Computational Learning Theory **Spring Semester, 2001/2**

Lecture 11: July 26 Part I

Lecturer: Yishay Mansour *Scribe: Gur Yaari, Tzur Sayag*

11.1 Basic Idea

We try to deal with non separable linear problem. We overcome the linearity constraint by mapping the problem into a higher dimension space. For solving this problem we use the Linear Learning Machine (LLM). In the higher dimension space there are "many" hyperplanes to choose from. We will also present a method for choosing among these hyperplane.

In this case, the VC dimension metric is no longer a suitable bounding measurement for learning complexity, we define a new bounding metric based on the Support Vector Machine (SVM) margin.

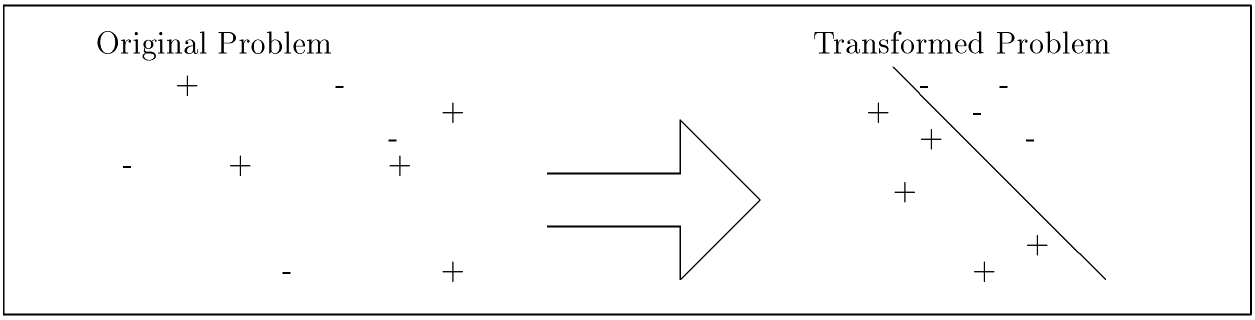

Figure 11.1: Basic Idea.

The SVM approach deals with the problem by dividing it into two separate phases:

- Kernel function which performs the embedding of the problem into a higher dimension.
- Learning algorithm working in the embedded space.

11.2 Basic Walkthrough: Kernel Perceptron

11.2.1 The Perceptron Algorithm

In essence, a perceptron is a weighted summation of several boolean inputs which gives a boolean classification according to a threshold. One can view the perceptron as a hyperplane decision surface. In this case, the input $\vec{x}$ is a vector of points in the n-dimensional space.

The algorithm is initialized with some set of weights $\vec{w} = (w_1, ..., w_n)$ and $\theta = 1$, which form an initial hypothesis h. The learning process proceeds as following: Given an example x, it uses $h(x)$ to classify it and outcome is compared to the correct one, i.e. $f(\vec{x})$. if $h(\vec{x}) = f(\vec{x})$, i.e. the algorithm was correct, $h(\vec{x})$ is maintained. if $h(\vec{x}) \neq f(\vec{x})$, i.e. a mistake was done, we update $h(\vec{x})$, for a complete description of the update rule, see Lecture 8.

11.2.2 Definitions

1. Hyperplane Margin (of a classifier)

$$D(\vec{W}) = \min_{<x_i,t_i>\in S} (\vec{W} \cdot \vec{X})$$

$x_i - Sample \in R^n$.
S - Data set.
$t_i \in \{-1, 1\} - x_i$'s classification.

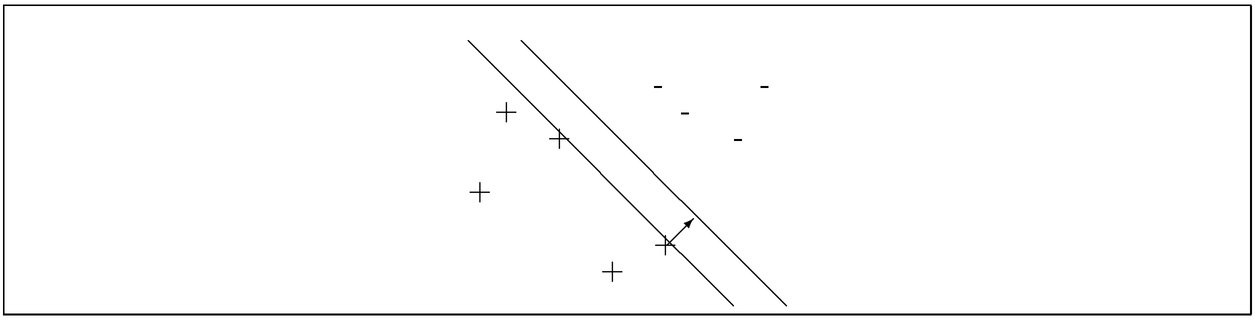

Figure 11.2: Classifier Margin.

2. Mistake bound for the Perceptron,
   we have previously shown that for the optimal hyperplane (which correctly classifies all the training set), the error bound depends on the hyperplane margin in the following manner:

$$M \leq \left(\frac{X_{max}}{D(W^*)} \right)^2$$

11.2.3 Observations

1. The perceptron algorithm generates a solution which is a linear combination of the inputs:
$$\vec{W} = \sum (a_i \cdot t_i) \cdot \vec{x_i}$$
$a_i > 0$. Note that the 'direction' is determined by t_i 's sign while a_i is used as the learning rate.

2. only inputs on which $sign(f(\vec{x_i})) \neq t_i$ influence our solution. (recall that in the perceptron algorithm, we only refine $\vec{W}$ if we make a mistake on sample $\vec{x_i}$).

Given these 2 observations, one can think of a support vector as the vector on which we were mistaken.

11.2.4 Dual Representation

We can rewrite the basic function:
$$f(\vec{x}) =< \vec{w}, \vec{x} > +b = \sum a_i t_i < \vec{x_i}, \vec{x} > +b$$
formally, set
$$\vec{w} = \sum (a_i t_i \vec{x_i}),$$
then
$$< \vec{w}, \vec{x} >=< \sum (a_i t_i \vec{x_i}), \vec{x} >= \sum (a_i t_i) < \vec{x_i}, \vec{x} >$$

Now recall the perceptron update rule:
$$t_j = 1, f(\vec{x_j}) = -1 \Rightarrow \vec{w_{j+1}} = \vec{w_j} + \vec{x_j}$$
$$t_j = -1, f(\vec{x_j}) = 1 \Rightarrow \vec{w_{j+1}} = \vec{w_j} - \vec{x_j}$$
rewriting it we get:
if $t_j \cdot (\sum a_i t_i < \vec{x_i}, \vec{x_j} >) < 0$ (mistake on sample $\vec{x_j}$)
then $a_j = a_i + 1$.
Observe that in this representation, the input data appears only in the inner product ($< \vec{x_i}, \vec{x_j} >$). We will use this later to replace the inner product with a different black box.

11.2.5 Perceptron Limitations

1. It is assumed that the training examples are linearly separable.

2. It converges only on linearly separable data.

3. It is defined on vectorial data.

11.3 The Idea Of Kernel

1. The basic idea is to transform the samples in to a different space (the new space is possibly with higher dimension).
2. Hopefully, in the new space we get a linearly separable data. (remember that for data that is linearly separable we know how to solve).

11.3.1 Kernel Mapping

We have shown that what is actually needed for computing the new weight vector, is to compute the inner product.

Let $M(x)$ be the mapping function.

Let $Kernel(\vec{x}, \vec{y}) =< M(\vec{x}), M(\vec{y}) >$. Clearly, it is sufficient to compute $Kernel(\vec{x}, \vec{y})$ and replace the original inner product ($< \vec{x}, \vec{y} >$) with it. notice that if $Kernel(\vec{x}, \vec{y})$ can be computed directly (without using the mapping operator (M) on each of the inputs (x and y), the dimension of $M(x)$ is less important. if the mapping is needed (to compute $Kernel(\vec{x}, \vec{y})$, depending on M 's dimension, it may be computationally expensive. Fortunately, some kernel functions can be directly computed without using the mapping operator.
Consider the following example:

$$M(x_1, x_2) = (x_1^2, x_2^2, \sqrt{x_1 \cdot x_2}))$$

In this example we save a quadric number of direct multiplications (4 instead of 12).

11.3.2 Kernel Matrix

In this example we have $x_1, ..., x_4$ as inputs. All kernel values can be computed in advance and saved in the Kernel Matrix for future usage. This matrix is also called the Gram matrix K.

$$a(i,j) = Kernel(x_i, x_j).$$

$$\begin{pmatrix} Kernel(x_1,x_1) & Kernel(x_1,x_2) & Kernel(x_1,x_3) & Kernel(x_1,x_4) \\ Kernel(x_2,x_1) & Kernel(x_2,x_2) & Kernel(x_2,x_3) & Kernel(x_2,x_4) \\ Kernel(x_3,x_1) & Kernel(x_3,x_2) & Kernel(x_3,x_3) & Kernel(x_3,x_4) \\ Kernel(x_4,x_1) & Kernel(x_4,x_2) & Kernel(x_4,x_3) & Kernel(x_4,x_4) \end{pmatrix}$$

11.3.3 Examples Of Basic Kernels

1. Polynomial, which is a natural generalization of the above example.

$$K(x, z) = (< x, z >)^d$$

It can be computed in linear time but maps the input to a space which it's dimension is powered by d. The figure below, demonstrates how a polynomial kernel can be useful for classification.

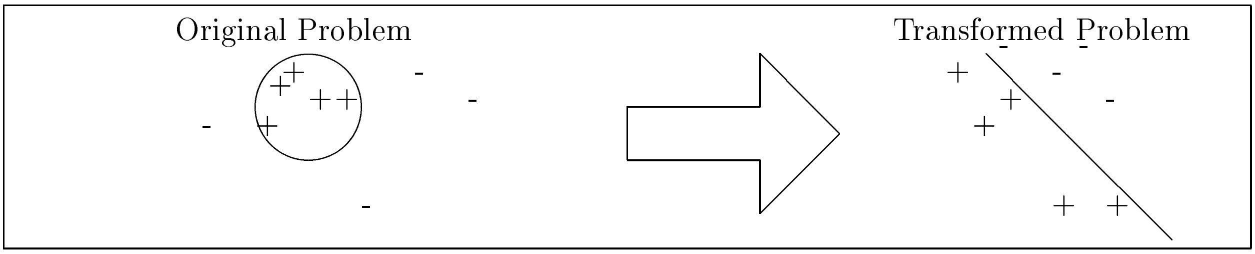

Figure 11.3: Polynomial Kernel.

2. Gaussian Kernel.

$$Kernel(x,z) = \exp\left(\frac{\|x-z\|^2}{2\sigma}\right)$$

11.3.4 Kernel Function Closure Properties

Kernel function are closed under these 4 operators. For any constant c and for $K', K'' \in$ {kernel functions}:

1. $K(x,z) = K'(x,z) + c$
2. $K(x,z) = K'(x,z) \cdot c$
3. $K(x,z) = K'(x,z) \cdot K''(x,z)$
4. $K(x,z) = K'(x,z) + K''(x,z)$

These properties give us the ability to create new kernels using the basic ones.