Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 / MoSIG M1 Second Semester 2011/2012

Lesson 20 27 April 2012

Learning Methods for Linear Detectors

Contents

Sources Bibliographiques :
"Neural Networks for Pattern Recognition", C. M. Bishop, Oxford Univ. Press, 1995.
"Pattern Recognition and Scene Analysis", R. E. Duda and P. E. Hart, Wiley, 1973.

Learning Linear Detectors

Linear Classifiers as Pattern Detectors

Linear classifiers are widely used to define pattern "detectors". This is used in computer vision, for example to detect faces or publicity logos, or other patterns of interest.

In the case of pattern detectors, K=2.

Class k=1: The target pattern.
Class k=2: Everything else.

In the following examples, we will assume that our training data is composed of M sample observations $\{\vec{X}_m\}$ where each sample is labeled with an indicator y_m

$y_m = +1$ for examples of the target pattern (class 1)
$y_m = -1$ for all other examples.

A variety of techniques exist to calculate the plane. The best choice can depend on the nature of the pattern class as well as the nature of the non-class data.

These include

1) Vector between center of gravities.
2) Fisher linear discriminant analysis,
3) Least Squares estimation
4) Perceptrons

In lesson 19 we saw the first two. In this lesson we look at least squares and perceptrons.

Least squares estimation of a hyperplane

Assume a training set of M labeled training samples $\{y_m, \vec{X}_m\}$
such that $y_m=+1$ for class 1 and $y_m = -1$ for class 2.

Our goal is to determine a discriminant function $g(\vec{X}) = \vec{W}^T \vec{X} + b$

which can also be expressed as : $g(\vec{X}) = \vec{X}^T \vec{W} + b$

We seek the "best" $\vec{W}$. This can eb determined by minimizing a "Loss" function:

$$L(\hat{W}) = \sum_{m=1}^{M} (y_m - \vec{X}_m^T \hat{W})^2$$

To build or function, we will use the M training samples to compose a matrix **X** and a vector **Y**.

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & x_{22} & \cdots & x_{2M} \\ \cdots & \cdots & & \cdots \\ x_{D1} & x_{D2} & \cdots & x_{DM} \end{pmatrix}$$ (D row by M columns)

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix}$$ (M coefficients).

We write $L(W) = (\mathbf{Y} - \mathbf{X}^T W)^T (\mathbf{Y} - \mathbf{X}^T W)$

To minimize the loss function, we calculate the derivative and solve for W when the derivative is 0.

$$\frac{\partial L(W)}{\partial W} = -2\,\mathbf{X}^T\mathbf{Y} + 2\,\mathbf{X}^T\mathbf{X}\,W = 0$$

Thus : $\mathbf{X}^T\mathbf{Y} = \mathbf{X}^T\mathbf{X}\,W$

$$W = (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y}$$

From this classifier an unknown event $\vec{X}$ as

if $\vec{W}^T\vec{X} + B > 0$ then $\hat{\omega}_1$ else $\hat{\omega}_2$

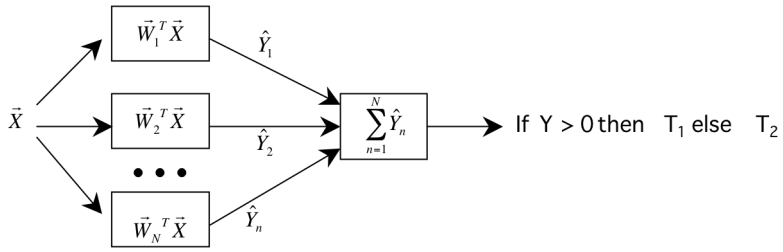
where "B" is an arbitrary "bias" constant.

We can trade False Positives for False negatives using the bias, B

A Committee of Boosted Classifiers

One of the more original ideas in machine learning the last decade is the discovery of a method by to learn a committee of classifiers by boosting. A boosted committee of classifiers can be made arbitrarily good: Adding a new classifier always improves performance.

A committee of classifiers decides by voting.



A feature vector is determined to be in the target class if the majority of classifiers vote > 0. Let us define v_i as the vote for the n^{th} classifier

For all i from 1 to I : If $\vec{W}_n^T\vec{X}_m + B > 0$ then $v_n = 1$ else $v_n = -1$.

$$\text{if } \sum_{n=1}^{N} v_i > 0 \text{ then } \hat{\omega}_1 \text{ else } \hat{\omega}_2$$

We can represent this with sgn :

$$v_n = sgn(\vec{W}_n^T\vec{X}_m + b)$$

where:

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x \le 0 \end{cases}$$

$$\text{if } \sum_{n=1}^{N} \text{sgn}(\vec{W}_n^T\vec{X}_m + b) \text{ then } \hat{\omega}_1 \text{ else } \hat{\omega}_2$$

To learn a boosted committee we iteratively add new classifiers to the committee. In each cycle we change the data set and learn a new classifier, W_i

The data set will be changed by giving additional weight to improperly classified samples. We learn the next class by multiplying the Y labels a weight vector, A_n.

$$\vec{W}_n = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T (\vec{A}_n \cdot \vec{Y})$$

Learning a Committee of Classifiers with Boosting

We can iteratively apply the above procedure to learn a committee of classifiers using boosting. For this we will create a vector of "weights" a_m for each training sample. Initially, all the weights are 1.

After each new classifier is added, we recalculate the weights to give more weight to improperly classified training samples.

As we add classifiers, whenever a sample is mis-classified by the committee we will increase its weight so that it carries more weight in the next classifier added.

Recall the committee vote is $\sum_{n=1}^{N} \text{sgn}(\vec{W}_n^T \vec{X}_m) > 0$ for class 1 (positive detection).

For m = 1 to M: if $(y_m \cdot \sum_{i=1}^{I} \text{sgn}(\vec{W}_i^T \vec{X}_m + b)) < 0$ then $a_m = a_m + 1$

The result is the $(n+1)^{th}$ weight vector A_{n+1}

We then learn the $n+1^{th}$ classifier from the re-weighted set by

$$\vec{W}_{n+1} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T (\vec{A}_{n+1} \cdot \vec{Y})$$

ROC Curve

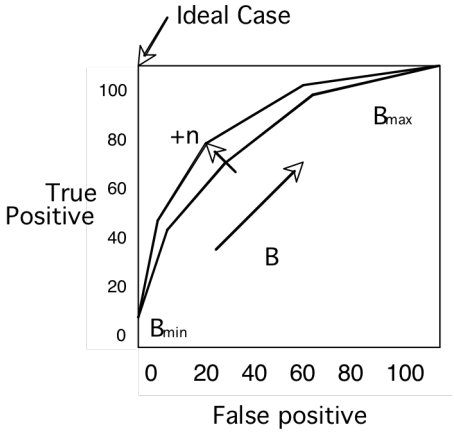
As we saw in lesson 19, The ROC describes the True Positives (TP) and False Positives (FP) for a classifier as a function of the global bias b.

For m = 1 to M:

if $\sum_{n=1}^{N} \text{sgn}(\vec{W}_i^T \vec{X}_m + b) > 0$ and $y_m > 0$ then TP=TP+1

if $\sum_{n=1}^{N} \text{sgn}(\vec{W}_n^T \vec{X}_m + b) > 0$ and $y_m < 0$ then FP=FP+1

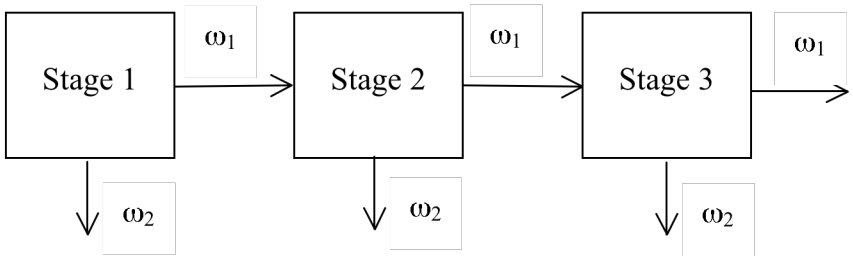
The Boosting theorem states that adding a new boosted classifier to a committee always improves the committee's ROC curve. We can continue adding classifiers until we obtain a desired rate of false positives and false negatives.



However, in general, the improvement provided for each new classifier becomes progressively smaller. We can end with a very very large number of classifiers.

Learning a Multi-Stage Cascade of Classifiers

We can optimize the computation time by using a multi-stage cascade.
With a multi-stage classifiers, only events labeled as positive are passed to the next stage.



Each stage is applied with a bias, so as to minimize False negatives.

$$\sum_{n=1}^{N} \mathrm{sgn}(\vec{W}_n^T \vec{X}_m + B) > 0$$

Stages are organized so that each committee is successively more costly and more discriminant.

Assume a set of M training samples $\{X_m\}$ with labels $\{y_m\}$.
Set a desired error rate for each stage j : (FP_j, FN_j).

For each stage, j, Train the j+1 stage with all positive samples from the previous stage.

Each stage acts as a filter, rejecting a grand number of easy cases, and passing the hard cases to the next stage. The stages become progressively more expensive, but are used progressively less often. Globally the computation cost decreases dramatically.

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 set of hyper-planes 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.

When are the training samples are separable, the algorithm uses the errors to update a 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.

Note that for all positive examples.

$$y_m(\vec{W}^T \vec{X}_m + B) > 0 \text{ if the classification is correct.}$$

The algorithm will apply a learning gain, η, to accelerate learning.

Algorithm:

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

After each stage the margin for each sample, m, is

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

The coefficients must be normalised to compute the margin.

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

The decision rule is as before :

$$\text{if } \operatorname{sgn}(\vec{W}^T\vec{X} + B)) > 0 \text{ then } \hat{\omega}_1 \text{ else } \hat{\omega}_2$$

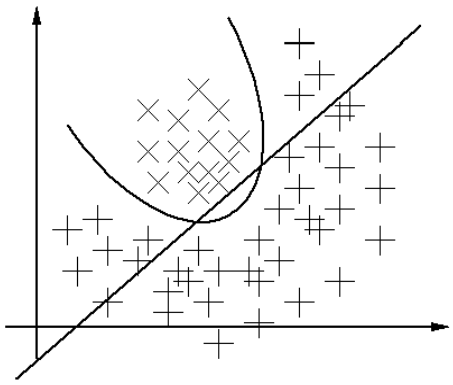
The quality of the perceptron is give by the histogram of the margins.

Kernel Methods

Kernel Methods transform 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.

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!

Linear methods are very well suited for use with very high dimensional feature space. We can map a quadratic decision space into a linear space by adding additional dimensions.

A quadratic suface in a D dimensional space can be tranformed into a linear surface in a D(D+1)/2 space by from the D dimensional space to a space P > D. using a kernel function, K().

For example a D=2 quadratic space is linear in 5 dimensions:

$\vec{X} = (x_1, x_2, ..., x_D)$ can be projected into a $P = \frac{D(D+1)}{2}$ dimension defined by $K(X) = W = (x_1, x_2, x_1^2, x_1x_2, x_2^2)$

Kernel Functions

Formally, 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, $\phi(x)$

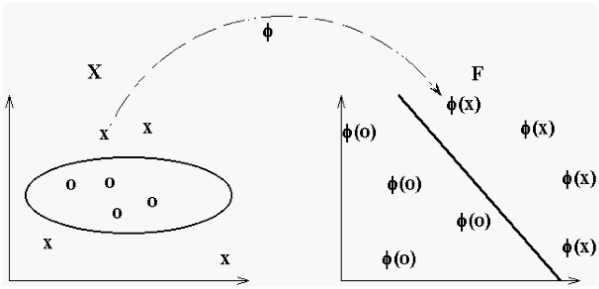
$$k(\vec{X}_1, \vec{X}_2) = \vec{\phi}(\vec{X}_1)^T \vec{\phi}(\vec{X}_2)$$

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

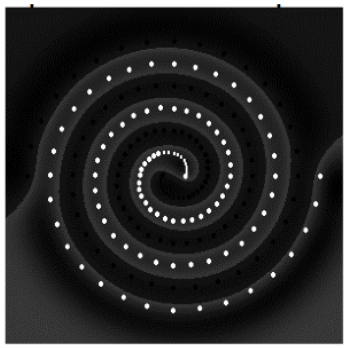
$$k(\vec{X}_1, \vec{X}_2) = k(\vec{X}_1, \vec{X}_2)$$

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 function for Symbolic Data

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.