Xi'an Jiaotong-Liverpool University

Pattern Recognition

Lecture 13. Discriminative methods : Linear Discriminant Functions

Dr. Shanshan ZHAO

School of AI and Advanced Computing
Xi'an Jiaotong-Liverpool University

Academic Year 2023-2024

Table of Contents

Notations

- w_0 : a scalar
- w : a vector
- c : denotes the class/label

Outline

Introduction

Optimal classifier

with *pdf* and parameters of the *pdf* are all known.

- Bayesian Decision Theory

Generative methods

with *pdf* or parameters of the *pdf* are **unknown**.

- Parametric Methods
  known *pdf* form, but unknown parameters of the *pdf*
- non-Parametric Methods
  estimate the *pdf* directly from the data, without assumptions about the *pdf* form.

- **major concern of generative methods**: to design classifiers based on probability density or probability functions

Introduction

- From this week, we will focus on the design of linear classifiers, **regardless of the underlying distributions describing the training data**.
- assumption: all feature vectors from the available classes can be classified correctly using a **linear classifier**.
- we shall be concerned with discriminant functions that are either linear in the components of x, or linear in some given set of functions of x.
- they can be optimal if the underlying distributions are cooperative, such as Gaussians having equal covariance.
- Even when they are not optimal, we might be willing to sacrifice some performance in order to gain the **advantage of their simplicity**.

Discriminative methods

Discriminative methods

- Linear Discriminant Functions
  - Hyperplane Geometry
- Logistic Regression Classifier
- Support Vector Machines

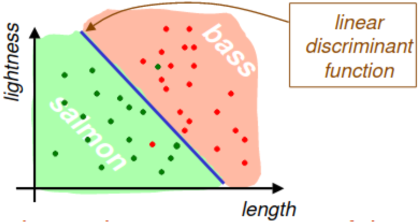
Outline

Role of Linear Discriminant Functions

- A Discriminative Approach, as apposed to Generative approach of Parameter Estimation
- Leads to perceptrons and Artificial Neural Networks
- Leads to Support Vector Machines

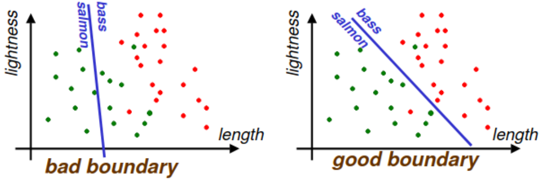
Preliminaries

- No probability distribution (no shape or parameters are known).
- Data with labels.
- The shape of discriminant functions is known.



- Need to estimate parameters of the discriminant functions.
- The problem of finding a linear discriminant function will be formulated as a problem of minimizing a criterion function.
- For classification purposes, the obvious criterion function is sample risk or training error.

LDF: Basic idea



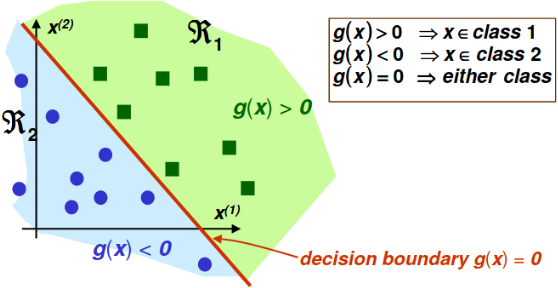
- Have samples from 2 classes $x_1, x_2, \ldots, x_n$.
- Assume 2 classes can be separated by a linear boundary $l(\theta)$ with some unknown parameters θ.
- Fit the "best" boundary to data by optimizing over parameters θ.
  - Minimize classification error on training data is an option

LDF: 2 Classes

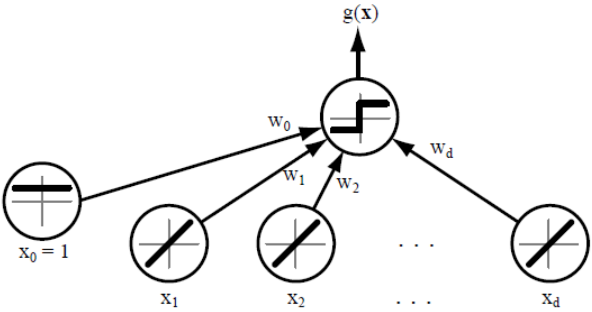
A discriminant function is linear if it can be written as

$$g(x) = w^t x + w_0 \tag{1}$$

- w is called the weight vector, and w_0 called bias or threshold

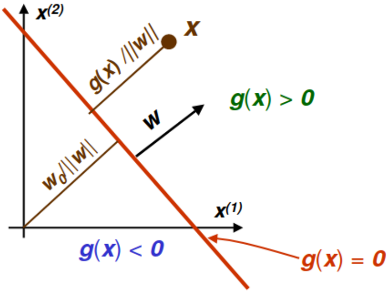


LDF: 2 Classes



A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the output unit sums all these products and emits a +1 if $w^t x + w_o > 0$ or a -1 otherwise.

LDF: 2 Classes



$$g(x) = w^t x + w_0 = 0 \tag{2}$$

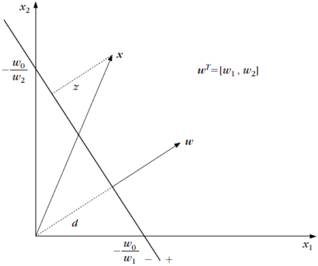
- w determines orientation of the decision hyperplane
- w_0 determines location of the decision surface
- in the function $g(x)$, w_1 is positive.

LDF: 2 Classes

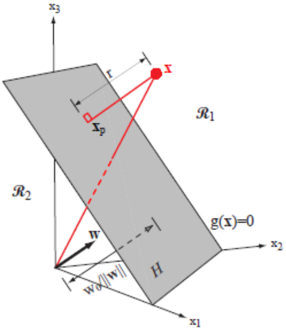
Decision boundary $g(x) = w^t x + w_0 = 0$ is

- a point in 1D
- a line in 2D
- a plane in 3D

LDF: 2 Classes



(a) 2D



(b) 3D

Figure 1: The linear decision boundary H, where $g(x) = w^t x + w_0 = 0$, separates the feature space into two half-spaces $\mathcal{R}_1$ (where $g(x) > 0$) and $\mathcal{R}_2$ (where $g(x) < 0$).

LDF: Multiple Classes

- We have M classes
- Define M linear discriminant functions

$$g_i(x) = w_i^T x + w_{i0} \quad i = 1, \ldots, M \tag{3}$$

- Given x, assign class c_i if

$$g_i(x) \geq g_j(x) \quad \forall j \neq i \tag{4}$$

- Such classifier is called a **linear machine**
- A linear machine divides the feature space into M decision regions, with $g_i(x)$ being the largest discriminant if x is in the regions R_i.

LDF: Augmented feature vector

- Linear discriminant function: $g(x) = w^T x + w_0$
- It can be rewritten as :

$$g(x) = \begin{bmatrix} w_0 & w^T \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix} = a^T y = g(y) \tag{5}$$

- y is called the augmented feature vector
- Add a dummy dimension to get a completely equivalent new Homogeneous problem

old problem: $g(x) = w^T x + w_0$

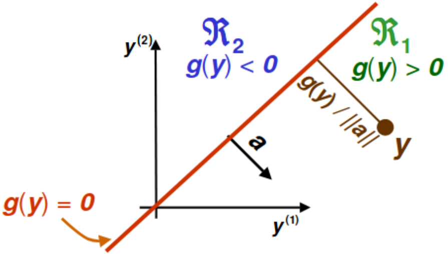
$$\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

new problem: $g(y) = a^T y$

$$\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

LDF: Augmented feature vector

Given samples $x_1, x_2, ..., x_n$, convert them to augmented samples $y_1, y_2, ..., y_n$ by adding a new dimension of value 1.



- This mapping from d-dimensional x-space to $(d+1)$-dimensional y-space is mathematically trivial but nonetheless quite convenient.
- The addition of a constant component to x preserves all distance relationships among samples.
- The resulting y vectors all lie in a d-dimensional subspace, which is the x-space itself. The hyperplane decision surface $\hat{H}$ defined by at $y = 0$ passes through the origin in y-space, even though the corresponding hyperplane H can be in any position in x-space.
- By using this mapping we reduce the problem of finding a weight vector w and a threshold weight w_0 to the problem of finding a single weight vector $\boldsymbol{a}$.

LDF: Train Error

- For the rest of lecture, we assume we have 2 classes
- Samples $y_1, \dots, y_n$ belongs to either class 1 or class 2.
- Our goal is to use these samples to determine weights a in the discriminant function $g(y) = a^T y$
- We need to decide which criterion for determining a.
  For now, suppose we want to minimize the training error, which means the number of misclassified samples $y_1, \dots, y_n$
- Recall that
  - $g(y_i) > 0 \Rightarrow y_i$ classified c_1
  - $g(y_i) < 0 \Rightarrow y_i$ classified c_2
- The training error is 0 if
  - $g(y_i) > 0 \quad \forall y_i \in c_1$
  - $g(y_i) < 0 \quad \forall y_i \in c_2$

LDF: Problem "Normalization"

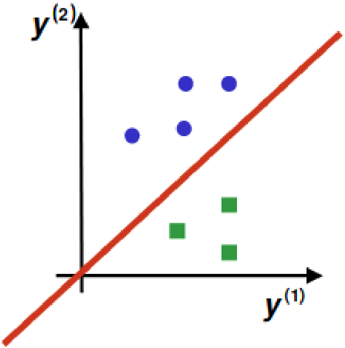
- Equivalently, training error is 0 if

$$\begin{cases} a^T y_i > 0 & \forall y_i \in c_1 \\ a^T(-y_i) > 0 & \forall y_i \in c_2 \end{cases}$$

- This suggest problem "normalization"
  - Replace all examples from class c_2 by their negative
    $y_i \Rightarrow -y_i \quad \forall y_i \in c_2$
  - seek weight vector $\boldsymbol{a}$
    $a^T y_i > 0 \quad \forall y_i$
    - If such $\boldsymbol{a}$ exists, it is called a separating or solution vector
    - original samples $x_1, ..., x_n$ can indeed be separated by a line

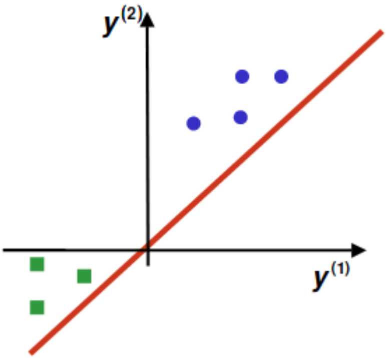
LDF: Problem "Normalization"

Before Normalization



Seek a hyperplane that separates patterns from different categories

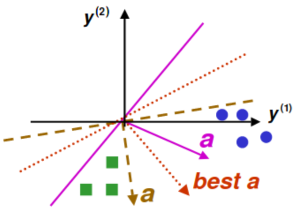
After Normalization

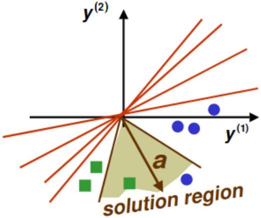


Seek a hyperplane that puts normalized patterns on the same side (should be positive)

LDF: Solution Region

- Find weight vector a, for all samples $y_1, ..., y_n$:
  $a^T y_i = \sum_{k=0}^{d} a_k y_i^{(k)} > 0$





- In general, there are many such solutions $\boldsymbol{a}$

Outline

Optimization

We need a criterion function $J(a)$

$J(a)$ is minimized if a is a solution vector.

Regarding the exact form of J(a), we will talk about it later.

This reduces our problem to one of minimizing a scalar function :

a problem that can often be solved by a gradient descent procedure.

Optimization: Gradient Descent

Basic idea of Gradient Descent

Gradient Descent

For minimizing any function $J(x)$ set k = 1 and $x^{(1)}$ to some initial guess for the weight vector

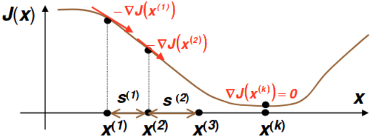
while $\eta^{(k)}|\nabla J(x^{(k)})| > \epsilon$ **do**

choose learning rate $\eta^{(k)}$

$x^{(k+1)} = x^{(k)} - \eta^{(k)}\nabla J(x^{(k)})$

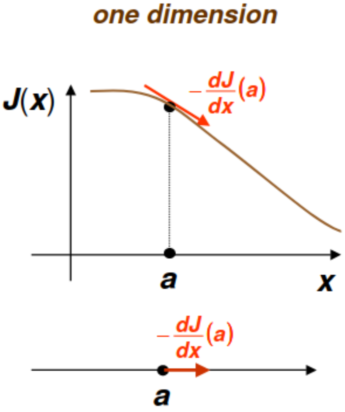
$k = k + 1$

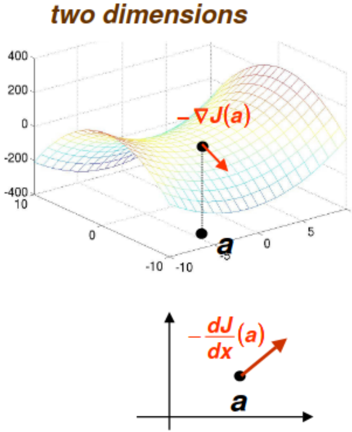
end



Optimization: Gradient Descent

- Gradient $\nabla J(x)$ points in direction of steepest increase of $J(x)$, and $-\nabla J(x)$ in direction of steepest decrease





Outline

LDF: Criterion Function

- Find weight vector ***a*** , for all samples $y_1, ..., y_n$

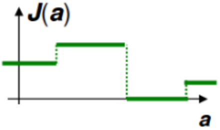
$$a^T y_i = \sum_{j=0}^{d} a_j y_{ij} > 0 \tag{6}$$

- Need criterion function ***J(a)*** which is minimized when ***a*** is a solution vector
- Let Y_M be the set of samples misclassified by ***a***

$$Y_M(a) = \{\text{sample } y_i \quad s.t. \quad a^T y < 0\} \tag{7}$$

- First natural choice: number of misclassified samples

$$J(a) = |Y_M(a)| \tag{8}$$

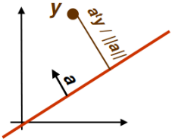


LDF: Perceptron Criterion Function

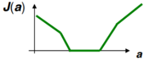
- Better choice: ***Perception*** criterion function

$$J_p(a) = \sum_{y \in Y_M} (-a^T y)$$

- If y is misclassified, $a^T y \leq 0$, so that $J_p(a) \geq 0$
- $J_p(a)$ is $||a||$ times the sum of distances of misclassified samples to decision boundary (figure a).
- $J_p(a)$ is piecewise linear and thus suitable for gradient descent (figure b).



(a)



(b)

LDF: Perceptron Batch Rule

- Perception criterion function

$$J_p(a) = \sum_{y \in Y_M} (-a^T y)$$

- Gradient of $J_p(a)$ is $\nabla J_p(a) = \sum_{y \in Y_M} (-y)$
  - Y_M are samples misclassified by $a^{(k)}$
  - It is not possible to solve $\nabla J_p(a) = 0$ analytically because of Y_M
- Update rule for gradient descent : $x^{(k+1)} = x^k - \eta^{(k)} \nabla J(x)$
- Thus gradient decent batch update rule for $J_p(a)$ is

$$a^{(k+1)} = a^{(k)} + \eta^{(k)} \sum_{y \in Y_M} y \tag{9}$$

- It is called batch rule here because it is based on all misclassified samples

LDF: Perceptron Single Sample Rule

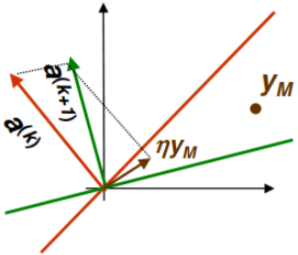
- In comparison, gradient decent single sample rule for $J_p(a)$ is :

$$a^{(k+1)} = a^{(k)} + \eta^{(k)} y_M \quad (10)$$

  - note that y_M is one sample misclassified by $a^{(k)}$
  - must have a consistent way of visiting samples
- Geometric Interpretation
  - y_M misclassified by $a^{(k)}$

$$(a^{(k)})^T y_M \leq 0$$

  - y_M is on the wrong side of decision hyperplane
  - adding ηy_M to $\boldsymbol{a}$ moves new decision hyperplane in the right direction with



LDF: Perceptron Single Sample Rule

$$a^{(k+1)} = a^{(k)} + \eta^{(k)} y_M$$

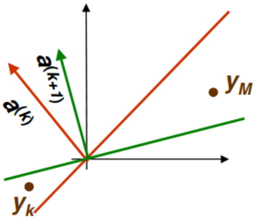


Figure 3: η is too large, previously correctly classified sample y_k is now misclassified

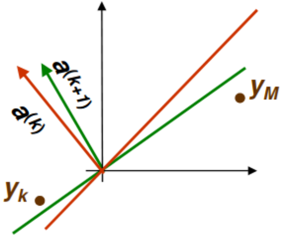


Figure 4: η is too small, y_M is still misclassified

LDF: Perceptron Example

	features				grade
name	**good attendance?**	**tall?**	**sleeps in class?**	**chews gum?**	
Jane	yes(1)	yes(1)	no(-1)	no(-1)	A
Steve	yes(1)	yes(1)	yes(1)	yes(1)	F
Mary	no(-1)	no(-1)	no(-1)	yes(1)	F
Peter	yes(1)	no(-1)	no(-1)	yes(1)	A

- **class 1**: students who get grade A
- **class 2**: students who get grade F

LDF: Perceptron Example

	features					grade
name	extra	**good attendance?**	**tall?**	**sleeps in class?**	**chews gum?**	
Jane	1	yes(1)	yes(1)	no(-1)	no(-1)	A
Steve	1	yes(1)	yes(1)	yes(1)	yes(1)	F
Mary	1	no(-1)	no(-1)	no(-1)	yes(1)	F
Peter	1	yes(1)	no(-1)	no(-1)	yes(1)	A

- Convert samples $x_1, .., x_n$ to augmented samples $y_1, ..., y_n$ by adding a new dimension of value 1.

LDF: Perceptron Example

	features					grade
name	extra	**good attendance?**	**tall?**	**sleeps in class?**	**chews gum?**	
Jane	1	yes(1)	yes(1)	no(-1)	no(-1)	A
Steve	-1	yes(-1)	yes(-1)	yes(-1)	yes(-1)	F
Mary	-1	no(1)	no(1)	no(1)	yes(-1)	F
Peter	1	yes(1)	no(-1)	no(-1)	yes(1)	A

- Replace all samples from class c_2 by their negative
  $y_i \Rightarrow -y_i \quad y_i \in c_2$
- Seek weight vector $\boldsymbol{a}$ s.t. $a^T y_i > 0 \quad \forall y_i$

LDF: Perceptron Example

	features					grade
name	extra	**good attendance?**	**tall?**	**sleeps in class?**	**chews gum?**	
Jane	1	yes(1)	yes(1)	no(-1)	no(-1)	A
Steve	-1	yes(-1)	yes(-1)	yes(-1)	yes(-1)	F
Mary	-1	no(1)	no(1)	no(1)	yes(-1)	F
Peter	1	yes(1)	no(-1)	no(-1)	yes(1)	A

- Sample is misclassified if $a^T y_i = \sum_{j=0}^{4} a_j y_{ij} < 0$

  (i is the numbering of the sample, j is the numbering of the dimension)
- Gradient descent single sample rule : $a^{(k+1)} = a^{(k)} + \eta^{(k)} \sum_{y \in Y_M} y$
- Here we set a fixed learning rate to $\eta^{(k)} = 1$

$$a^{(k+1)} = a^{(k)} + y_M$$

LDF: Perceptron Example

- set initial weights $a^{(1)} = [0.25, 0.25, 0.25, 0.25, 0.25]$
- visit all samples sequentially, modifying the weights for after finding a misclassified example

name	$a^T y$	misclassified?
Jane	0.25*1+0.25*1+0.25*1+0.25*(-1)+0.25*(-1) >0	no
Steve	0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1)<0	yes

- new weights

$$\begin{aligned} a^{(2)} = a^{(1)} + y_M &= [0.25 \quad 0.25 \quad 0.25 \quad 0.25 \quad 0.25]+ \\ &+ [-1 \quad -1 \quad -1 \quad -1 \quad -1] \\ &= [-0.75 \quad -0.75 \quad -0.75 \quad -0.75 \quad -0.75] \end{aligned}$$

LDF: Perceptron Example

$$a^{(2)} = [-0.75 \quad -0.75 \quad -0.75 \quad -0.75 \quad -0.75]$$

name	$a^T y$	misclassified?
Mary	-0.75*(-1)-0.75*1-0.75*1-0.75*1-0.75*(-1) <0	yes

- new weights

$$\begin{aligned} a^{(3)} = a^{(2)} + y_M &= [-0.75 \quad -0.75 \quad -0.75 \quad -0.75 \quad -0.75]+ \\ &+ [-1 \quad 1 \quad 1 \quad 1 \quad -1] \\ &= [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75] \end{aligned}$$

LDF: Perceptron Example

$$a^{(3)} = [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75]$$

name	$a^T y$	misclassified?
Peter	-1.75*1+0.25*1+0.25*(-1)+0.25*(-1)-0.75*1 <0	yes

- new weights

$$\begin{aligned} a^{(4)} = a^{(3)} + y_M &= [-1.75 \quad 0.25 \quad 0.25 \quad 0.25 \quad -1.75] + \\ &+ [1 \quad 1 \quad -1 \quad -1 \quad 1] \\ &= [-0.75 \quad 1.25 \quad -0.75 \quad -0.75 \quad -0.75] \end{aligned}$$

LDF: Perceptron Example

$$a^{(4)} = [-0.75 \quad 1.25 \quad -0.75 \quad -0.75 \quad -0.75]$$

name	$a^T y$	wrong?
Jane	-0.75*1+1.25*1-0.75*1-0.75*(-1)-0.75*(-1) >0	no
Steve	-0.75*(-1)+1.25*(-1)-0.75*(-1)-0.75*(-1)-0.75*(-1) >0	no
Mary	-0.75*(-1)+1.25*1-0.75*1-0.75*1-0.75*(-1) >0	no
Peter	-0.75*1+1.25*1-0.75*(-1)-0.75*(-1)-0.75*1 >0	no

- The discriminant function is

$$g(y) = -0.75 * y^{(0)} + 1.25 * y^{(1)} - 0.75 * y^{(2)} - 0.75 * y^{(3)} - 0.75 * y^{(4)}$$

- Converting back to the original features x:

$$g(x) = -0.75 + 1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)}$$

LDF: Perceptron Example

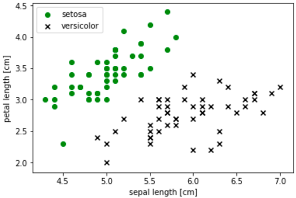
- Converting back to the original features x:

$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} > 0.75 \Rightarrow gradeA$$
$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} < 0.75 \Rightarrow gradeF$$

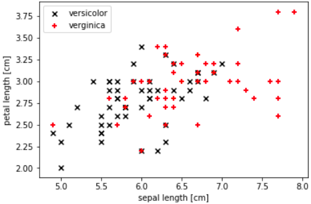
- This is just one possible solution vector
- If we started with weight $a^{(1)} = [0, 0.5, 0.5, 0, 0]$, the solution would be different: $[-1, 1.5, -0.5, -1, -1]$

Perceptron Criterion Function

- If classes are linearly separable, the perceptron rule is guaranteed to converge to a valid solution
- However, if the two classes are not linearly separable, the perceptron rule will not converge.
  - Since there is no weight vector $\boldsymbol{a}$ can correctly classify every sample in a non-separable dataset, the corrections in the perceptron rule will never cease.



(a) Linearly Separable Data (IRIS Dataset)



(b) Linearly Non-Separable Data (IRIS Dataset)

Outline

Minimum Squared Error Procedures

The classical MSE criterion provides an alternative to the perceptron rule

Perceptron

- 1. Focused on the **misclassified** samples
- 2. seek a vector $\boldsymbol{a}$ making all inner product $a^T y_i$ positive
- 3. Try to find the solution to a set of linear inequalities

MSE

- 1. Involves **all** the samples
- 2. Seek a vector $\boldsymbol{a}$ making $a^T y_i = b_i$, where b_i is some arbitrarily specified positive constants
- 3. Find the solution to a set of linear equations

Minimum Squared Error Procedures

- The treatment of simultaneous linear equations is simplified by introducing matrix notation.
- Let Y be the n-by-$\hat{d}$ matrix ($\hat{d} = d + 1$) whose ith row is the vector y_i^T
- let b be the column vector $b = (b_1, ..., b_n)^T$
- Then our problem is to find a weight vector a satisfying

$$\begin{bmatrix} Y_{10} & Y_{11} & \cdots & Y_{1d} \\ Y_{20} & Y_{21} & \cdots & Y_{2d} \\ \vdots & \vdots & & \vdots \\ Y_{n0} & Y_{n1} & \cdots & Y_{nd} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_d \end{bmatrix}$$

or

$$Ya = b$$

Minimum Squared Error Procedures

- If Y were nonsingular, we could write $a = Y^{-1}b$ and obtain a formal solution at once.
- However, Y is rectangular, usually with more rows than columns. When there are more equations than unknowns, a is overdetermined, and ordinarily no exact solution exists.
- However, we can seek a weight vector a that minimizes some function of the error between Ya and b.
- If we define the error vector e by

$$e = Ya - b$$

- Then one approach is to try to minimize the squared length of the error vector. This is equivalent to minimizing the sum-of-squared-error criterion function

$$J_s(a) = ||Ya - b||^2 = \sum_{i=1}^{n}(a^T y_i - b_i)^2$$

Minimum Squared Error Procedures

The problem of minimizing the sum of squared error is a classical one. It can be solved by a gradient search procedure. A simple closed-form solution can also be found by forming the gradient

$$\nabla J_s = \sum_{i=1}^{n} 2(a^T y_i - b_i)y_i = 2Y^T(Ya - b)$$

and setting it equal to zero. This yields the necessary condition

$$Y^T Ya = Y^T b$$

If matrix $Y^T Y$ is square and nonsingular, we can solve a uniquely

$$a = (Y^T Y)^{-1} Y^T b$$

Minimum Squared Error

The MSE solution depends on the margin vector b, and we shall see that different choices for b give the solution different properties. If b is fixed arbitrarily, there is no reason to believe that the MSE solution yields a separating vector in the linearly separable case. However,it is reasonable to hope that by minimizing the squared-error criterion function we might obtain a useful discriminant function in both the separable and the non-separable cases.

MSE example

Compute the perceptron and MSE solution for the dataset

- $X1 = [\,(1{,}6), (7{,}2), (8{,}9), (9{,}9)]$
- $X2 = [\,(2{,}1), (2{,}2), (2{,}4), (7{,}1)]$

Perceptron leaning

- Assume $\eta = 0.1$ and an online update rule
- Assume $a(0) = [0.1, 0.1, 0.1]$
- SOLUTION
  - Normalize the dataset

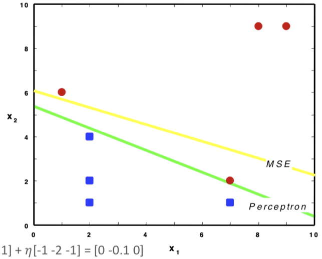
$$Y = \begin{bmatrix} 1 & 1 & 6 \\ 1 & 7 & 2 \\ 1 & 8 & 9 \\ 1 & 9 & 9 \\ -1 & -2 & -1 \\ -1 & -2 & -2 \\ -1 & -2 & -4 \\ -1 & -7 & -1 \end{bmatrix}$$

  - Iterate through all the examples and update $a(k)$ on the ones that are misclassified
    - Y(1) ⇒ [1 1 6]*[0.1 0.1 0.1]^T>0 ⇒ no update
    - Y(2) ⇒ [1 7 2]*[0.1 0.1 0.1]^T>0 ⇒ no update

      ...
    - Y(5) ⇒ [-1 -2 -1]*[0.1 0.1 0.1]^T<0 ⇒ update a(1) = [0.1 0.1 0.1] + η[-1 -2 -1] = [0 -0.1 0]
    - Y(6) ⇒ [-1 -2 -2]*[0 -0.1 0]^T>0 ⇒ no update

      ....
    - Y(1) ⇒ [1 1 6]*[0 -0.1 0]^T<0 ⇒ update a(2) = [0 -0.1 0] + η[1 1 6] = [0.1 0 0.6]
    - Y(2) ⇒ [1 7 2]*[0.1 0 0.6]^T>0 ⇒ no update

      ...



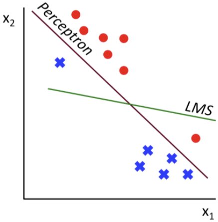
  - In this example, the perceptron rule converges after 175 iterations to $a = [-3.5\ 0.3\ 0.7]$
  - To convince yourself this is a solution, compute Ya (you will find out that all terms are non-negative)

MSE

- The MSE solution is found in one shot as $a = (Y^TY)^{-1}Y^Tb = [-1.1870\ \ 0.0746\ 0.1959]$
  - For the choice of targets b = [1 1 1 1 1 1 1 1]^T
  - As you can see in the figure, the MSE solution misclassifies one of the samples

Perceptron vs. MSE

- **Percepton criterion**
  - The perceptron rule always find a solution if the classes are linearly separable, but does not converge if the classes are non-separable.
- **MSE criterion**
  - The MSE solution has guaranteed convergence, but it may not find a separating hyperplane if classes are linearly separable.
    - Notice that MSE tries to minimize the sum of the squares of distances of the training data to the separating hyperplane, as opposed to finding this hyperplane.



Summary

- Linear Discriminant functions, decision surfaces (geometry)
- Gradient descent procedures
- Criterion function
    - perception criterion function
    - minimum squared error

Thank You !

Q & A