Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 / MoSIG M1 Second Semester 2010/2011

Lesson 20 6 May 2011

Least Squares, Perceptrons and Kernel Methods

Contents

Sources Bibliographiques :
"Pattern Recognition and Machine Learning", C. M. Bishop, Springer Verlag, 2006.

Notation

x	a variable
X	a random variable (unpredictable value)
$\vec{x}$	A vector of D variables.
$\vec{X}$	A vector of D random variables.
D	The number of dimensions for the vector $\vec{x}$ or $\vec{X}$
E	An observation. An event.
k	Class index
K	Total number of classes
ω_k	The fact that E $\in C_k$
$\hat{\omega}_k$	The decision (estimation) that E $\in C_k$
M_k	Number of examples for the class k. (think M = Mass)
M	Total number of examples. $M = \sum_{k=1}^{K} M_k$
$\{X_m^k\}$	A set of M_k examples for the class k. $\{X_m\} = \bigcup_{k=1,K} \{X_m^k\}$
$\{y_m\}$	A set of class labels (indicators) for the samples For detection (K=2), $y \in \{+1, -1\}$

Least Squares Estimation

We seek to estimate a decision rule

if $y(X) = (\vec{w}^T \cdot \vec{X} + w_o) \geq 0$ then C_1 else C_2

The decision surface is the hyperplane where $y(X) = \vec{w}^T \cdot \vec{X} + w_o = 0$

Homogeneous Coordinate Notation

It will often be convenient to use "homogeneous coordinates" to represent $\vec{X}$.

That is, we will add an extra "dummy" dimension to $\vec{X}$ to represent y(X) as vector product. In this case, $\vec{X}$ becomes a D+1 vector, with 1 as the last coefficient.

We also add w_o as the D+1 coefficient of $\vec{w}$

$$\vec{X} = (1, x_1, x_2, \ldots, x_D)$$
$$\vec{w} = (w_o, \vec{w})$$

The linear decision surface becomes

$$y(\vec{X}) = \vec{w}^T \cdot \vec{X} = \sum_{d=0}^{D} w_d x_d$$

The function $y(\vec{X}) = \vec{w}^T \cdot \vec{X}$ is sometimes known as a linear regression on $\vec{X}$.

Estimating a two-class decision surface (K=2) with Least Squares

We illustrate least squares with a two-class problem. The method can be extended to multiple classes. Our problem is to estimate a weight vector, $\vec{w}$ and a constant w_o that separates two classes.

The decision rule is

if $y(X) = (\vec{w}^T \cdot \vec{X} + w_o) \geq 0$ then C_1 else C_2

The decision surface is the hyperplane where $y(X) = \vec{w}^T \cdot \vec{X} + w_o = 0$

We can "bias" the decision surface towards class 1 or class 2 by adding a constant, b, to w_o. The constant is referred to as the "bias" (not to be confused with "bias" in estimating a variance).

If we normalize the vector $\vec{w}$ to unit norm, then

$\vec{N} = \frac{\vec{w}}{\|\vec{w}\|}$ is the normal to this hyperplane. and

$d = \frac{w_o}{\|\vec{w}\|}$ is the (signed) perpendicular distance from the plane to the origin

For least square regression, assume that each of the M training samples $\{\vec{X}_m\}$ are labeled with an indicator variable, y_m such that $y_m=1$ for Class 1 and $y_m= -1$ for class 2.

A least-squares estimate for the function $y(\vec{X}) = \vec{w}^T\vec{X} + w_o$ can be obtained in closed form.

Define a "Loss" function: $L(\hat{W}) = \sum_{m=1}^{M} (y_m - \vec{w}^T\vec{X}_m)^2$

We will use the M training samples to compose a matrix **X** and a vector **Y**.

$X = \begin{pmatrix} \vec{X}_1 & \vec{X}_2 & \cdots & \vec{X}_M \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \vec{X}_1 & \vec{X}_2 & \cdots & \vec{X}_M \end{pmatrix}$ (D+1 rows by M columns)

$\mathbf{Y} = (y_1, y_2, ..., y_M)^T$ (M rows).

We seek the D+1 Coefficient homogenous coordinate weight vector $\vec{w} = \begin{pmatrix} w_o \\ \vec{w} \end{pmatrix}$

We write $L(\vec{w}) = (Y - X^T\vec{w})^T(Y - X^T\vec{w})$

To minimize the loss function, we calculate the derivative and solve for $\vec{w}$ when the derivative is 0.

$$\frac{\partial L(\vec{w})}{\partial \vec{w}} = -2X^TY + 2X^TX\vec{w} = 0$$

Thus $X^TY = X^TX\vec{w}$ and $\vec{w} = (X^TX)^{-1}X^TY$

Our decision surface is : $y(\vec{X}) = \vec{w}^T \cdot \vec{X} = 0$

The term $X^+ = (X^TX)^{-1}X^T$ is the Moore Penrose pseudo inverse.

Perceptrons

A perceptron is an incremental learning method for linear classifiers invented by Frank Rosenblatt in 1956. The perceptron is an on-line learning method in which a linear classifier is improved by its own errors.

A perceptron learns a hyper-plane to separate training samples. When the training data are perfectly separated the data is said to be "separable". Otherwise, the data is said to be non-separable.

The "margin", γ, is the smallest separation between the two classes. (The distance to the point closest to the hyper-plane).

When all the training samples are separable, the algorithm uses the errors to update the hyperplane plane until there are no more errors. When the training data is non-separable, the method may not converge, and must be arbitrarily stopped after a certain number of iterations.

The perceptron linear decision function is

$$\text{if } (\vec{w}^T\vec{X}+b) > 0 \text{ then positive else negative}$$

This is sometimes written as : $f(\vec{x}) = \vec{w}^T\vec{x}+b$

$$h(\vec{x}) = sign(\vec{w}^T\vec{x}+b)$$

Assume that we have a training set of M samples $S=\{\vec{X}_m, y_m\}$
where $y_m=+1$ for positive detection and -1 for negative detection.

A classifier is defined by a D coefficient weight vector $\vec{W}$ and a bias b.

A classifier correctly classifies the sample $(\vec{X}_m, y_m)$ if

$$y_m(\vec{w}^T\vec{X}_m+b) > 0$$

The learning algorithm uses the update rule:

$$\text{if } y_m(\vec{w}_i^T\vec{X}_m+b) \le 0 \text{ then } \vec{w}_{i+1}^T \leftarrow \vec{w}_i^T \eta\, y_m\vec{X}_m$$

where η is a learning rate.

The result is a linear combination of the training samples:

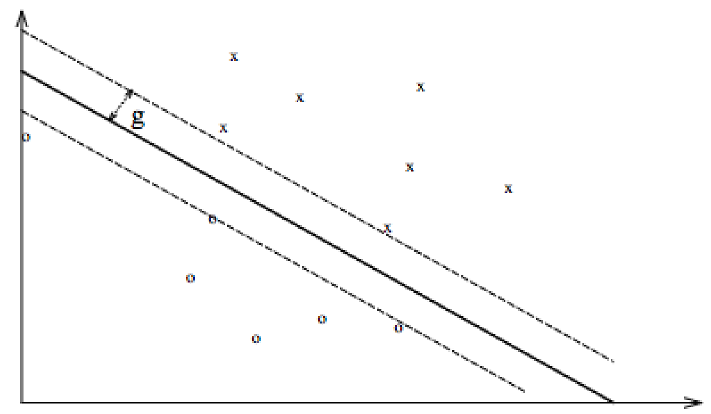
$$\vec{w}^T = \sum_{m=1}^{M} a_m\, y_m \vec{X}_m \quad \text{where } a_m \geq 0.$$

Only mistakes are used to drive learning. The coefficient a_m reflects the difficulty of classifying the training sample $(\vec{X}_m, y_m)$.

Algorithm:

```
$\vec{w}_o \leftarrow 0; b_o \leftarrow 0; i \leftarrow 0;$
$R \leftarrow max\{ || \vec{X}_m || \}$
REPEAT
    FOR m = 1 TO  M DO
        if $y_m(\vec{w}_i^T \vec{X}_m + b_i) \leq 0$ then $\vec{w}_{i+1}^T \leftarrow \vec{w}_i^T \eta\, y_m \vec{X}_m$
            $b_{i+1} \leftarrow b_i + \eta\, y_m R^2;$
            $i \leftarrow i + 1;$
        END IF
    END FOR
UNTIL no mistakes in FOR loop.
```

The margin, γ is the minimum distance of a sample from the hyperplane



If the coefficients are normalized:

$$W_i' = \frac{W_i}{\|W_i\|} \qquad b_i' = \frac{b_i}{\|W_i\|}$$

Then after each stage the margin for each sample, m, is

$$\gamma_m = y_m(\vec{w}_i^T \vec{X}_m + b_i)$$

and the margin is $\gamma = min\{\gamma_m\}$

The quality of the perceptron is give by the histogram of the margins.

Duality of Perceptrons.

A dual representation for a perceptron is :

$$f(\vec{X}) = \vec{w}^T \vec{X} + b = \sum_{m=1}^{M} a_m y_m \langle \vec{X}_m, \vec{X} \rangle + b$$

where $\vec{w}^T = \sum_{m=1}^{M} a_m y_m \vec{X}_m$ and $\langle \vec{X}_m, \vec{X} \rangle = \sum_{d=1}^{D} X_d^m X_d$

is an inner product.

The update rule can be rewritten as

$$\text{if } y_m \sum_{m=1}^{M} a_m y_m \langle \vec{X}_m, \vec{X} \rangle + b \leq 0 \text{ then } a_m \leftarrow a_m + \eta$$

Note that in the dual representation, data only appears inside the inner product. This is an important property for kernel methods.

A perceptron is a sort of Support Vector machine.
All SVM's have the property of duality.

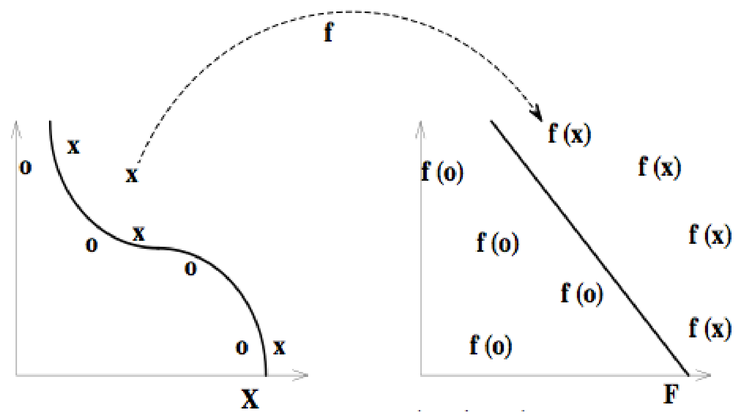
Perceptrons and SVMs are called "Linear Learning Machines" or LLMs

Using Linear Learning Machines for non-linear problems.

Linear classifiers are easy to learn, and can execute very fast.
However,
1) LLMs are sensitive to noisy data
2) LLMs require linearly separable data
3) LLMs are applied to Numerical feature Vectors.

We can apply linear classifiers to non-linear problems using a non-linear mapping of the feature space.

Map a non-linear problem onto a space where the data is linearly separable:



However, there this can require VERY large D.

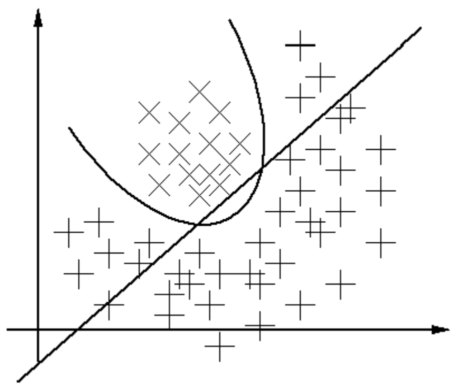
Solution: Kernel methods

Kernel Methods

Kernel Methods transform a non-linear function into a linear function, often in a much higher dimensional space. Thus they enable linear discriminant methods to be applied to a large class of problems where the data are dispersed in a non-linear manner.

Linear methods are very well suited for use with very high dimensional feature space provided that the patters can be separated by a plane.

Kernel Methods provide an elegant solution for clustering and classifying patterns in complex non-linear data by mapping the data into a higher dimensional space where the data can be separated by a linear method.



Kernels make it possible to
1) Solve the computational problems of high dimensional spaces
2) Extend LLMs to infinite dimensional spaces
3) Extend LLMs to non-numerical and symbolic data!

Dual representation for an LLM: $f(\vec{X}) = \sum_{m=1}^{M} a_m \, y_m \left\langle \vec{X}_m, \vec{X} \right\rangle + b$

To apply a kernel, we replace two arguments by the dot product of a function, *$\phi(X)$*

$$\left\langle \vec{X}_1, \vec{X}_2 \right\rangle \leftarrow k(\vec{X}_1, \vec{X}_2) = \left\langle \phi(\vec{X}_1), \phi(\vec{X}_2) \right\rangle$$

This gives an LLM of the form:

$$f(\vec{X}) = \sum_{m=1}^{M} a_m\, y_m \left\langle \phi(\vec{X}_m), \phi(\vec{X}) \right\rangle + b$$

SVM's are Linear Learning Machines that

1) Use a dual representation and
2) Operate in a kernel induced space

Kernel Functions and Kernel Methods

A Kernel is a function that returns the inner product of a function applied to two arguments. The Kernel matrix is also known as the Gram Matrix.

$$f(\vec{X}) = \sum_{m=1}^{M} a_m y_m \left\langle \phi(\vec{X}_m), \phi(\vec{X}) \right\rangle + b$$

The key notion of a kernel method is an inner product space.

$$\langle \vec{x}, \vec{z} \rangle = \sum_{d=1}^{D} x_d z_d$$

In general, we will define a kernel function as a quadratic mapping of a feature space, *$\phi(x)$*

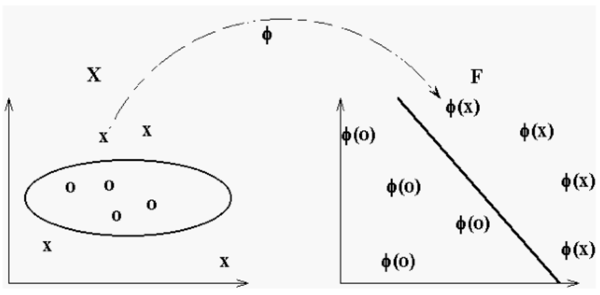
$$k(\vec{X}_1, \vec{X}_2) = \vec{\phi}(\vec{X}_1)^T \vec{\phi}(\vec{X}_2)$$

Note that the kernel is a symmetric function of its arguments, so that

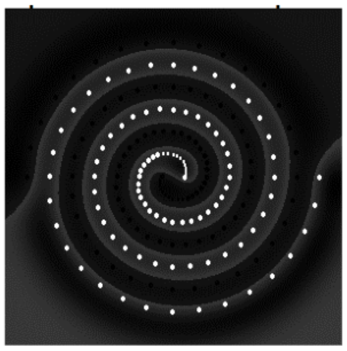
$$k(\vec{X}_1, \vec{X}_2) = k(\vec{X}_1, \vec{X}_2)$$

There are a large variety of possible kernel functions that can be used, depending on the problem.

example: Polynomial Kernel:



Spiral (separated with Gaussian Kernels)



In order to be "valid", a kernel must correspond to a scalar product of some feature space. That is, there must exist a space such that

$$k(\vec{X}_1,\vec{X}_2)=\vec{\phi}(\vec{X}_1)^T\vec{\phi}(\vec{X}_2)=\sum_{n=1}^{N}\phi_n(\vec{X}_1)\cdot\phi_n(\vec{X}_2)$$

For example, consider a quadratic kernel in a space where D=2.

In this case, $k(\vec{x},\vec{z})=(\vec{x}^T\vec{z})^2=(x_1z_1+x_2z_2)^2=(x_1^2z_1^2+2x_1z_1x_2z_2+x_2^2z_2^2)$

This can be expressed as an inner product space where

$$\phi(\vec{x})=x_1^2+\sqrt{2}x_1x_2+x_2^2$$

giving:

$$k(\vec{x},\vec{z})=\vec{\phi}(\vec{x})^T\vec{\phi}(\vec{z})$$

A necessary, and sufficient condition that a Kernel function be "valid" is that the GRAM matrix be positive and semidefinite for all choices of $\{\vec{X}_m\}$

A GRAM (or Grammian) Matrix for $\vec{x}$ is $\vec{x}^T\vec{x}$

The linear vector $\vec{x}$ is projected onto a quadratic surface

Gaussian Kernel

The Gaussian exponential is very often used as a kernel function.
In this case:

$$k(\vec{x},\vec{x}') = e^{-}\frac{\|\vec{x}-\vec{x}'\|}{2\sigma^2}$$

This is often called the Gaussian Kernel. It is NOT a probability density.
We can see that it is a valid kernel because:

$$\|\vec{x}-\vec{x}'\|^2 = \vec{x}^T\vec{x} - 2\vec{x}^T\vec{x}' + \vec{x}'^T\vec{x}'$$

Among other properties, the feature vector has infinite dimensionality.

Kernel functions can be defined over graphs, sets, strings and text!

Consider for example, a non-vectoral space composed of a Set of words S.
Consider two subsets of S : $A_1 \subset S$ and $A_2 \subset S$

The can compute a kernel function of $A_1$and A_2 as

$$k(\vec{x},\vec{x}') = 2^{|A_1 \cap A_2|}$$

where |A| denotes the number of elements (the cardinality) of a set.