Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 / MoSIG M1 Second Semester 2009/2010

Lesson 20 7 May 2010

# Linear Classification Methods

**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.

# 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 |
| $\omega_k$ | The statement (assertion) that E $\in T_k$ |
| $M_k$ | 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 $\in \{+1, -1\}$ |

# Learning Linear Classifiers.

$$\hat{\omega}_k = d(\vec{g}(\vec{X}))$$

In lesson 16 we saw that the classification function can be decomposed into two parts: $d(g_k(X))$ and $g_k(X)$:

$$\vec{g}(\vec{X}) = \begin{pmatrix} g_1(\vec{X}) \\ g_2(\vec{X}) \\ \dots \\ g_K(\vec{X}) \end{pmatrix}$$ A set of discriminant functions : $R^D \rightarrow R^K$

d() : a decision function $R^K \rightarrow \{\omega_K\}$

We derived the canonical form for the discriminant function.

$$g_k(\vec{X}) = \vec{X}^T D_k \vec{X} + \vec{W}_k^T \vec{X} + b_k$$

where: $D_k = -\frac{1}{2} C_k^{-1}$

$\vec{W}_k = -2C_k^{-1}\vec{\mu}_k$

and $b_k = -\frac{1}{2}\vec{\mu}_k{}^T C_k^{-1}\vec{\mu}_k - Log\{\det(C_k)\} + Log\{p(\omega_k)\}$

A set of K discrimination functions $g_k(\vec{X})$ partitions the space $\vec{X}$ into a disjoint set of regions with quadratic boundaries. At the boundaries between classes:

$g_i(\vec{X}) - g_j(\vec{X}) = 0$

In our last lesson we saw that in many cases the quadratic term can be ignored and the partitions take on the form of hyper-surfaces. In this case, the discrimination function can be reduced to a linear equation.

$$g_k(\vec{X}) = \vec{W}_k^T \vec{X} + b_k$$

This is very useful because there are simple powerful techniques to calculate the terms of such an equation.

**Pattern detectors as linear classifiers.**

Linear classifiers are also widely used to define pattern "detectors". This is widely 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.

For example:

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

Wednesday we saw the first 2. Here we look at least-squares

# 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 function $g(\vec{X}) = \vec{W}^T \vec{X} + b$

Such that we minimize 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** = $(\vec{X}_1, \vec{X}_2, ..., \vec{X}_M)$ (D row by M columns)

**Y** = $(y_1, y_2, ..., y_M)^T$ (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 $\vec{W}^T \vec{X} > 0$ then $\hat{\omega}_1$ else $\hat{\omega}_2$

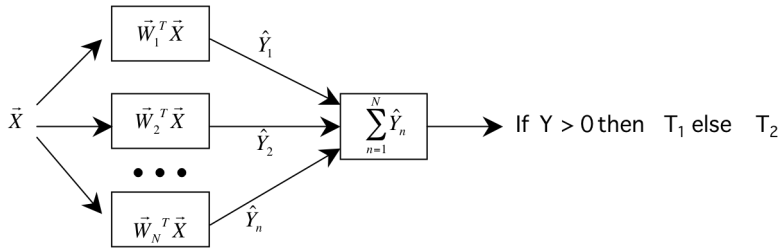
We can also trade False Positives for False negatives using a bias, B

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

# 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.

$$\text{if } \sum_{i=1}^{I}(\vec{W}_i{}^T\vec{X}_m)+b>0 \text{ then } \hat{\omega}_1 \text{ else } \hat{\omega}_2$$

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, $A_i$.

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

**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 misclassified by the committee we will increase its weight so that it carries more weight in the next classifier added.

Recall the committee vote is $\sum_{i=1}^{I}(\vec{W}_i^T \vec{X}_m) > 0$ for class 1 (positive detection).

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

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

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

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

**ROC Curve**

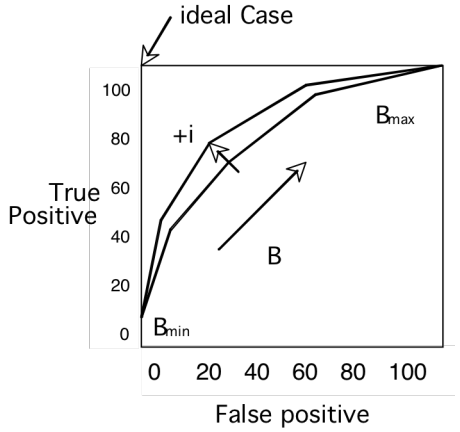
As we saw in lesson 19, The ROC describes the True Positives (TP) and False Positives (FP) for a classifier as a function of the global bias B.

For m = 1 to M:

if $\sum_{i=1}^{I}(\vec{W}_i^T \vec{X}_m) + B > 0$ and $y_m > 0$ then TP=TP+1

if $\sum_{i=1}^{I}(\vec{W}_i^T \vec{X}_m) + B > 0$ and $y_m < 0$ then FP=FP+1

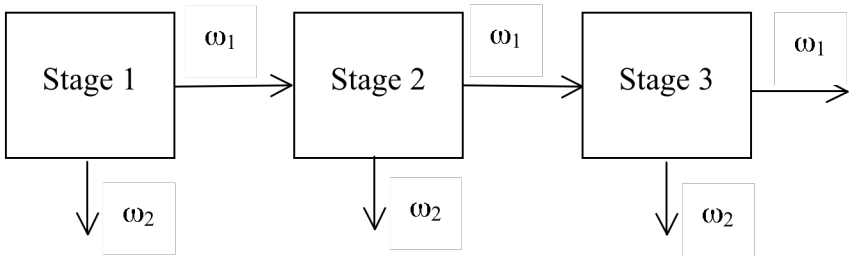
The Boosting theorem states that adding a new boosted classifier to a committee always improves the committee's ROC curve. We can continue adding classifiers until we obtain a desired rate of false positives and false negatives.




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.

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", $\gamma$, 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, $\eta$, 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.

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

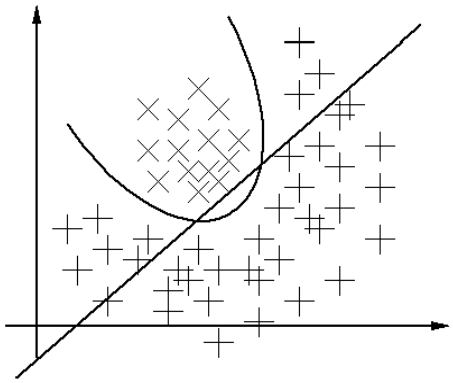
The decision rule is as before :

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

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

# Kernel Methods

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 addiing additional dimensions.



A quadratic suface in a D dimensional space can be transformed 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 $K(X) = W = (x_1, x_2, x_1^2, x_1x_2, x_2^2)$

$\vec{W} = (x_1, x_2, ..., x_D, x_1^2, x_1x_2, x_1x_3, .... x_{D-1}x_D, x_D^2)$

Ainsi, une fonction quadratique en D = 2 dimensions est linéaire en P = 5 dimensions

$$\vec{X} = (x_1, x_2) \qquad K(\vec{X}) = \vec{W} = (x_1, x_2, x_1^2, x_1x_2, x_2^2)$$
$$g(K(\vec{X})) = g(\vec{W}) = aw_1 + b\,w_2 + c\,w_3 + d\,w_4 + e\,w_5$$
$$= ax_1 + b\,x_2 + c\,x_1^2 + d\,x_1x_2 + e\,x_2^2$$

The Normal is often used as a Kernel:

$K(\vec{X}) = \mathcal{N}(\vec{X}; \vec{\mu}, \sigma^2)$