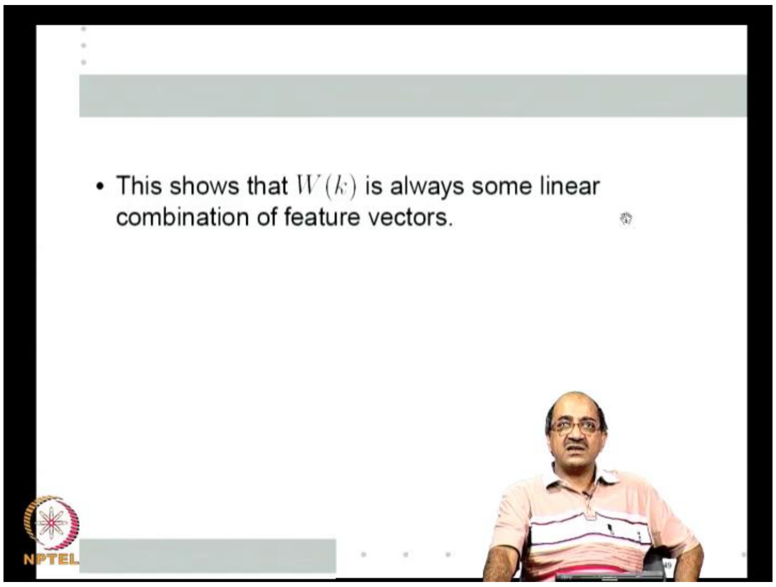
- Under the perceptron algorithm we have $W(k+1) = W(k) + X(k), \forall k.$
- Hence we have

$$\begin{aligned} W(k) &= W(k-1) + X(k-1) \\ &= W(k-2) + X(k-2) + X(k-2) \\ &\cdot \\ &\cdot \\ &= W(0) + \sum_{i=0}^{k-1} X(i) \end{aligned}$$

We will get one more once again under the algorithm W(k plus 1) is equal to W(k) plus X k. Hence, W(k) can be written as W(k) minus 1 plus X k minus 1. Now, W(k) minus 1 can be written as W(k) minus 1 plus X k minus 1 plus X k minus 1. This should be, I am sorry this is X k minus 1 and so on you recurs you get W 0 plus i is equal to 0 to k minus 1 X i. Please note that this k minus 2 is actually k minus 1.

So, using under this algorithm W(k) is simply W 0 plus i is equal to 0 to k minus 1 X i. This is not surprising because every time we are adding the X. Some X 1 of the feature vectors which happened to be incorrectly classified is added to W. So, at any given time, the current W is simply sum of sum of the X i those X i's that I have picked up and made which are incorrectly classified added to W 0 of course. W 0 is taken to be 0. So, while this is not really important for us in the proof as such of this relation is important.

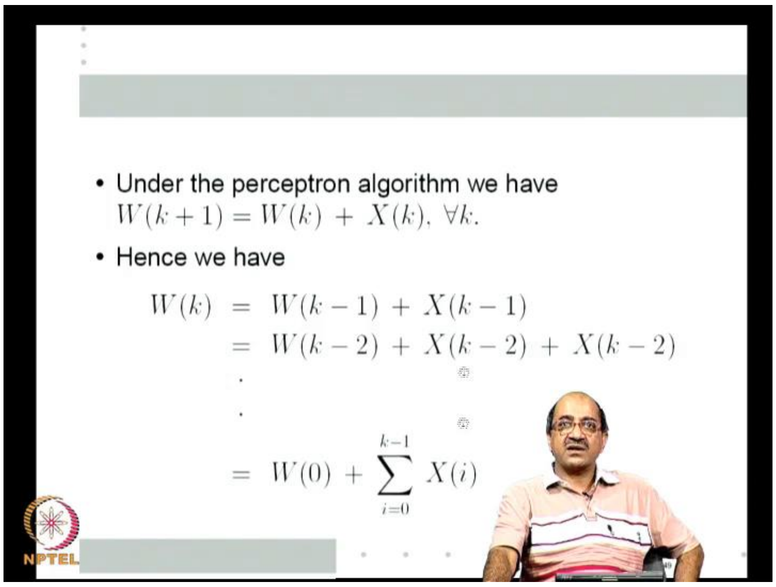
(Refer Slide Time: 42:12)



- This shows that $W(k)$ is always some linear combination of feature vectors.

This relation also tells us one thing extra namely that this shows that W(k) is always some linear combination of the feature vectors.

(Refer Slide Time: 41:01)

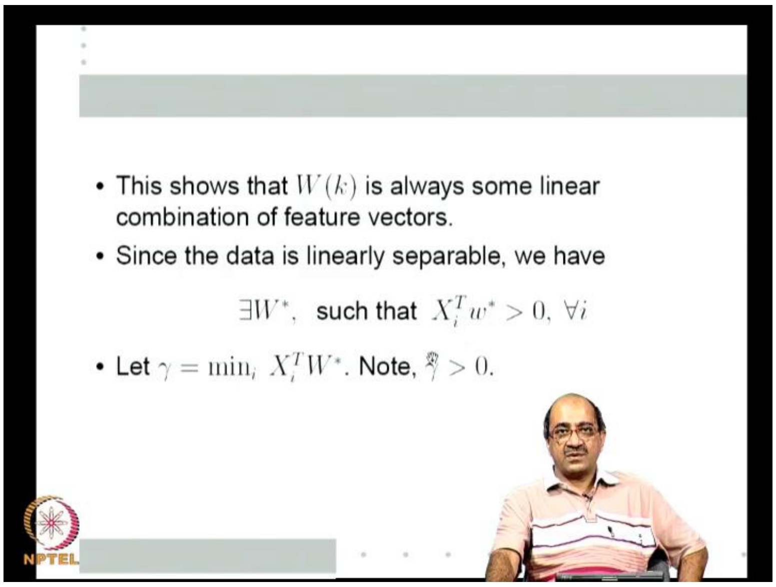


- Under the perceptron algorithm we have $W(k+1) = W(k) + X(k), \forall k.$
- Hence we have

$$\begin{aligned} W(k) &= W(k-1) + X(k-1) \\ &= W(k-2) + X(k-2) + X(k-2) \\ &\cdot \\ &\cdot \\ &= W(0) + \sum_{i=0}^{k-1} X(i) \end{aligned}$$

Each of X brackets i is one of the feature vectors while I do not know which one ultimately this sum will be some linear combination of the feature vectors. So, at any given time W(k) is a linear combination of the feature vectors. Hence, my separating hyperplane because we will ultimately going to show this algorithm minus separating hyperplane would be actually a linear combination of feature vectors. While that fact is not really important in this proof that the separating hyperplane will be a linear combination of feature vectors will become useful to us later in the course, so I have just mentioned it, but anyway let that be.

(Refer Slide Time: 42:46)



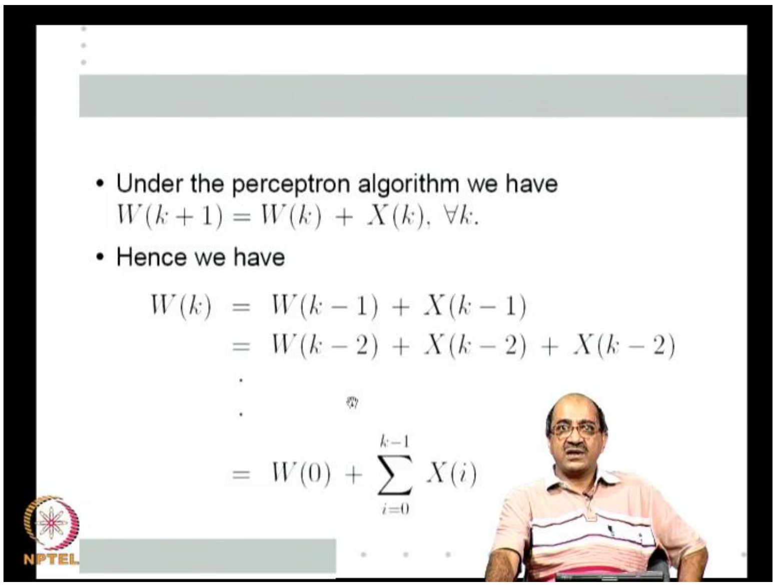
- This shows that $W(k)$ is always some linear combination of feature vectors.
- Since the data is linearly separable, we have

$$\exists W^*, \text{ such that } X_i^T w^* > 0, \ \forall i$$

- Let $\gamma = \min_i \ X_i^T W^*$. Note, $\gamma > 0$.

Now, so far we have not used the fact that we are assuming data to be linearly separable because we are assuming data to be linearly separable there exists a w star such that x subscript i transpose w star is greater than subscript i. When use this specific i th training sample x within brackets i is the as i am going down the feature vector down the training sample keep picking feature vectors that is the i th time I made a correction. So, that can be any of these X’s but, this X subscript i transpose is feature vector. So, for each of the i is equal to 1 to n X i transpose w star is greater than 0 such a w star exists. Now, let gamma be minimum over all these X i transpose w star now because each of them is greater than 0 this gamma will also be greater than 0.

(Refer Slide Time: 43:42)

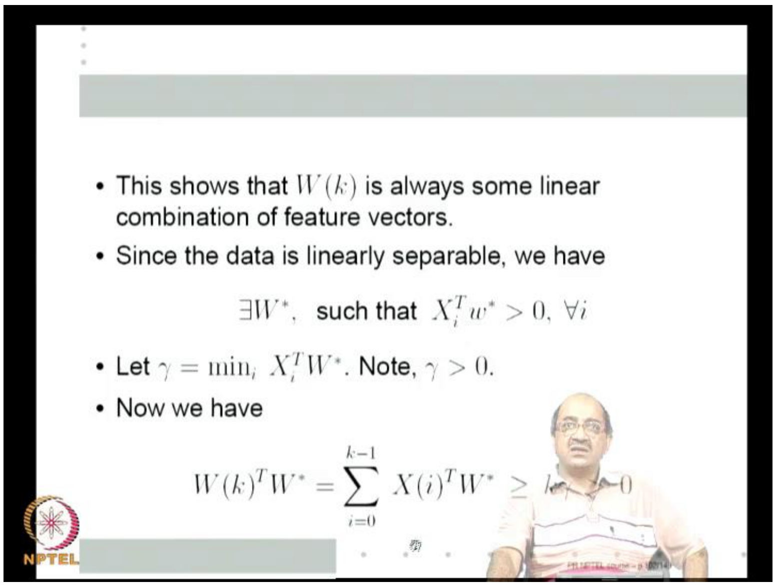


- Under the perceptron algorithm we have $W(k+1) = W(k) + X(k), \forall k.$
- Hence we have

$$\begin{aligned} W(k) &= W(k-1) + X(k-1) \\ &= W(k-2) + X(k-2) + X(k-2) \\ &\cdot \\ &\cdot \\ &= W(0) + \sum_{i=0}^{k-1} X(i) \end{aligned}$$

Now, we had W(k) is this is anyway 0. So, if I prove 2 W(k) transpose w star that will be sum of X i transpose w star.

(Refer Slide Time: 43:51)



- This shows that $W(k)$ is always some linear combination of feature vectors.
- Since the data is linearly separable, we have

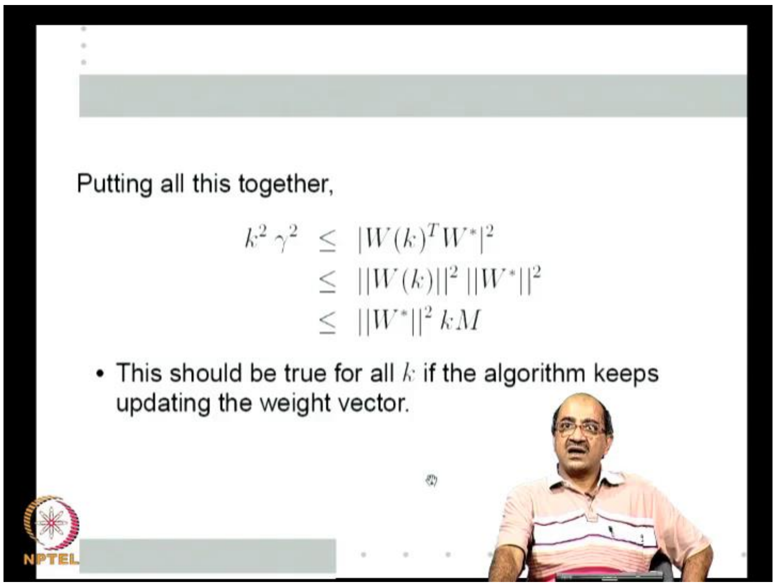
$$\exists W^*, \text{ such that } X_i^T w^* > 0, \forall i$$

- Let $\gamma = \min_i X_i^T W^*$. Note, $\gamma > 0$.
- Now we have

$$W(k)^T W^* = \sum_{i=0}^{k-1} X(i)^T W^* \geq k\gamma > 0$$

So, W(k) transpose w star is sum of X i transpose w star and we know each of this is at least gamma they are k of them. So, this is greater than equal to k gamma. So, we show that W(k) transpose w star is greater than equal to k gamma where once again this is gamma is some quantity greater than 0. So, this greater than 0 so it showed two things one is norm W square is norm W(k) square is bounded above by k times a constant and now W(k) transpose w star is bounded from the other side by k gamma.

(Refer Slide Time: 44:31)



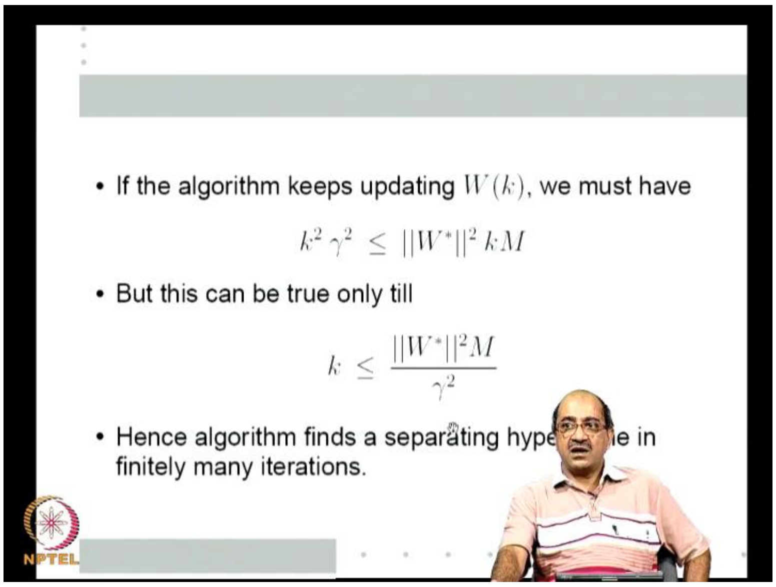
Putting all this together,

$$\begin{aligned} k^2\gamma^2 &\leq |W(k)^T W^*|^2 \\ &\leq ||W(k)||^2\,||W^*||^2 \\ &\leq ||W^*||^2\,kM \end{aligned}$$

- This should be true for all k if the algorithm keeps updating the weight vector.

Let us put all these together because this k gamma is positive we also know k square gamma square is less than or equal to this whole square. So, first we have k square gamma square less than equal to W(k) transpose W star Whole Square. This is an inner product as you know the inner product between two vectors is bounded above by these squares of the individual norm this so called Cauchy’s horizon equality. So, W(k) transpose w star whole square is bounded above by W(k) norm square plus W star norm square and then we know W(k) norm square is bounded above by k times M. So, this is W star norm square into k times M. So, if these iterations keep going on that is if misclassifications of the feature vector keep going on right then this has to be satisfied. For all k so, this should be true for all k if the algorithm keeps updating the weight vector without any limit.

(Refer Slide Time: 45:39)



- If the algorithm keeps updating $W(k)$, we must have

$$k^2\,\gamma^2 \;\le\; ||W^*||^2\,kM$$

- But this can be true only till

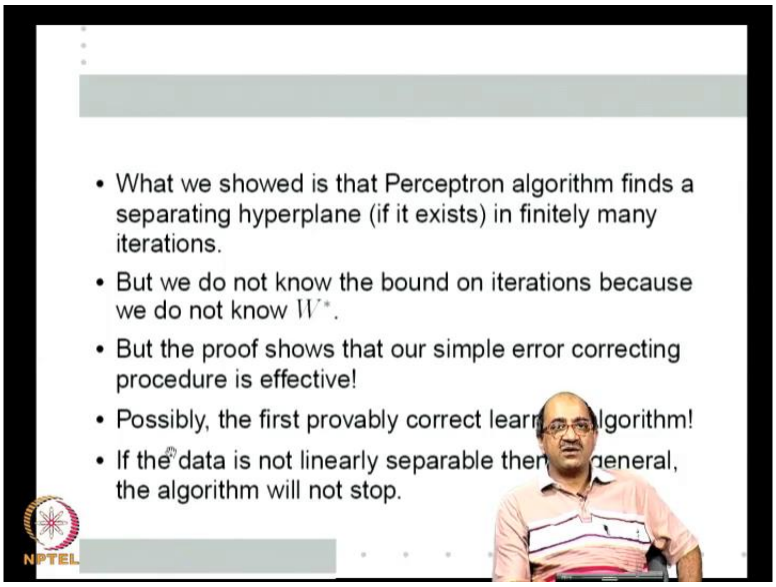
$$k \;\le\; \frac{||W^*||^2 M}{\gamma^2}$$

- Hence algorithm finds a separating hype[illegible]e in finitely many iterations.

So, what we have showed so far is if the algorithm keeps updating W(k) without an upper limit on k, then I should have the satisfy, but this I cannot satisfy because the left hand side is growing as k square. Right hand side is growing only as k so irrespective of the values of gamma M and W star k must come when this k no long will be satisfied. As a matter of fact I know what dividing by k we know that this can be true only till k less than some quantity W star whole square M by gamma square, which means the corrections cannot go on forever. After this k the corrections have to stop if the patterns are separable.

Hence, the algorithm finds a separating hyperplane in finitely many iteration, it is a remarkable result. We showed that this very simple minded algorithm which just does some local corrections of rotating the hyperplane clockwise or counter clockwise around the origin. Just to locally take care of the current sample will stop after finitely many iterations with and if it stops it has to be a separating hyperplane. If the training data is linearly separable, of course the training data is linearly separable it is given by a being a w star such that this gamma exists.

(Refer Slide Time: 47:29)

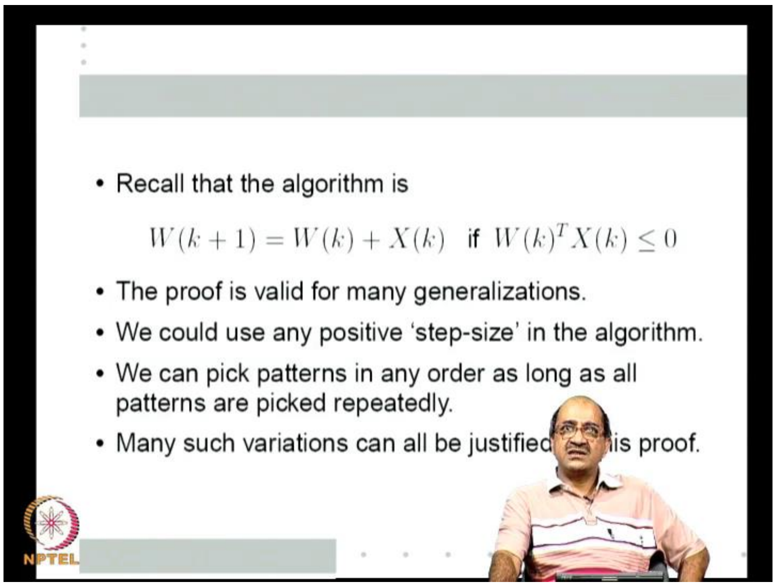


- What we showed is that Perceptron algorithm finds a separating hyperplane (if it exists) in finitely many iterations.
- But we do not know the bound on iterations because we do not know W^*.
- But the proof shows that our simple error correcting procedure is effective!
- Possibly, the first provably correct learn algorithm!
- If the data is not linearly separable then general, the algorithm will not stop.

So, what we have showed is that the perceptron algorithm finds a separating hyperplane in finitely many iterations. Finitely many because k cannot go beyond this number, this number is some finite number. M is the upper bound on the norm of all the X i W star is the separating hyperplane and gamma is the minimum W star transpose X i over all i which must be positive because w star is a separating hyperplane. So, what we showed is that perceptron algorithm finds a separating hyperplane if it exists in finitely many iterations, but of course we do not know the bound on the iterations. To know the bound on the iterations we need to know W star as well as gamma M we know because we know X i's, but we neither know W star because we do not know gamma.

So, while the proofs tells me that it has to stop infinitely many iterations I cannot calculate the bound which means in a particular case I do not know how many iterations to run the algorithm. Only thing I am guaranteed is if the training data is linearly separable and if I run the algorithm for enough time I will always converge with a separating hyperplane. It is better the fact that I cannot calculate the bound the proof shows that our simple error correcting procedure is effective which is remarkable this is arguably the first provably correct learning algorithm you know 1962. If the data is not linearly separable then in general the algorithm will not stop that means it will go into an infinite loop. So, if the data is linearly separable I know sometime or the other it will stop and give me a separating hyperplane. But if the data is not linearly separable then I cannot even guarantee that the algorithm stops.

(Refer Slide Time: 48:51)



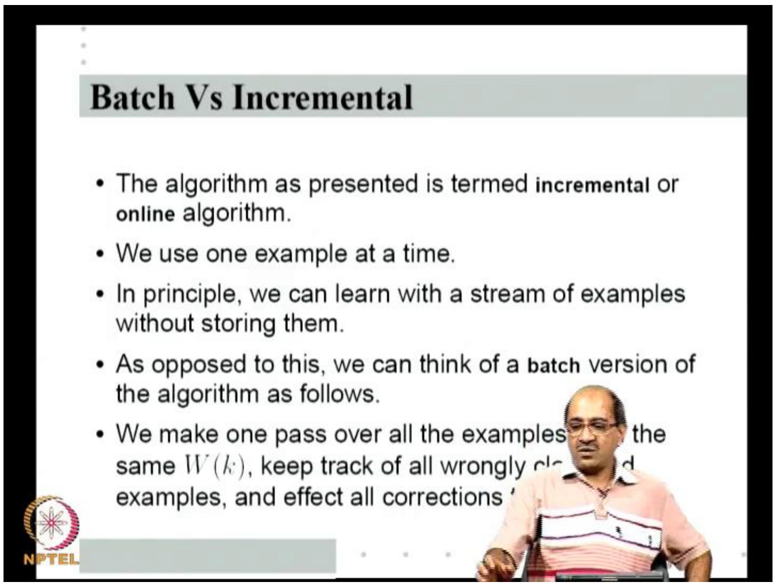
- Recall that the algorithm is

$$W(k+1) = W(k) + X(k) \quad \text{if} \quad W(k)^T X(k) \leq 0$$

- The proof is valid for many generalizations.
- We could use any positive 'step-size' in the algorithm.
- We can pick patterns in any order as long as all patterns are picked repeatedly.
- Many such variations can all be justified [illegible]is proof.

This is the algorithm under the notation that all the y is equal to 0 patterns set multiplied by minus 1 the proof can be the proof that we did also takes care of many generalisations for example, I am allowed to put a step-size here you finish up adding X k if I put eta times X k it is still all right because as long as eta is positive everything in the proof goes through. Also we did not really assume anything specific about how patterns are picked for concreteness we say patterns are picked one by one. But you can pick patterns in any which way. For example, in one pulse you can pick them from left to right another pulse you can pick them from right to left absolutely no problem. As long as you keep picking all patterns repeatedly we do not leave out anything so that when corrections stop we know that all the patterns are classified correctly as long as you keep picking all patterns repeatedly we can use any reasonable order of picking patterns.

(Refer Slide Time: 49:57)



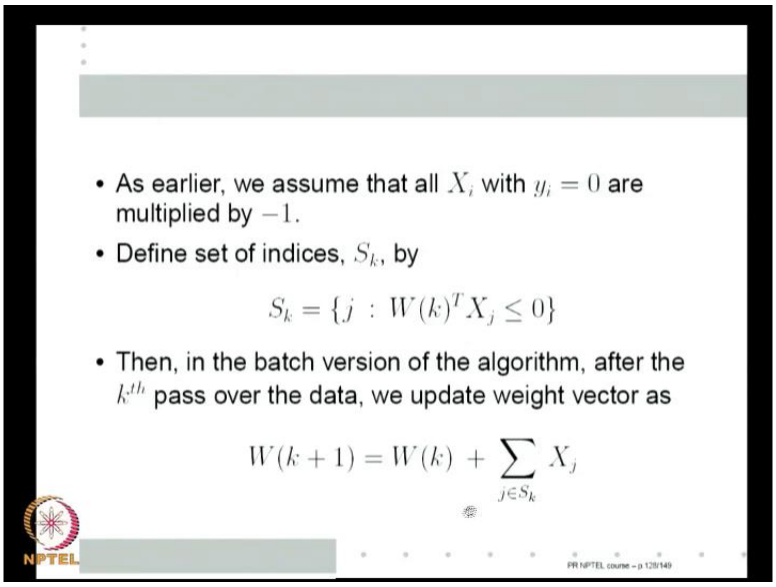
Batch Vs Incremental

- The algorithm as presented is termed **incremental** or **online** algorithm.
- We use one example at a time.
- In principle, we can learn with a stream of examples without storing them.
- As opposed to this, we can think of a **batch** version of the algorithm as follows.
- We make one pass over all the examples the same $W(k)$, keep track of all wrongly d examples, and effect all corrections

So, many such variations can be justified from the same proof, there are couple of other issues in the algorithm, we let us quickly go through them the algorithm as presented is termed incremental or online algorithm what does that mean. We essentially use one example at a time as I say picking. So, you give me one example so, I can ask somebody else to give me an example from the training set.

So, we use only one example at a time in principle I do not need to store all the example. If I have a stream of example coming and they are statistically such that if I keep doing these corrections sooner or later everything will be correctly classified or if the stream repeats itself. So in principle I can learn from a stream of examples without storing them that is what is meant by incremental. I do not need all the examples in the main memory for example, as I push to this we can have a batch version where I can keep all the examples with me and then do the corrections. For example, what I can do is I can make one pass all examples with the same W, k keep track of our all the wrongly classified examples and then effect all the corrections. So, now k will not be iterations like earlier but, now k keeps track of the passes over the data my first pass over the data I keep track of all the X’s that I incorrectly classified and then do only one short correction.

(Refer Slide Time: 51:13)



- As earlier, we assume that all X_i with $y_i = 0$ are multiplied by -1.
- Define set of indices, S_k, by

$$S_k = \{j \ : \ W(k)^T X_j \leq 0\}$$

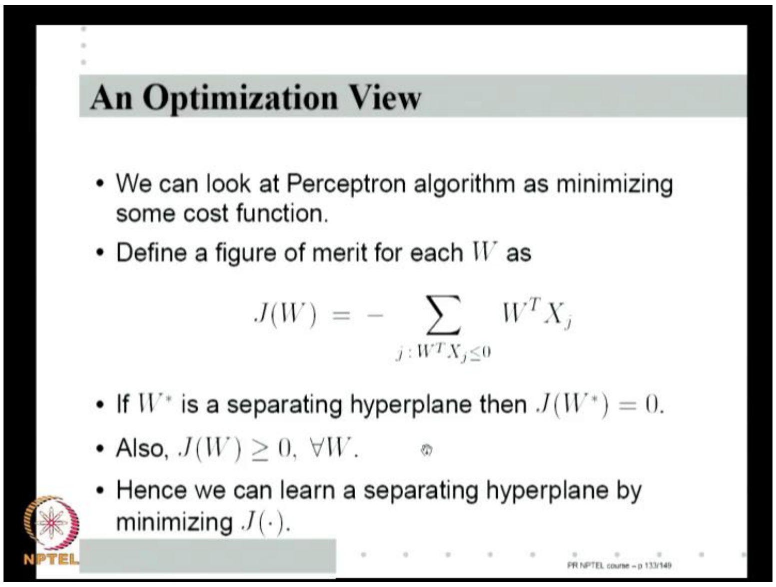
- Then, in the batch version of the algorithm, after the k^{th} pass over the data, we update weight vector as

$$W(k+1) = W(k) + \sum_{j \in S_k} X_j$$

So, for example, I can have it like this let us say as earlier we assumed that X i with y i is equal to 0 multiplied by minus 1. So, I essentially looking for a W's as the W transpose X i is greater than 0. So, at a given pass gate pass of the data W, k transpose X if W(k) transpose X j less than 0 put j in a set called S k. So, S k contains the indices of all the training samples which are incorrectly classified by the current W. Now, I can have a batch version which simply means after the k-th pass over the data W is updated by W(k plus 1) is W(k) plus sum of all the X j's over the set S k.

So, I had k 1 W(k) with which I classify all the patterns remember which are all the patterns which are wrongly classified add all those X j's to this W(k) this is called a batch version of the perceptron algorithm intuitively this should also work and we will show why this should work by looking at a different perspective. Earlier we looked at a perceptron as in error correcting thing locally correcting errors that is the incremental algorithm.

(Refer Slide Time: 52:26)



An Optimization View

- We can look at Perceptron algorithm as minimizing some cost function.
- Define a figure of merit for each W as

$$J(W) = - \sum_{j : W^T X_j \leq 0} W^T X_j$$

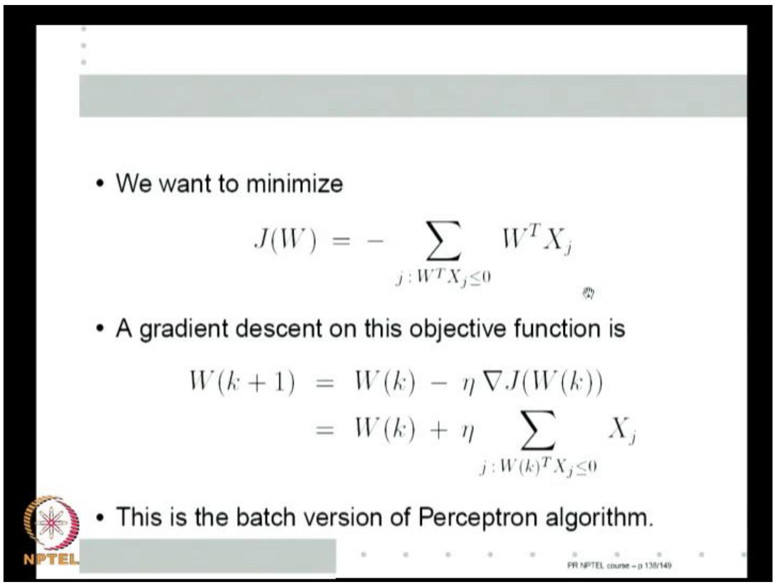
- If W^* is a separating hyperplane then $J(W^*) = 0$.
- Also, $J(W) \geq 0, \ \forall W$.
- Hence we can learn a separating hyperplane by minimizing $J(\cdot)$.

PR NPTEL course – p.133/149

We look at the batch algorithm as a simple optimization procedure as you have seen during the Baye's classifier. We can rate algorithms, we can rate different classifiers through some risk functions which are some figure of merit for the classifier and then minimize that figure of merit. so, for each W like that we can define a figure of merit j, let us call that figure of merit j, j of W is minus of W transpose X j where j goes over all W j is said that w transpose X j less than equal to 0.

So, given any W, I will find out what are all the X j's that it wrongly classifies and then add all those W transpose X j's and put a minus sign outside and that is the figure of merit for w. So, more things it wrongly classifies j is larger, but it does not tell me the number of wrongly classified because it actually adds this W transpose X j of all the wrongly classified ones. First let us notice that if W star is a separating hyperplane then for all X j W star transpose X j will be positive and the sum will be 0 there will be nothing in this sum so, j of W star will be 0. Secondly I am adding W transpose X j only when W transpose X j is less than equal to 0 and I put a minus sign here. So, j W is always greater than 0, for any W or this is separating hyperplane for all possible vectors W, j of W is greater than 0. So, I can simply minimize j of W to get the best W.

(Refer Slide Time: 54:01)



- We want to minimize

$$J(W) = -\sum_{j : W^T X_j \le 0} W^T X_j$$

- A gradient descent on this objective function is

$$\begin{aligned} W(k+1) &= W(k) - \eta \nabla J(W(k)) \\ &= W(k) + \eta \sum_{j : W(k)^T X_j \le 0} X_j \end{aligned}$$

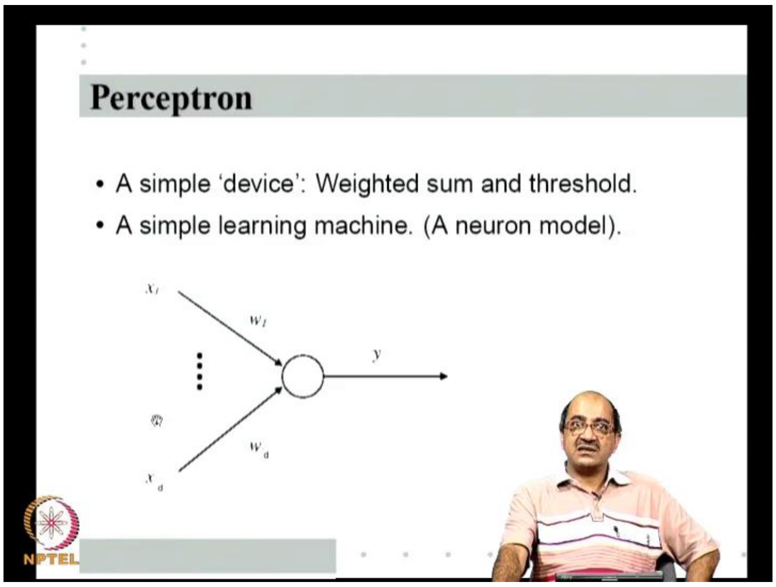
- This is the batch version of Perceptron algorithm.

PR NPTEL course – p.138/149

So, we can learn this separating hyperplane by minimizing j so, we want to minimize this function how do I minimize. I can use a simple gradient descent on the objective function so, what will be a gradient descent W(k plus 1) is W(k) minus some step size eta times gradient of this function j evaluated at the current value W k, this is the standard gradient descent algorithm. what is the gradient of this with respect to W is simply summation of X j's so, that will be W(k) plus eta times summation of X j all j's so, that W(k) transpose X j are less than 0. So, this is same as j belonging to S k as we called earlier so, this becomes plus because of this minus sign in my definition of j which means a gradient descent on this on this criterion function is the batch version of perceptron.

So, this criterion function is called the perceptron criterion function and the perceptron algorithm the batch version can be seen to be minimizing this criterion. So, perceptron algorithm can also be viewed as simply a gradient descent on a nice reasonable criterion function and hence it should perform well. We can either look at it as locally correcting or we can look at it as minimizing a well defined criterion function ok.

(Refer Slide Time: 55:27)

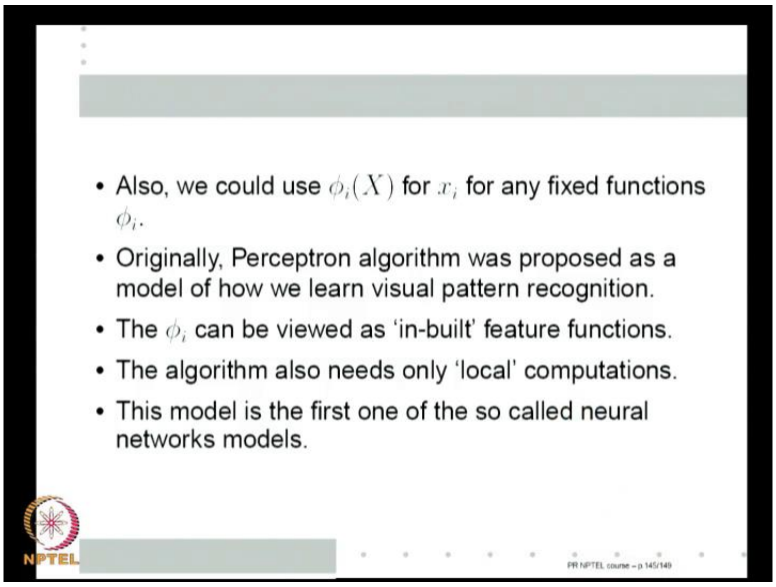


Perceptron

- A simple 'device': Weighted sum and threshold.
- A simple learning machine. (A neuron model).

So, let us sum up the perceptron is a very interesting device it is a simple weighted sum and threshold and it is also a learning machine it is a neuron model. Most of our brains computing elements are called neurons and at a simple level certainly at the level neurons are understood in 60s are a very good model for a neuron. A neuron simply as many inputs and it takes a weighted sum of its inputs and thresholds it at some point to decide whether it is going to have a 1 output or a 0 output.

(Refer Slide Time: 56:08)



- Also, we could use $\phi_i(X)$ for x_i for any fixed functions ϕ_i.
- Originally, Perceptron algorithm was proposed as a model of how we learn visual pattern recognition.
- The ϕ_i can be viewed as 'in-built' feature functions.
- The algorithm also needs only 'local' computations.
- This model is the first one of the so called neural networks models.

PR NPTEL course – p.145/149

Now, as a matter of fact as we have already mentioned these X’s can be phi i of X, there can be any fixed functions. Originally the perceptron algorithm is proposed as a model of how we learn visual pattern recognition. So, the way we can think of it is that, this is actually a phi of X so, in front of our retina there might be some visual pattern and we are inbuilt with some feature detectors. So, that d such feature detectors let us say given a particular pattern each of them will have some values we have built with those feature detectors, the feature detectors not built assuming that we need to differentiate between ambassador cars and maruti cars and differentiate between you know dogs and cats or whatever.

So, whatever we have built, born with or inbuilt so, given that we have some fixed feature functions which are inbuilt. If I give an examples then the idea is can I learn, can my neurons learn the weights in the weighted sum so, that I can learn to categorize that is what the perceptron algorithm does. So, originally a perceptron algorithm is proposed as a model of how we learn visual pattern recognition the phi’s can be viewed as inbuilt feature functions the and the other thing is the algorithm only needs local computations because to update W, I add x which means to update this weight I only need to know this value.

So, it is locally available so, even though there might be many connections here each weight can update itself by locally knowing the value here that is also a an interesting this thing. so, the model is first one of this so called neural network models for learning. So, we will stop there for the perceptron algorithm next class we will see how to go beyond perceptron to be able to take care of cases where data is not linearly separable. We will start by what I have just done we look at the perceptron algorithm again just the final bit of perceptron and then go beyond that.

Thank you.