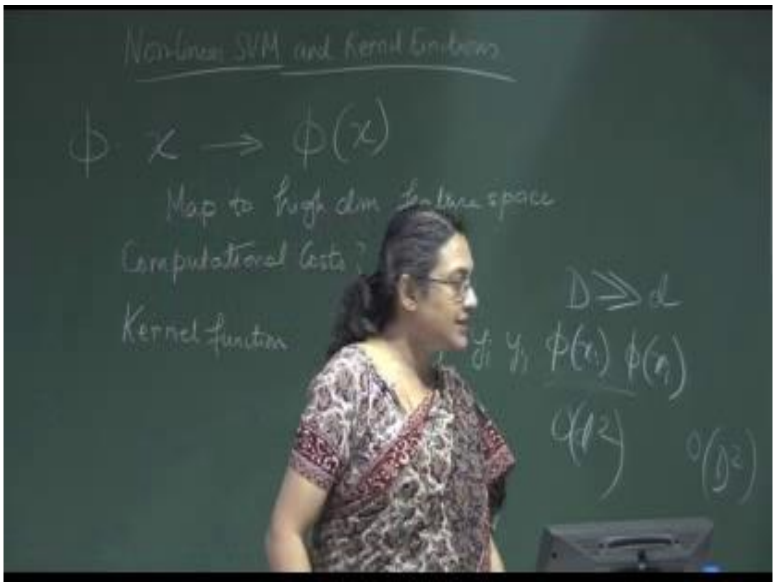
Introduction to Machine Learning
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 24
Nonlinear SVM and Kernel Function

Good morning. Today, we will look at non-linear SVM and kernel functions in order to work with non-linear SVM. In the last class, we have looked at linear SVM and we have also seen how to deal with noise in linear SVM. So, we can when the data sets are actually linearly separable, but there is some amount of noise in the boundary, we can approximate by a linear decision surface and allow soft merging by incorporating the slack variables, but what is the data set is truly non-linearly separable.

(Refer Slide Time: 00:31)

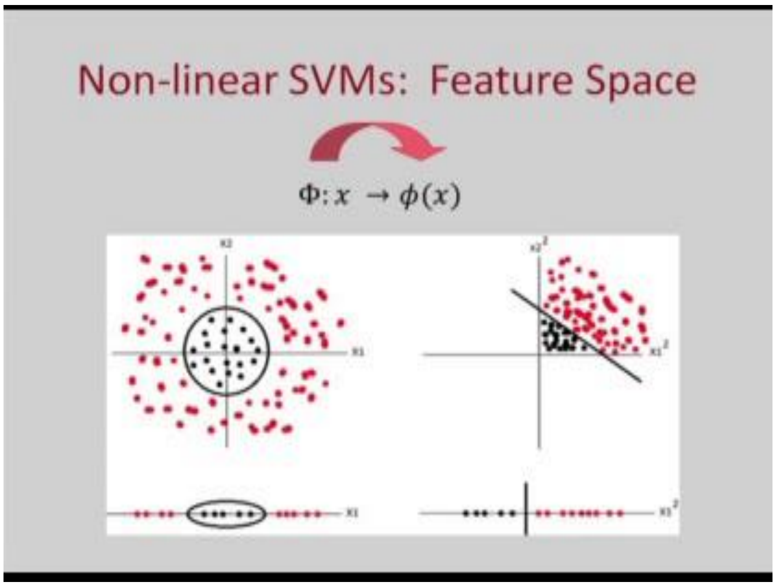

Nonlinear SVM and Kernel Functions
$\phi \; x \rightarrow \phi(x)$
Map to high dim feature space
Computational Costs?
Kernel function
$D \gg d$
$y_i \, \phi(x_i) \, \phi(x_j)$
$O(d^2)$
$O(D^2)$

What we can do is that we can use, suppose x is our input features. We can use mapping to map x to phi x, the original features space or original attributes space. We can transform into a new feature space and in some cases it is possible that the training points are linearly separable in the transformed feature space. So, usually we will be mapping data to a higher dimensional feature space and in some cases it is found that when you map the data into an appropriate higher dimensional feature space, in some case dimension maybe where infinite points become linearly separable. Now, if we do use a big set of features then what about computational costs.

We have seen earlier that while solving the SVM, we need to find sigma alpha i alpha j y i y j x i dot x j. So, we need to find the pair wise. If have m training examples, we need to do m square computations like this and each time we need to find a dot product of x i dot x j. Now, if these vectors have dimensionality D then finding the dot product takes computation time O D square. So, total m square times D square time is taken. Now, if we transform it to a higher dimensional space this x i x j will become phi x i dot phi x j right and if this dimensionality D much greater than small d then this time taken will be o capital d square which will be quite large.

So, normally if we transform this feature to a higher dimensional feature space. It will lead to higher computational cost, but we will see that by using kernel function. We can achieve this transformation without any major implication on computational cost.

(Refer Slide Time: 04:28)

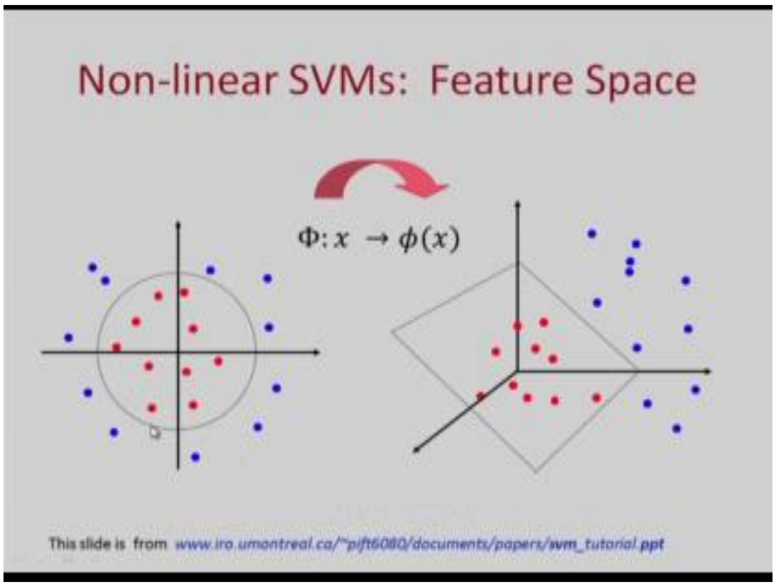

Non-linear SVMs: Feature Space

$\Phi: x \rightarrow \phi(x)$

So, first let us see why we would we able to convert to a different feature space where the points are linearly separable. Let us look at the first picture; here we have the black points in a circle and the red points outside the circle. Now, these points in the x 1 x 2 with x 1 x 2 dimensions they are not separable by a linear classifier. However, if we go to a different feature space where the axes are x 1 square and x 2 square, we can plot this black and red points here again and in this case we find that they can be separated linearly in the transformed feature space x 1 square x 2 square.

A simpler example let us look at this point when there is a single feature x 1, we have this red point and the black point and clearly they are not linearly separable. Now, we may convert this to a point x 1 square and these points may become linearly separable, suppose this is the origin. So, this is the origin and these are the points are those points which are you know, let us say this is point 3 and this is minus 3. Those within minus 3 and plus 3 are linearly separable. Now, when we transform this to this space then these points are will lie between 0 and 9 and these points beyond 9. So, they will become linearly separable.

(Refer Slide Time: 06:18)



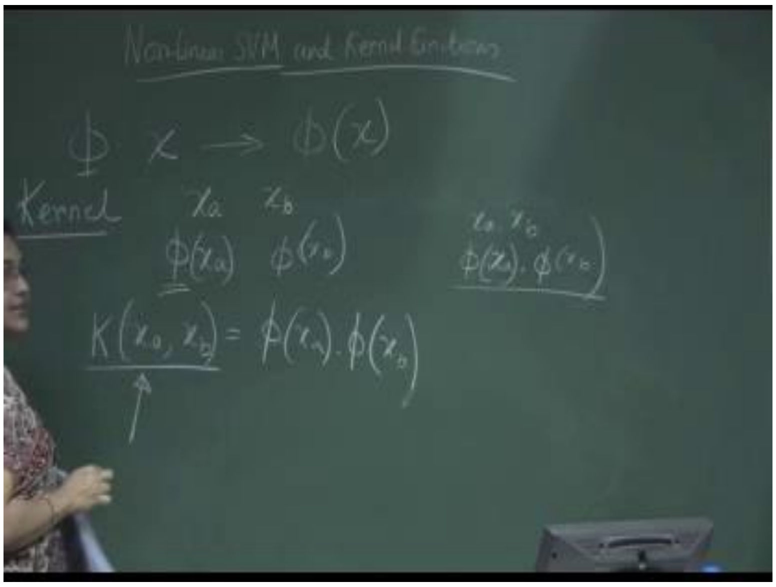
Non-linear SVMs: Feature Space

This slide is from www.iro.umontreal.ca/~pift6080/documents/papers/svm_tutorial.ppt

It is possible as we saw in this picture that in certain cases when you convert the feature space to an appropriate high dimensional feature space, the points may become linearly separable. Here is 1 more example, here we have this red and blue points after transformation we have we transform it. So, originally we have 2 features, now we have 3 features and with these 3 features we are able to separate these points in the new feature space. So, this brings us, this is one aspect that points which are not separable linearly in up in the original feature space may become separable in the non-linear feature space.

The second thing that we will look at is there are certain cases, certain types of feature transformations where after transformation the SVM can be solved efficiently. This brings us to a notion of kernel functions or kernels.

(Refer Slide Time: 07:13)

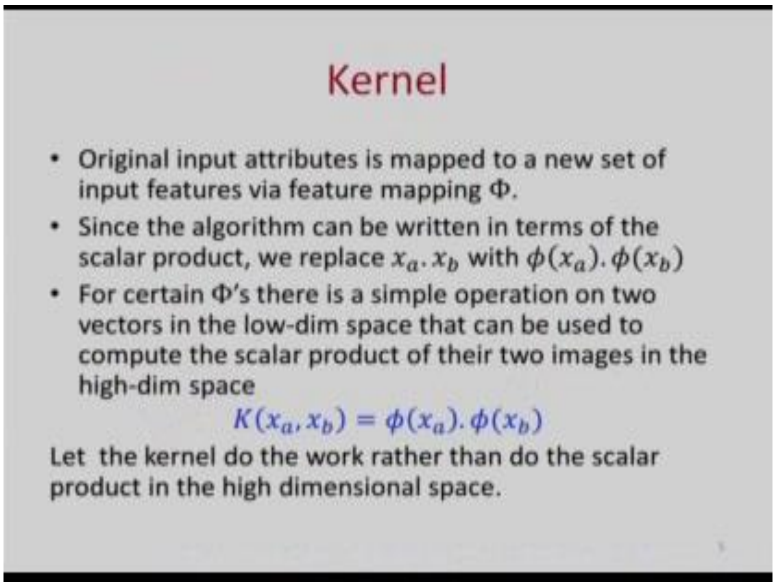

Non-linear SVM and Kernel functions
$\phi \quad x \rightarrow \phi(x)$
Kernel
$x_a \quad x_b$
$\phi(x_a) \quad \phi(x_b)$
$x_a \cdot x_b$
$\phi(x_a) \cdot \phi(x_b)$
$K(x_a, x_b) = \phi(x_a) \cdot \phi(x_b)$

Now, we have originally, suppose 2 points x a and x b and by using this transformation phi they now become the new features become new inputs become phi x a and phi x b. Now, as we saw in the solution of SVM, we need to find x a dot x b and when we use these phi we will have to solve phi x a dot phi x b.

Now, there are certain phi's, there are certain phi's not for all functions. So, that corresponding to them we have something like this that phi x a dot phi x b is a function of x a and x b and this function of x a and x b in certain cases can be computed without expanding phi x a and phi x b. So, this particular function k x a x b is a function of x a x b and the value of this is same as the dot product of phi x a and pi x b, but this function may be easy to compute and more specifically, we may be able to compute this function without expanding this phi's individual phi's. So, such functions are called kernel function.

So, if you use kernel functions we can use the kernel to do the work. So, in the solution of SVM where we have to find phi x a dot phi x b, normally we will for every example we will expand phi x a expand phi x b take the dot product, but for kernel functions we can simply use this function k x a x b. So, for certain phi's there is a simple operation k by which we can find is dot product of phi in a simple times. So, this is called the kernel trick.

(Refer Slide Time: 09:38)



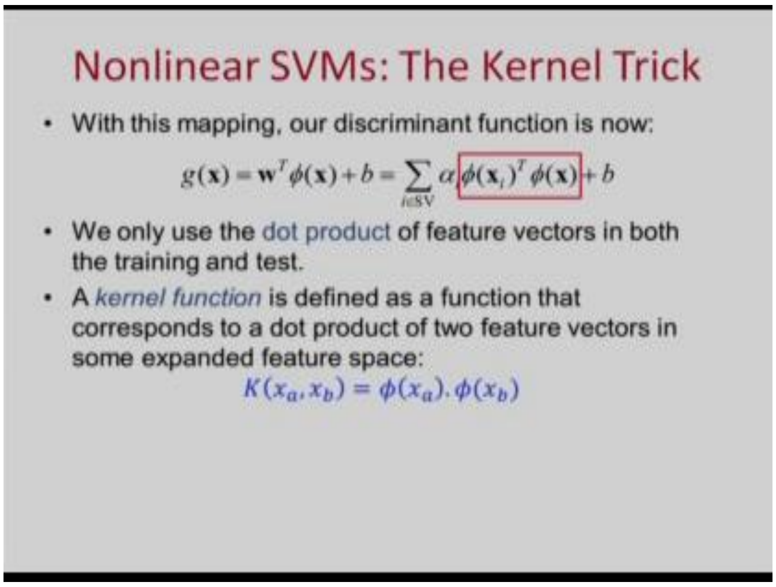
Kernel

- Original input attributes is mapped to a new set of input features via feature mapping Φ.
- Since the algorithm can be written in terms of the scalar product, we replace $x_a . x_b$ with $\phi(x_a) . \phi(x_b)$
- For certain Φ's there is a simple operation on two vectors in the low-dim space that can be used to compute the scalar product of their two images in the high-dim space

$$K(x_a, x_b) = \phi(x_a) . \phi(x_b)$$

Let the kernel do the work rather than do the scalar product in the high dimensional space.

(Refer Slide Time: 09:41)



Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in SV} \alpha_i \boxed{\phi(\mathbf{x}_i)^T \phi(\mathbf{x})} + b$$

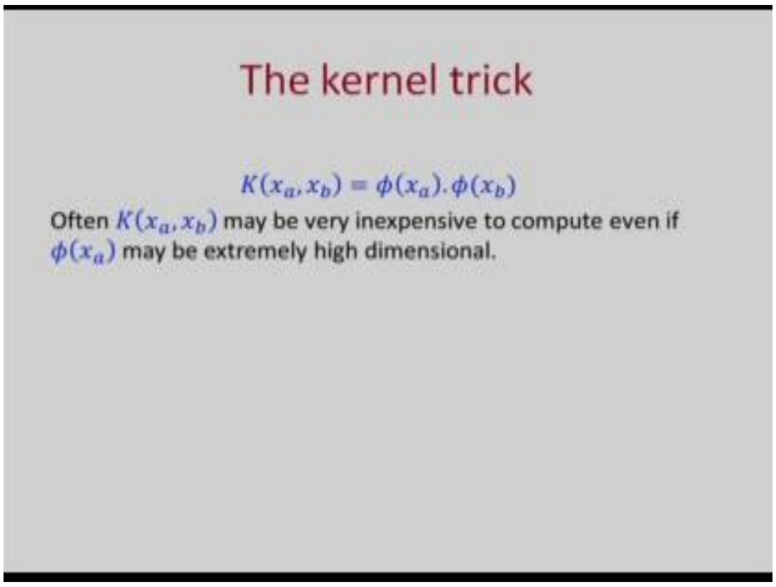
- We only use the dot product of feature vectors in both the training and test.
- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(x_a, x_b) = \phi(x_a).\phi(x_b)$$

And when we apply the kernel trick as we saw that when we use an SVM to classify a test point x, we apply w dot phi x plus b which is equal to alpha i phi x i dot phi x plus b where i corresponds to the support vectors for which alpha i is non-zero.

Now, if we are using the kernel trick this phi x i dot phi x, which we have to compute can be computed as kernel function of x i and x. So, we have seen that in the formulation of SVM and the solution of the dual problem the dot product we are using the feature vectors both in training as well as test only as the dot product or scalar product and we have also seen that a kernel function is function which corresponds to the dot product of 2 feature vectors in some expanded feature space. So, if you are using the kernel function what we can do is that instead of phi x i dot phi x we can have k x i x and this will make the computation simpler.

(Refer Slide Time: 10:59)



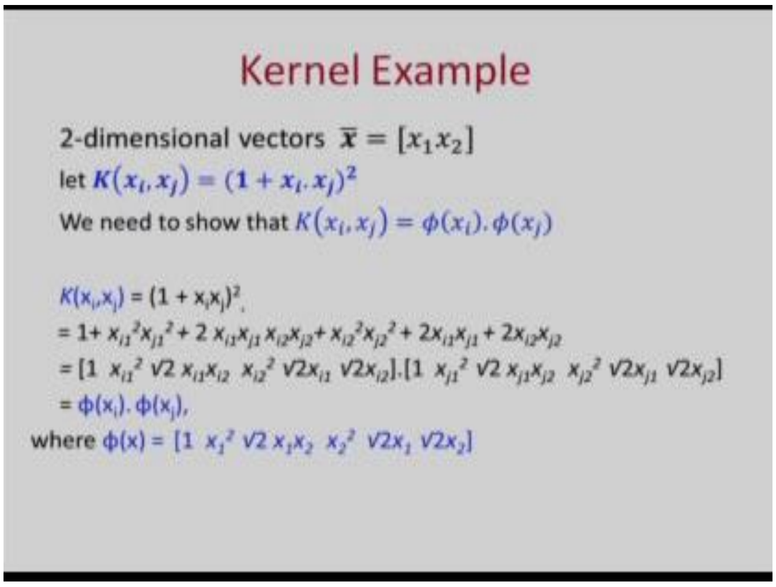
The kernel trick

$$K(x_a, x_b) = \phi(x_a).\phi(x_b)$$

Often $K(x_a, x_b)$ may be very inexpensive to compute even if $\phi(x_a)$ may be extremely high dimensional.

So, this k x i x may be in expensive to compute even phi x may be extremely high dimensional.

(Refer Slide Time: 11:05)



Kernel Example

2-dimensional vectors $\overline{\mathbf{x}} = [x_1 x_2]$

let $\boldsymbol{K(x_i, x_j) = (1 + x_i . x_j)^2}$

We need to show that $K(x_i, x_j) = \phi(x_i).\phi(x_j)$

$K(x_i,x_j) = (1 + x_i x_j)^2,$

$= 1 + x_{i1}^2 x_{j1}^2 + 2\, x_{i1} x_{j1}\, x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1} x_{j1} + 2x_{i2} x_{j2}$

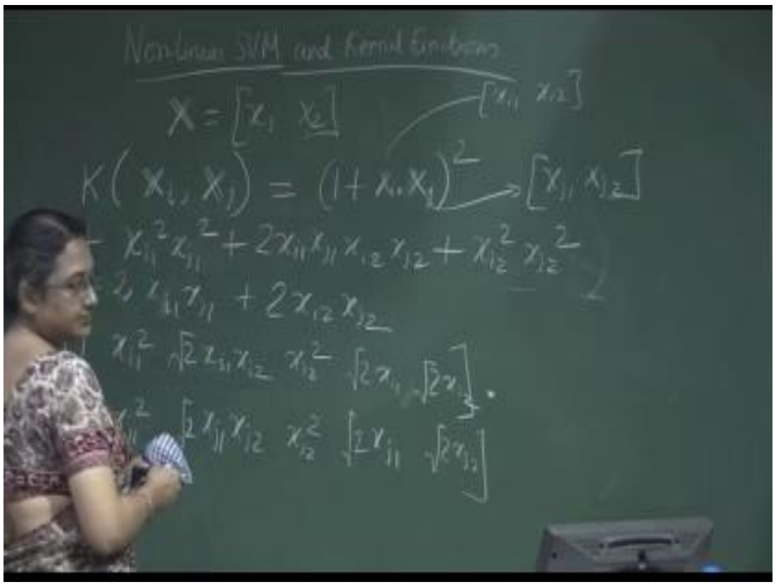
$= [1\;\; x_{i1}^2\;\; \sqrt{2}\, x_{i1} x_{i2}\;\; x_{i2}^2\;\; \sqrt{2} x_{i1}\;\; \sqrt{2} x_{i2}].[1\;\; x_{j1}^2\;\; \sqrt{2}\, x_{j1} x_{j2}\;\; x_{j2}^2\;\; \sqrt{2} x_{j1}\;\; \sqrt{2} x_{j2}]$

$= \phi(x_i).\phi(x_j),$

where $\phi(x) = [1\;\; x_1^2\;\; \sqrt{2}\, x_1 x_2\;\; x_2^2\;\; \sqrt{2} x_1\;\; \sqrt{2} x_2]$

Let us now look at an example of a kernel function. So, let us assume a 2-dimensional vector x which has 2 components x 1 and x 2; x 1 and x 2 are the 2 attributes of this vector x.

(Refer Slide Time: 11:25)


Non-linear SVM and Kernel functions
$X = [x_1 \; x_2]$
$[x_{i1} \; x_{i2}]$
$K(X_i, X_j) = (1 + X_i . X_j)^2 \rightarrow [x_{j1} \; x_{j2}]$
$x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2$
$2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$
$x_{i1}^2 \; \sqrt{2} x_{i1} x_{i2} \; x_{i2}^2 \; \sqrt{2} x_{i1} \; \sqrt{2} x_{i2}] .$
$x_{j1}^2 \; \sqrt{2} x_{j1} x_{j2} \; x_{j2}^2 \; \sqrt{2} x_{j1} \; \sqrt{2} x_{j2}]$

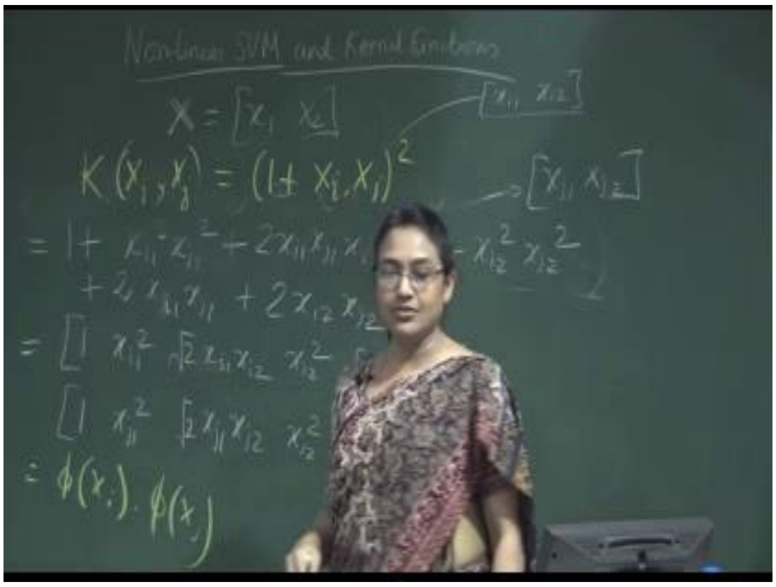
Now, let us say k of x i x j, let us take k x i x j equal to 1 plus x i dot x j whole square. So, k of x i x j x i x j are 2 vectors corresponding to the attributes in the original attribute space k x i x j is equal to now this is the kernel function which is 1 plus x i dot x j whole square.

Now, this function is easy to compute. In order to compute this function you take the dot product of the original attribute space, add 1 and take the square. Now, we will show that this k x i x j corresponds to a kernel function. Now, we can expand k x i x j 2 1 plus x i x j whole square. So, x i has 2 components x i 1 x i 2. Similarly, x j has 2 component x j 1 x j 2. Now, we can expand this function as 1 plus x i 1 square x j 1 square plus 2 times x i 1 x j 1 x j 2 x i 2 x j 2 plus we have x i 2 square x j 2 square plus 2 x i 1 x j 1 plus 2 times x i 2 x j 2.

Now, this algebraic expression can be rewritten as the product of 1 plus x i 1 square or rather it can be rewritten as the dot product of 2 vectors which you can easily verify 1 x i 1 square root 2 x i 1 x i 2 and x i 2 square and root 2 x i 1 root 2 x i 2 dot product, with let me write in the line below 1 the next component is x j 1 square next component is root over 2 x j 1 x j 2 then x j 2 square then next component is root over to x j 1 root over 2 x j two. So, each of this is 3, 4, 5, 6 each of this is a 6-dimensional vector.

So, this expression can be written as a dot product of these 2 vectors and you can easily verify by expanding this. So, k xi x j which is 1 plus x i dot x j whole square can be written as a dot product of these 2 vectors.

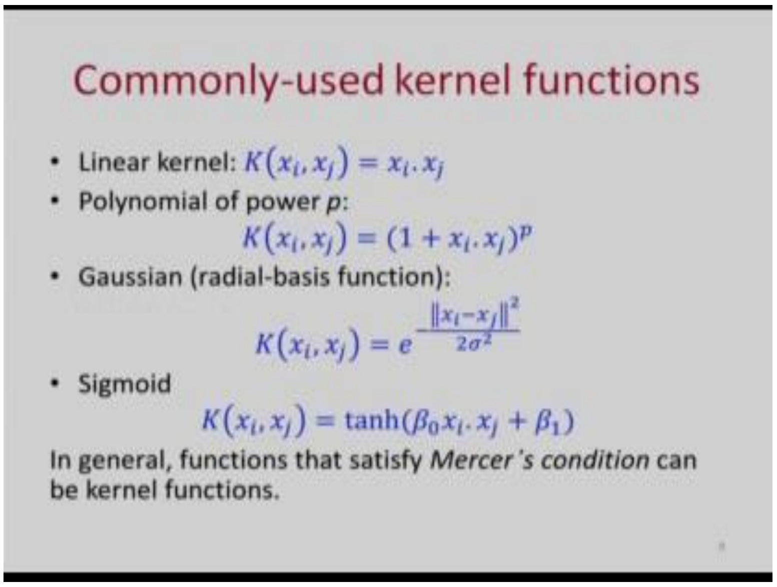
(Refer Slide Time: 15:48)


Non-linear SVM and Kernel functions
$X = [x_1 \ x_2]$
$[x_{i1} \ x_{i2}]$
$K(x_i, x_j) = (1 + x_i . x_j)^2$
$[x_{j1} \ x_{j2}]$
$= \phi(x_i) . \phi(x_j)$

So, we can write that k x 1 x 2 equal to 1 plus x i dot x j whole square instead of 1, 2. Let me write x i x j k x i x j equal to 1 plus x i x j whole square equal to dot product of this and this can be written as phi of x i dot phi of x j right. So, if we transform the x i to this new space where phi x i transforms to 1 x i 1 square root 2 x i 1 x i 2 x i 2 square root 2 x i 1 root 2 x i 2 instead of the original vector which is x i 1 x i 2 then the dot product in this space is equal to k x i x j equal to 1 plus x i dot x i whole square.

So, corresponding to this transformation phi the kernel function can be very easily computed like this without expanding this particular feature. So, this is an example of a kernel function. There are many other commonly used kernel functions, some of them we can see in this slide.

(Refer Slide Time: 17:22)



Commonly-used kernel functions

- Linear kernel: $K(x_i, x_j) = x_i . x_j$
- Polynomial of power p:
$$K(x_i, x_j) = (1 + x_i . x_j)^p$$
- Gaussian (radial-basis function):
$$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$
- Sigmoid
$$K(x_i, x_j) = \tanh(\beta_0 x_i . x_j + \beta_1)$$

In general, functions that satisfy *Mercer's condition* can be kernel functions.

So, the linear SVM corresponds to linear kernel where phi x i equal to x i phi x j equal to x j that is the identity mapping. So, k of x i x j equal to x i dot x j called linear kernel for general polynomial kernel k x i x j is 1 plus x i dot x j to the power p. We saw the example where of quadratic kernel where k x i x j is 1 plus x i x j whole square.

But this time we generalised to polynomial kernel, where we have 1 plus x i x j to the power p; this is the polynomial kernel of power p. Then we have the Gaussian kernel or the kernel corresponding to the radial basis function where we have k x i comma x j equal to exponential of minus x i minus x j whole square by 2 times sigma square. So, this is the Gaussian kernel.

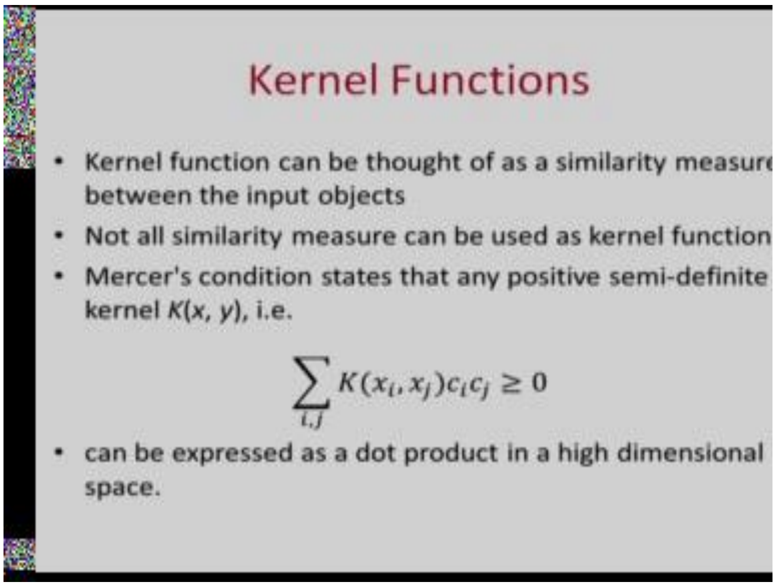
(Refer Slide Time: 18:18)

Non-linear SVM and Kernel Functions
$X = [x_1 \; x_2]$
$K_1(x_i, x_j) = (1 + x_i . x_j)^p$
$K_2(x_i, x_j) = \exp$
$K_3(x_i, x_j) = \tanh$

Another popular kernel is the sigmoid kernel which is given by k x i x j equal to 10 hyperbolic of parameter beta 0 x i dot x j plus beta 1. So, these are some examples of kernel functions which are quite popular. There can be kernel functions; measures some similarity between instances x i and x j and this kernel functions you know we can have define many types of kernel functions depending on the domain and certain notions of similarity.

For example, you can use kernel function for text classification based on the similarity of the words in the text and so on. In general functions that qualify as kernel functions satisfy Mercer's condition; those functions that satisfy Mercer's condition are called kernel functions.

(Refer Slide Time: 20:03)



Kernel Functions

- Kernel function can be thought of as a similarity measur
  between the input objects
- Not all similarity measure can be used as kernel function
- Mercer's condition states that any positive semi-definite
  kernel $K(x, y)$, i.e.

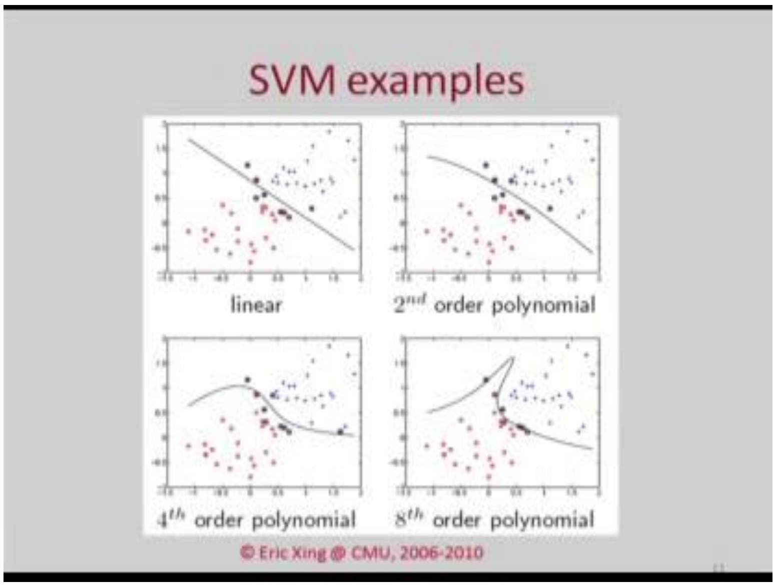
$$\sum_{i,j} K(x_i, x_j)c_i c_j \geq 0$$

- can be expressed as a dot product in a high dimensional
  space.

We will not go into a lot of detail here, but we can say that kernel functions are some similarity measures and this similarity as usually symmetric that is k x i x j is equal to k x j x i, but all similarities are not kernel functions. Mercer's conditions states that if you can write the similarities between the points as a matrix and this matrix is symmetric positive semi definite then a kernel function will exist or in general mercers conditions states that any positive semi definite kernel k x y for which sigma over i j k of x i x j times c i c j is greater than or equal to 0 for any real number c i c j, they can be considered as kernel function because such functions can be expressed as a dot product in high dimensional space.

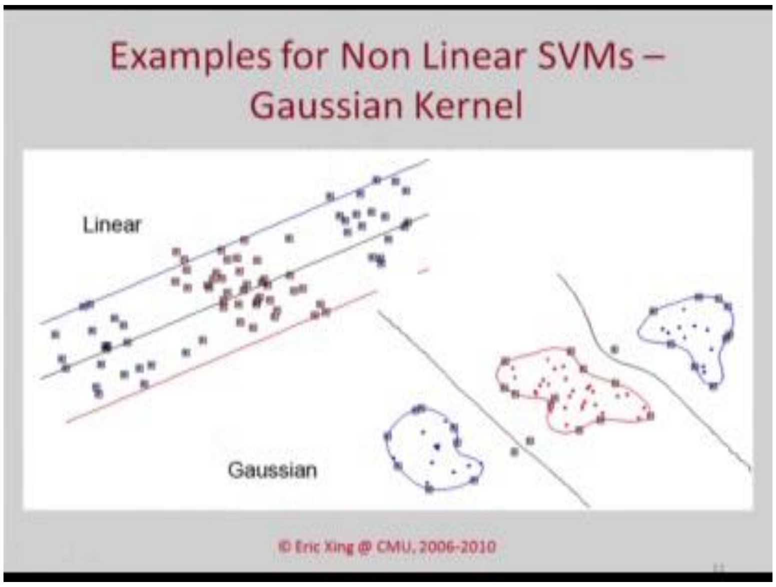
We will not going to details of this, but this gives us a general characterisation of kernel functions and we can also compose more than 1 kernel function to get new kernel function, but we will not make further discussion into this.

(Refer Slide Time: 21:28)



But I will show you a certain cases where the decision surface is linear or non-linear and the kernel functions which are appropriate in those cases. If we look at this picture for this case linear kernel with noise will be quite appropriate whereas, in this case the decision surface is actually do not have good linear decision surface. You have a quadratic decision surface and the second order polynomial a quadratic kernel will be appropriate. In this case you have a fourth order polynomial which gives a good decision surface. Here, eighth order polynomial and these examples can be used with kernels of order 8 polynomial, order 4 polynomial and so on.

(Refer Slide Time: 22:22)

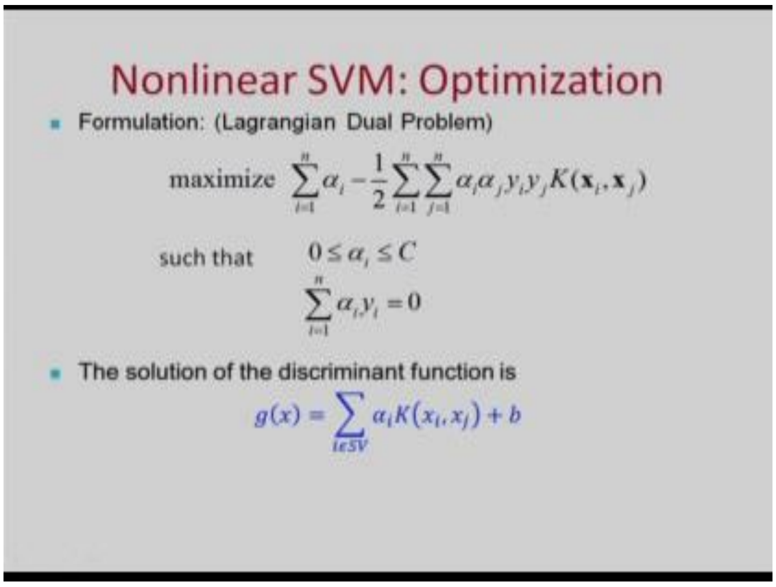


Examples for Non Linear SVMs – Gaussian Kernel

© Eric Xing @ CMU, 2006-2010

This picture shows points this red and blue are the 2 classes and we are not separable by linear kernel. However, you can use Gaussian kernels to separate them. Gaussian kernel can be used to separate points where these points can lie within Gaussian of each other now. So, here Gaussian kernels will be appropriate.

(Refer Slide Time: 22:53)



Nonlinear SVM: Optimization

- Formulation: (Lagrangian Dual Problem)

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{such that} \quad 0 \le \alpha_i \le C$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

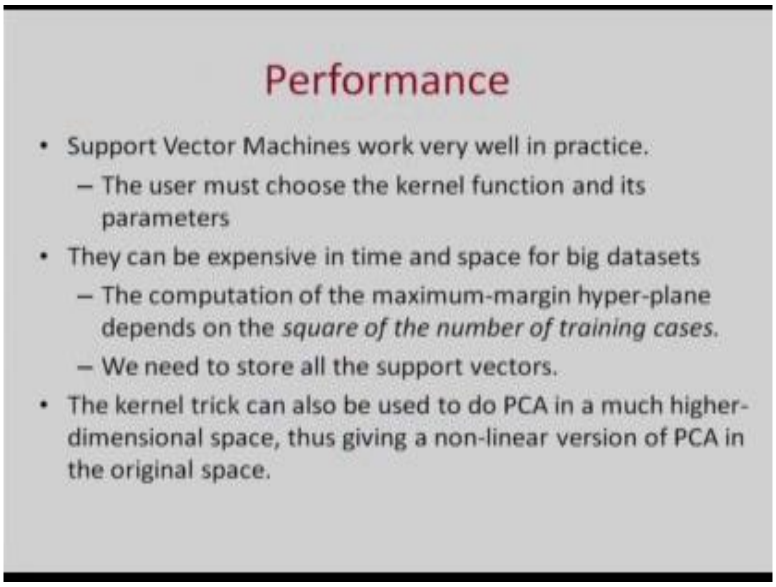
- The solution of the discriminant function is

$$g(x) = \sum_{i \in SV} \alpha_i K(x_i, x_j) + b$$

Now, let us see that once we have this kernel function, how we can solve the SVM. So, the formulation is very similar. We have phi x i and phi x j, it amounts to maximising. In the dual formulation, the solution will amount to maximising sigma alpha i minus half summation over i j alpha i alpha j y i y j phi x i dot phi x j, which can be written as k of x i x j and the conditions are same alpha i lies between 0 and c sigma alpha i y i equal to 0 and after you have solved for this alphas, you get a solution and when you get a test point x.

The classification of the test point can be found by g x equal to sigma i ranges over the support vectors alpha i phi x i dot x plus b which becomes k of x i x plus b. So, for all points k of x i x plus b can be used to get the solution and if k is easy to compute then the solution computation cost of solution in the training face as well as the testing face is not much different from the linear SVM.

(Refer Slide Time: 24:24)



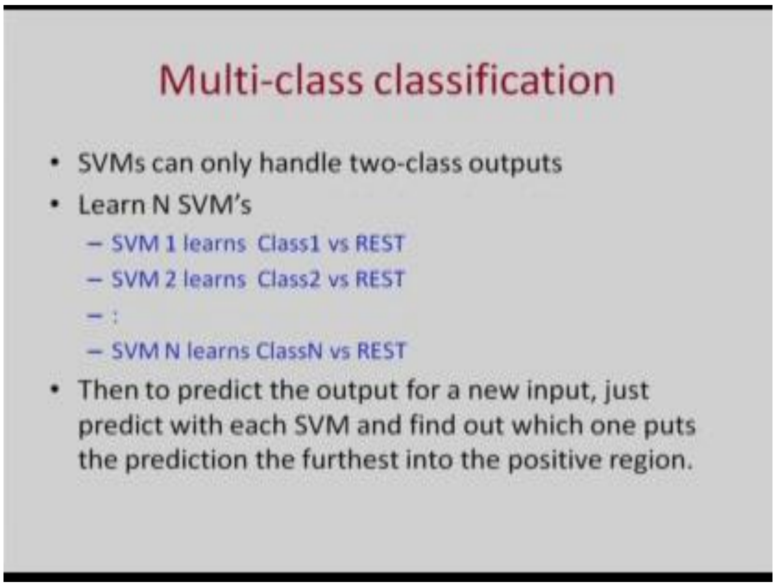
Performance

- Support Vector Machines work very well in practice.
  - The user must choose the kernel function and its parameters
- They can be expensive in time and space for big datasets
  - The computation of the maximum-margin hyper-plane depends on the *square of the number of training cases.*
  - We need to store all the support vectors.
- The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space.

So, support vector machine in conclusion, we can say they perform; they have in found to perform quite well in practice, but in certain cases if an appropriate kernel function exists. However, the user has to choose the appropriate kernel function, if an appropriate kernel function exists to map the original attribute space to a features space where the points are linearly separable then the support vector machine works well, but the user has to define the appropriate kernel. They can be expensive in time and space for big datasets.

But because in the solution, you look at the pairwise training examples, but the computation of maximum margin hyper plane depends on the square of the number of training cases. Secondly, for the solution we need to store all the support vectors. If the number of support vectors is large then also solution time will increase the solution. Time depends on the number of support vectors and number of training examples. The kernel trick can also be used to do principle component analysis in a higher dimensional space. We have already talked about principle component analysis where we take the features and reduce it, but we could to PCA in a higher dimensional space using kernel functions, but we will not talk about this today.

(Refer Slide Time: 26:04)



Multi-class classification

- SVMs can only handle two-class outputs
- Learn N SVM's
  - SVM 1 learns Class1 vs REST
  - SVM 2 learns Class2 vs REST
  - :
  - SVM N learns ClassN vs REST
- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

We will just conclude by seeing the support vector, classify that we have seen is used by merrily for classifying 2 classes, separating 2 classes what if you have multi classes. So, there is the support vector machine by itself only looks at 2 classes, but we can use a combination of SVM's to deal with multiple classes. Suppose, we have n classes and you can learn n support vector machine. Suppose, SVM 1 learns to classify class 1 versus the REST, SVM 2 to separate class 2 versus REST, SVM n to separate class n versus REST.

So, we develop n different support vector classifiers.

Now, to predict output for a new test point, we will apply all the support vector machines SVM 1, SVM 2, SVM n and then we will find out which 1 of the SVM's give the prediction with the highest confidence. So, highest confidence is we find w i x i, w i x i plus b and that should be greater than times y i, that should be greater than equal to 1 minus i, i the bigger the value is higher the confidence because further it lies from the margin. So, we can choose out of this n outputs, the 1 for which the point is for this margin and we can take that positive classification corresponding to that class with this we come to the end of kernel SVM.

Thank you.