Pattern Recognition and Machine Intelligence

James L. Crowley

ENSIMAG 3 MMIS First Semester 2010/2011

Lesson 10 12 January 2010

Perceptrons and Kernel Methods

Contents

Source:
"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 $\vec{X}$
$\{y_m\}$	A set of class labels (indicators) for the samples.
	For a 2 Class problem, t_m is -1 or +1
$S = \{\vec{X}_m, y_m\}$	A set of M Training samples M samples and their indicator variable.
M	Total number of training samples. (think M = Mass)

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

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:

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

where η is a learning rate.

The result is a linear combination of the training samples:

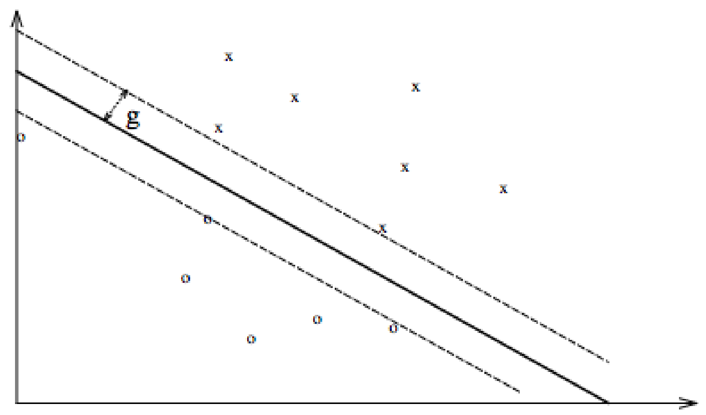
$$\vec{w}^T = \sum_{m=1}^{M} a_m \, y_m \vec{X}_m \quad \text{where } a_m \geq 0.$$

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.

The dual representation for a perceptron is :

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

where $\vec{w}^T = \sum_{m=1}^{M} a_m\, y_m \vec{X}_m$

The update rule can be rewritten as

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

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.

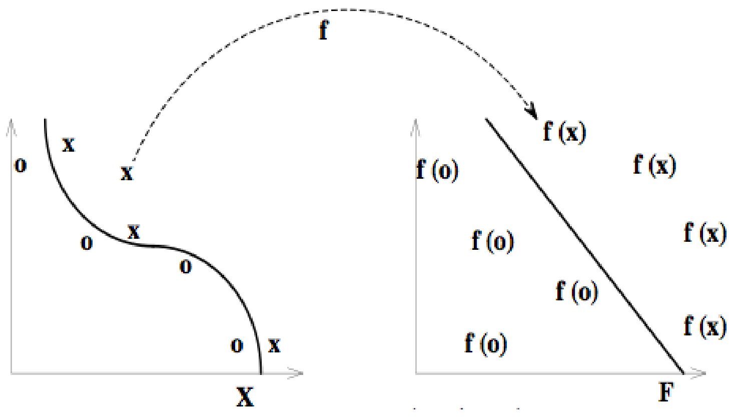
Perceptrons and SVMs are called "Linear Learning Machines" or LLMs

Using Linear Learning Machines for non-linear problems.

Linear classifiers are easy to learn, and can execute very fast.
However,
1) they are sensitive to noisy data
2) They require linearly separable data
3) They can only be 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

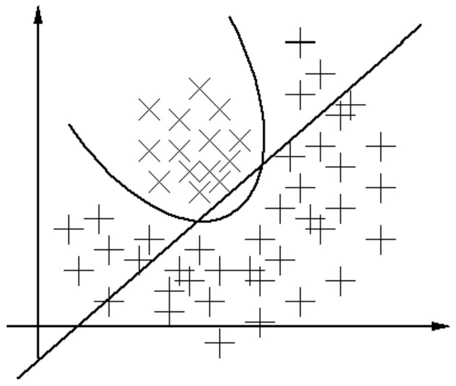
Introduction to Kernel Methods

Kernel Methods transform map a non-linear function into a linear function 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.

As an example, consider the appearance manifold in PCA space of the a normalized face imagette, under changes in face orientation. The imagettes map to a continuous manifold in the PCA space, but the manifold is a very non-linear subspace of PCA.

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) Be extended to infinite dimensional spaces
3) Be extended 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, *$\phi(x)$*

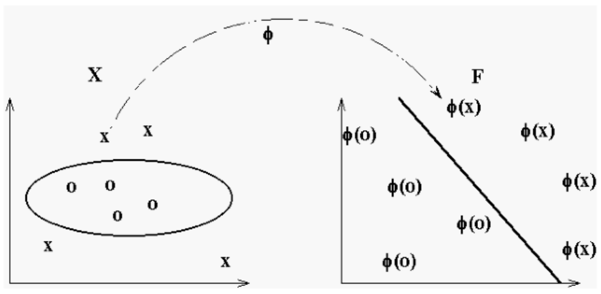
$$k(\vec{X}_1, \vec{X}_2) = \vec{\phi}(\vec{X}_1)^T \vec{\phi}(\vec{X}_2)$$

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

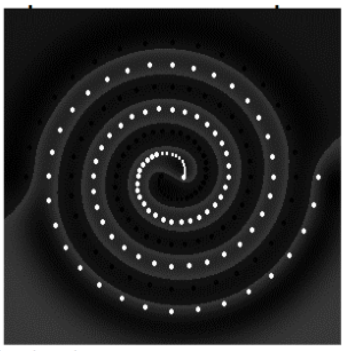
$$k(\vec{X}_1, \vec{X}_2) = k(\vec{X}_1, \vec{X}_2)$$

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.

Probabilistic generative models tend to be more robust with missing data and data of variable length, while Probabilistic Discriminative models tend to give better performance and lower cost.

We can combine generative and discriminative models using a kernel.

Given a generative model p(X) we can define a kernel as:

$$k(\vec{x},\vec{x}') = p(\vec{x})p(\vec{x}')$$

This is clearly a valid kernel because it is a 1-D inner product. Intuitively, it says that two feature vectors, x, are similar if they both have high probability.

We can extend this with conditional probabilities to

$$k(\vec{x},\vec{x}') = \sum_{n=1}^{N} p(\vec{x} \mid n) p(\vec{x}' \mid n) p(n)$$

Two vectors, $\vec{x},\vec{x}'$ will give large values for the kernel, and hence be seen as similar, if they have significant probability for the same components.

Kernel functions enable application of linear classifiers to non-linear problems.
Let us look at some linear classifiers and how they can be used.

Fisher Linear Discriminant.

The Discrimination problem can be viewed as a problem of projecting the D dimensional feature space onto a lower dimensional K dimensional space.

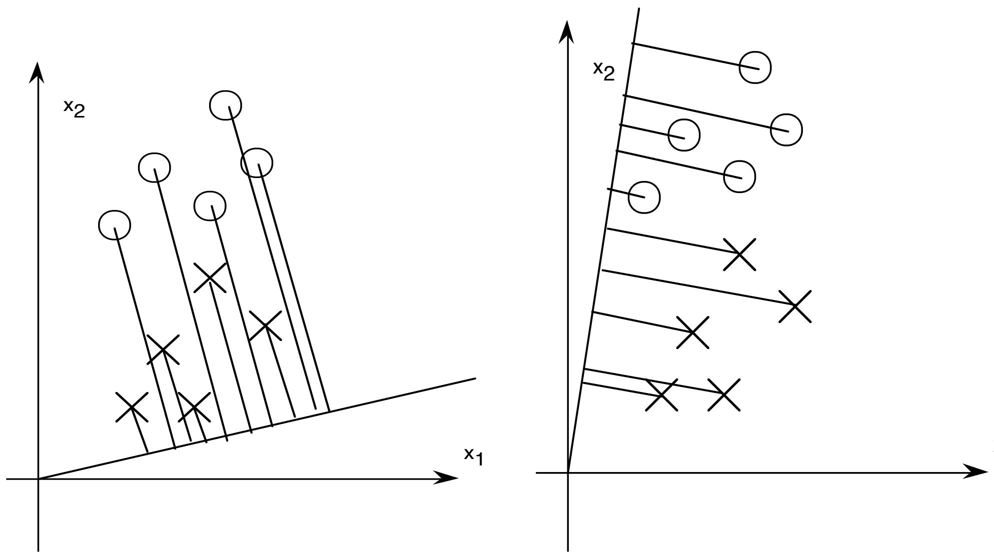
The tool for such projection is the Fisher discriminant.

Two Class solution

The principle of the Fisher linear discriminant is to project the vector X with D_x onto a variable z (D=1) by a linear projection F such that the classes are most separated.

$$z = \vec{F}^T \cdot \vec{X}$$

A Fisher metric, J(F) is used to choose F such that the two classes are most separated.



The error rates of the classification (FP, FN) depends on the direction of F.

Note that F is commonly normalized so that $\|\vec{F}\| = 1$

Assume a set of M_k training samples for each class, $\{\vec{X}_m^k\}$

The average for each class is:

$$\vec{\mu}^k = E\{\vec{X}^k\} = \frac{1}{M_k}\sum_{m=1}^{M_k}\vec{X}_m^k$$

Moments are invariant under projections. Thus the projection of the average is the average of the projection.

$$\mu_z^k = E\{F^T \cdot \vec{X}_m^k\} = F^T \cdot E\{\vec{X}_m^k\} = F^T \cdot \vec{\mu}_k$$

The inter-class distance between between classes 1 and 2 is

$$d_{12} = \mu_z^1 - \mu_z^2 = \vec{F}(\vec{\mu}_1 - \vec{\mu}_2)$$

The Fisher metric is designed to make the inter-class distance, d_{12}, as large as possible. The key concept is the "scatter" of the samples. Scatter can be seen as unormalised covariance.

The “scatter” for the M_k samples $\{\vec{X}_m^k\}$ of the set k is a matrix : $\mathbf{S}_k$.
This is the same as an "unnormalised" covariance.

$$S_k = M_k \Sigma_k = \sum_{m=1}^{M_k} (\vec{X}_m^k - \vec{\mu}^k)(\vec{X}_m^k - \vec{\mu}^k)^T$$

The transformation F projects the vector $\vec{X}$ onto a scalar z.

$$z = \vec{F}^T \cdot \vec{X}$$

The scatter of the class after projection is

$$S_z^k = \sum_{m=1}^{M_k} (z_m^k - \mu_z^k)^2$$

The fisher criteria tries to maximize the ratio of the separation of the classes compared to their scatter by maximizing the ratio of within and between class scatter.

$$J(F) = \frac{\left(\mu_z^1 - \mu_z^2\right)^2}{s_z^1 + s_z^2}$$

Let us define the between class scatter as $S_B = (\vec{\mu}_1 - \vec{\mu}_2)(\vec{\mu}_1 - \vec{\mu}_2)^T$

then $\left(\mu_z^1 - \mu_z^2\right)^2 = F^T\left((\vec{\mu}_1 - \vec{\mu}_2)(\vec{\mu}_1 - \vec{\mu}_2)^T\right)F = F^T S_B F$

And let us define within class scatter as

$$S_W = S_1 + S_2 = \sum_{m=1}^{M_1} (\vec{X}_m^1 - \vec{\mu}_1)(\vec{X}_m^1 - \vec{\mu}_1)^T + \sum_{m=1}^{M2} (\vec{X}_m^2 - \vec{\mu}_2)(\vec{X}_m^2 - \vec{\mu}_2)^T$$

Then

$$s_z^1 + s_z^2 = F^T(S_1 + S_2)F = F^T S_W F$$

Then

$$J(F) = \frac{\left(\mu_z^1 - \mu_z^2\right)^2}{s_z^1 + s_z^2} = \frac{F^T S_B F}{F^T S_W F}$$

Taking the derivative with respect to F, we find that J(F) is maximized when

$$\left(F^T S_B F\right) S_W F = \left(F^T S_W F\right) S_B F$$

Because $S_B F$ is always in the direction $\vec{\mu}_1 - \vec{\mu}_2$

Dropping the scale factors $\left(F^T S_B F\right)$ and $\left(F^T S_W F\right)$ we obtain

$$S_W F = \vec{\mu}_1 - \vec{\mu}_2$$

and thus $F = S_W^{-1}\left(\vec{\mu}_1 - \vec{\mu}_2\right)$

Fisher's Discriminant for Multiple Classes.

Fisher's method can be extended to the derivation of K > 2 linar discriminants.
Let us assume that the number of features is greater than the number of classes,
D > K.

We will look for functions that project the D features on D' < D features to form a new feature vector, $\vec{Y} = \vec{w}^T \vec{X}$ (note that there is no constant term).

as before, we define the class Mean, $\vec{\mu}_k$, class Scatter S_k and within-class scatter S_W

Class Mean: $\vec{\mu}_k = \frac{1}{M_k}\sum_{m=1}^{M_k} \vec{X}_m^k$

Class Scatter: $S_k = \sum_{m=1}^{M_k} (\vec{X}_m^k - \vec{\mu}_k)(\vec{X}_m^k - \vec{\mu}_k)^T$

Within Class Scattter $\vec{\mu}_k = \frac{1}{M_k}\sum_{m=1}^{M_k} \vec{X}_m^k$

We need to generalisation of the between class covariant.
The total mean is:

$$\vec{\mu} = \frac{1}{M}\sum_{k=1}^{K}\sum_{m=1}^{M_k} \vec{X}_m^k = \frac{1}{M}\sum_{k=1}^{K} M_k \vec{\mu}_k$$

The between class scatter is:

$$S_B = \sum_{k=1}^{K} M_k (\vec{\mu}_k - \vec{\mu})(\vec{\mu}_k - \vec{\mu})^T$$

Which gives the total scatter as

$$S_T = S_W + S_B$$

We can define similar scatters in the target space:

$$\vec{\mu}_k = \frac{1}{M_k}\sum_{m=1}^{M_k} \vec{Y}_m^k \qquad \vec{\mu} = \frac{1}{M}\sum_{k=1}^{K}\sum_{m=1}^{M_k} \vec{Y}_m^k = \frac{1}{M}\sum_{k=1}^{K} M_k \vec{\mu}_k$$

$$S'_W = \sum_{k=1}^{K}\ \sum_{m=1}^{M_k} (\vec{Y}_m^k - \vec{\mu}_k)(\vec{Y}_m^k - \vec{\mu}_k)^T$$

$$S'_B = \sum_{k=1}^{K} M_k (\vec{\mu}_k - \vec{\mu})(\vec{\mu}_k - \vec{\mu})^T$$

We want to construct a set of projections that maximizes the between class scatter

$$J(W)=Tr\{W\cdot S_W\cdot W^T)^{-1}(WS_BW^T)$$

The W values are determined by the D' eigenvectors of $S_W^{-1}S_B$ that correspond to the D' largest Eigenvalues.