CS434b/654b : Pattern Recognition
Prof. Olga Veksler

Lecture 9

Linear Discriminant Functions

Today

- Linear Discriminant Functions
  - Introduction
  - 2 classes
  - Multiple classes
  - Optimization with gradient descent
  - Perceptron Criterion Function
    - Batch perceptron rule
    - Single sample perceptron rule

Announcements

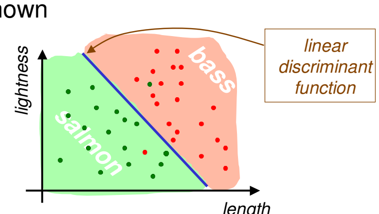
- Final project proposal due March 8
  - 1-2 paragraph description
- Final project progress report
  - Meet with me the week of March 20-24
- Final project due April 11

Linear discriminant functions on Road Map

a lot is known

- No probability distribution (no shape or parameters are known)
- Labeled data 

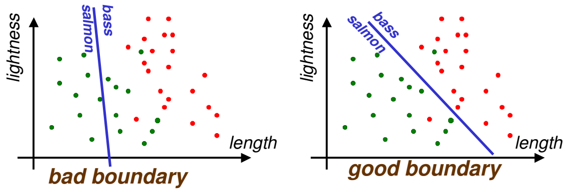
- The shape of discriminant functions is known



- Need to estimate parameters of the discriminant function (parameters of the line in case of linear discriminant)

little is known

Linear Discriminant Functions: Basic Idea



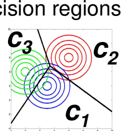
- Have samples from 2 classes $x_1, x_2, \ldots, x_n$
- Assume 2 classes can be separated by a linear boundary $l(\theta)$ with some unknown parameters θ
- Fit the "best" boundary to data by optimizing over parameters θ
- What is best?
  - Minimize classification error on training data?
    - Does not guarantee small testing error

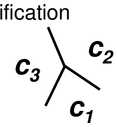
LDF: Introduction

- Discriminant functions can be more general than linear
- For now, we will study linear discriminant functions
  - Simple model (should try simpler models first)
  - Analytically tractable
- Linear Discriminant functions are optimal for Gaussian distributions with equal covariance
- May not be optimal for other data distributions, but they are very simple to use
- Knowledge of class densities is not required when using linear discriminant functions
  - we can say that this is a non-parametric approach

Parametric Methods vs. Discriminant Functions

Parametric Methods	Discriminant Functions
Assume the shape of density for classes is known $p_1(x\|\theta_1)$, $p_2(x\|\theta_2)$,…	Assume discriminant functions are or known shape $l(\theta_1)$, $l(\theta_2)$, with parameters $\theta_1, \theta_2, \ldots$
Estimate $\theta_1, \theta_2, \ldots$ from data	Estimate $\theta_1, \theta_2, \ldots$ from data
Use a Bayesian classifier to find decision regions	Use discriminant functions for classification





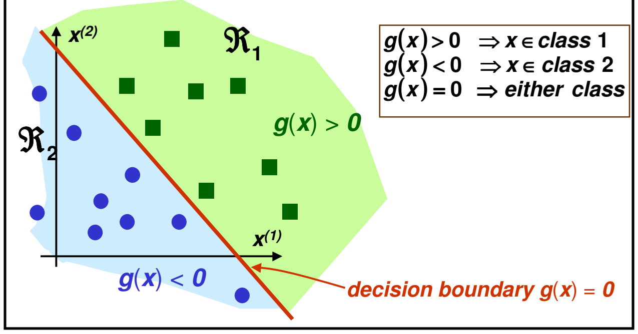
- In theory, Bayesian classifier minimizes the risk
  - In practice, do not have confidence in assumed model shapes
  - In practice, do not really need the actual density functions in the end
- Estimating accurate density functions is much harder than estimating accurate discriminant functions
- Some argue that estimating densities should be skipped
  - Why solve a harder problem than needed ?

LDF: 2 Classes

- A discriminant function is linear if it can be written as

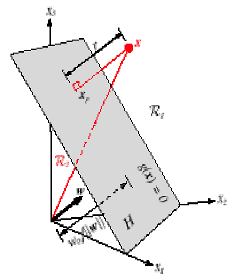
$$g(x) = w^t x + w_0$$

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



LDF: 2 Classes

- Decision boundary $g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = 0$ is a ***hyperplane***
  - set of vectors x which for some scalars $\alpha_0, \ldots, \alpha_d$ satisfy $\alpha_0 + \alpha_1 x^{(1)} + \ldots + \alpha_d x^{(d)} = 0$
- A hyperplane is
  - a point in 1D
  - a line in 2D
  - a plane in 3D



LDF: 2 Classes

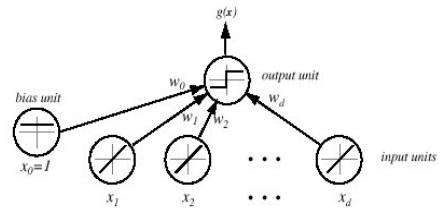
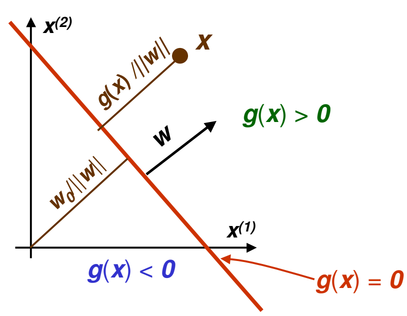


FIGURE 5.1. A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the effective input at the output unit is the sum all these products, $\sum w_i x_i$. We show in each unit its effective input-output function. Thus each of the d input units is linear, emitting exactly the value of its corresponding feature value. The single bias unit unit always emits the constant value 1.0. The single output unit emits a +1 if $\mathbf{w}^t\mathbf{x} + w_0 > 0$ or a −1 otherwise. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

LDF: 2 Classes

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

- $\mathbf{w}$ determines orientation of the decision hyperplane
- w_0 determines location of the decision surface



LDF: Many Classes

- Suppose we have m classes
- Define m linear discriminant functions

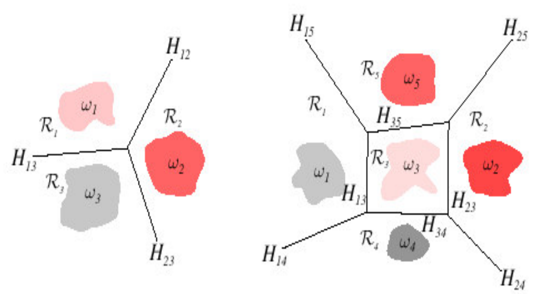
$$g_i(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} + w_{i0} \qquad i = 1, \ldots, m$$

- Given $\mathbf{x}$, assign class c_i if

$$g_i(\mathbf{x}) \geq g_j(\mathbf{x}) \qquad \forall j \neq i$$

- Such classifier is called a ***linear machine***
- A linear machine divides the feature space into c decision regions, with $g_i(\mathbf{x})$ being the largest discriminant if $\mathbf{x}$ is in the region R_i

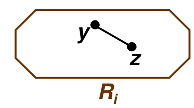
LDF: Many Classes



LDF: Many Classes

- Decision regions for a linear machine are ***convex***

$$y, z \in R_i \Rightarrow \alpha y + (1-\alpha) z \in R_i$$



$$\forall j \neq i \quad g_i(y) \geq g_j(y) \text{ and } g_i(z) \geq g_j(z) \Leftrightarrow$$
$$\Leftrightarrow \forall j \neq i \quad g_i(\alpha y + (1-\alpha) z) \geq g_j(\alpha y + (1-\alpha) z)$$

- In particular, decision regions must be spatially contiguous

R_j is a valid decision region

R_j is not a valid decision region

LDF: Many Classes

- For a two contiguous regions R_i and R_j; the boundary that separates them is a portion of hyperplane H_{ij} defined by:

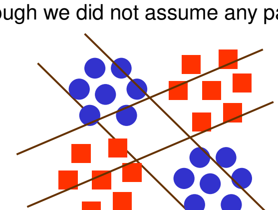
$$g_i(x) = g_j(x) \quad \Leftrightarrow w_i^t x + w_{i0} = w_j^t x + w_{j0}$$
$$\Leftrightarrow (w_i - w_j)^t x + (w_{i0} - w_{j0}) = 0$$

- Thus $w_i - w_j$ is normal to H_{ij}
- And distance from x to H_{ij} is given by

$$d(x, H_{ij}) = \frac{g_i(x) - g_j(x)}{\|w_i - w_j\|}$$

LDF: Many Classes

- Thus applicability of linear machine to mostly limited to unimodal conditional densities $p(x|\theta)$
  - even though we did not assume any parametric models
- Example:



  - need non-contiguous decision regions
  - thus linear machine will fail

LDF: Augmented feature vector

- Linear discriminant function: $g(x) = w^t x + w_0$
- Can rewrite it: $g(x) = \underbrace{[w_0 \;\; w^t]}_{\text{new weight vector } a} \underbrace{\begin{bmatrix} 1 \\ x \end{bmatrix}}_{\text{new feature vector } y} = a^t y = g(y)$
- y is called the ***augmented feature vector***
- Added a dummy dimension to get a completely equivalent new ***homogeneous*** problem

old problem	*new problem*
$g(x) = w^t x + w_0$	$g(y) = a^t y$
$\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$	$\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$

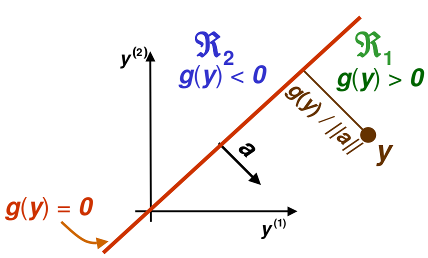
LDF: Training Error

- For the rest of the lecture, assume we have 2 classes
- Samples $y_1, \ldots, y_n$ some in class 1, some in class 2
- Use these samples to determine weights a in the discriminant function $g(y) = a^t y$
- What should be our criterion for determining a?
  - For now, suppose we want to minimize the training error (that is the number of misclassifed samples $y_1, \ldots, y_n$)
- Recall that $g(y_i) > 0 \Rightarrow y_i$ *classified* c_1; $g(y_i) < 0 \Rightarrow y_i$ *classified* c_2
- Thus training error is 0 if $\begin{cases} g(y_i) > 0 & \forall y_i \in c_1 \\ g(y_i) < 0 & \forall y_i \in c_2 \end{cases}$

LDF: Augmented feature vector

- Feature augmenting is done for simpler notation
- From now on we always assume that we have augmented feature vectors
  - Given samples $x_1, \ldots, x_n$ convert them to augmented samples $y_1, \ldots, y_n$ by adding a new dimension of value 1

$$y_i = \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$



LDF: Problem "Normalization"

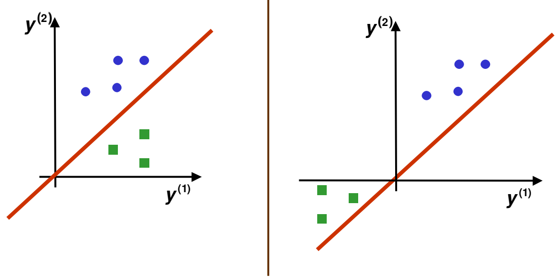
- Thus training error is 0 if $\begin{cases} a^t y_i > 0 & \forall y_i \in c_1 \\ a^t y_i < 0 & \forall y_i \in c_2 \end{cases}$
- Equivalently, training error is 0 if

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

- This suggest problem "normalization":
  1. Replace all examples from class c_2 by their negative
     $$y_i \rightarrow -y_i \qquad \forall y_i \in c_2$$
  2. Seek weight vector a s.t.
     $$a^t y_i > 0 \qquad \forall y_i$$
  - If such a exists, it is called a *separating* or *solution* vector
  - Original samples $x_1, \ldots, x_n$ can indeed be separated by a line then

LDF: Problem "Normalization"

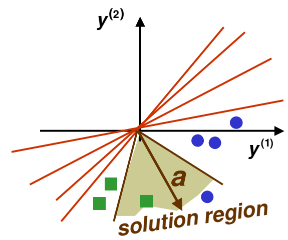
before normalization | *after "normalization"*



Seek a hyperplane that separates patterns from different categories ⇐ Seek hyperplane that puts *normalized* patterns on the same (positive) side

LDF: Solution Region

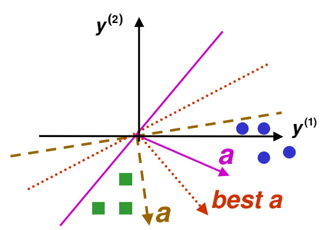
- ***Solution region*** for a: set of all possible solutions
  - defined in terms of normal a to the separating hyperplane



LDF: Solution Region

- Find weight vector a s.t. for all samples $y_1, \ldots, y_n$

$$a^t y_i = \sum_{k=0}^{d} a_k y_i^{(k)} > 0$$



- In general, there are many such solutions a

Optimization

- Need to minimize a function of many variables

$$J(x) = J(x_1, \ldots, x_d)$$

- We know how to minimize $J(x)$
  - Take partial derivatives and set them to zero

$$\begin{bmatrix} \frac{\partial}{\partial x_1} J(x) \\ \vdots \\ \frac{\partial}{\partial x_d} J(x) \end{bmatrix} = \nabla J(x) = 0$$

gradient

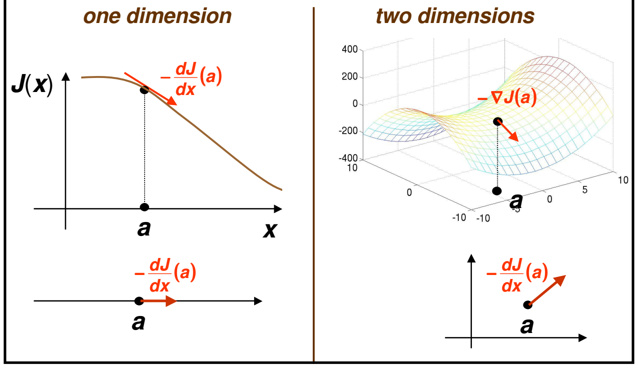
- However solving analytically is not always easy
  - Would you like to solve this system of nonlinear equations?

$$\begin{cases} \sin(x_1^2 + x_2^3) + e^{x_4^2} = 0 \\ \cos(x_1^2 + x_2^3) + \log(x_5^3)^{x_4^2} = 0 \end{cases}$$

  - Sometimes it is not even possible to write down an analytical expression for the derivative, we will see an example later today

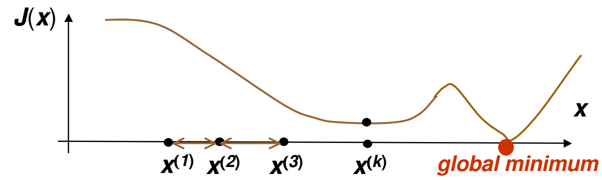
Optimization: Gradient Descent

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



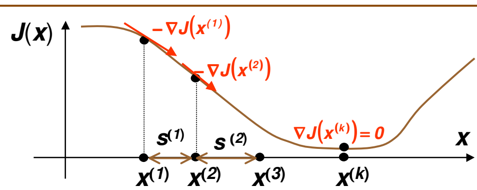
Optimization: Gradient Descent

- Gradient descent is guaranteed to find only a local minimum



- Nevertheless gradient descent is very popular because it is simple and applicable to any function

Optimization: Gradient Descent



Gradient Descent for minimizing any function $J(\mathbf{x})$

set $k = 1$ and $\mathbf{x}^{(1)}$ to some initial guess for the weight vector

while $\eta^{(k)} \left| \nabla J\left(\mathbf{x}^{(k)}\right) \right| > \varepsilon$

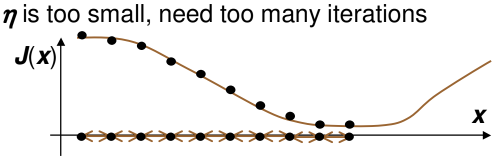
choose *learning rate* $\eta^{(k)}$

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

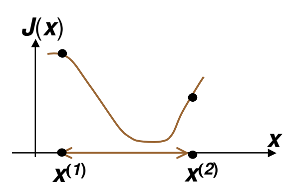
$k = k + 1$

Optimization: Gradient Descent

- Main issue: how to set parameter η (***learning rate***)
- If η is too small, need too many iterations



- If η is too large may overshoot the minimum and possibly never find it (if we keep overshooting)

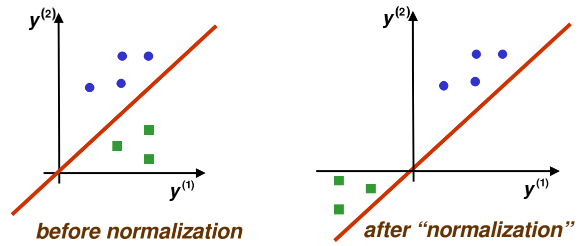


Today

- Continue Linear Discriminant Functions
  - Perceptron Criterion Function
    - Batch perceptron rule
    - Single sample perceptron rule

LDF

- Augmented and "normalized" samples $y_1,\dots, y_n$
- Seek weight vector a s.t. $a^t y_i > 0 \quad \forall y_i$

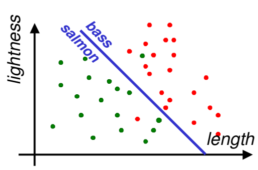


- If such a exists, it is called a *separating* or *solution* vector
- original samples $x_1,\dots, x_n$ can indeed be separated by a line then

LDF: Augmented feature vector

- Linear discriminant function:

$$g(x) = w^t x + w_0$$



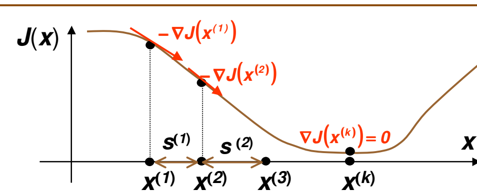
  - need to estimate parameters w and w_0 from data
- Augment samples x to get equivalent homogeneous problem in terms of samples y:

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

- "normalize" by replacing all examples from class c_2 by their negative

$$y_i \rightarrow -y_i \qquad \forall y_i \in c_2$$

Optimization: Gradient Descent



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

Gradient Descent for minimizing any function $J(x)$

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

while $\eta^{(k)}\left|\nabla J\left(x^{(k)}\right)\right| > \varepsilon$

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

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

  $k = k + 1$

LDF: Criterion Function

- Find weight vector $\boldsymbol{a}$ s.t. for all samples $\boldsymbol{y_1},\dots,\boldsymbol{y_n}$

$$a^t y_i = \sum_{k=0}^{d} a_k y_i^{(k)} > 0$$

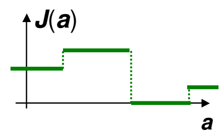
- Need criterion function $\boldsymbol{J}(\boldsymbol{a})$ which is minimized when $\boldsymbol{a}$ is a solution vector
- Let $\boldsymbol{Y_M}$ be the set of examples misclassified by $\boldsymbol{a}$

$$Y_M(a) = \{\textit{sample } y_i \textit{ s.t. } a^t y_i < 0\}$$

- First natural choice: number of misclassified examples

$$J(a) = |Y_M(a)|$$

  - piecewise constant, gradient descent is useless



LDF: Perceptron Batch Rule

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

- Gradient of $\boldsymbol{J_p}(\boldsymbol{a})$ is $\nabla J_p(a) = \sum_{y \in Y_M} (-y)$
  - $\boldsymbol{Y_M}$ are samples misclassified by $\boldsymbol{a}^{(k)}$
  - 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 $\boldsymbol{J_p}(\boldsymbol{a})$ is:

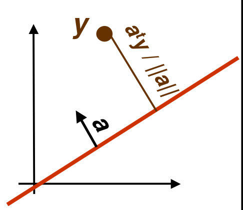
$$a^{(k+1)} = a^{(k)} + \eta^{(k)} \sum_{y \in Y_M} y$$

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

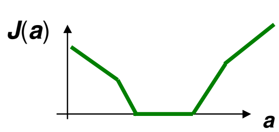
LDF: Perceptron Criterion Function

- Better choice: ***Perceptron*** criterion function

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



- If $\boldsymbol{y}$ is misclassified, $a^t y \le 0$
- Thus $J_p(a) \ge 0$
- $\boldsymbol{J_p}(\boldsymbol{a})$ is -||$\boldsymbol{a}$|| times sum of distances of misclassified examples to decision boundary
- $\boldsymbol{J_p}(\boldsymbol{a})$ is piecewise linear and thus suitable for gradient descent



LDF: Perceptron Single Sample Rule

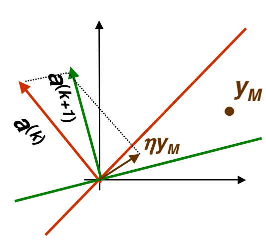
- Thus *gradient decent single sample rule* for $\boldsymbol{J_p}(\boldsymbol{a})$ is:

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

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

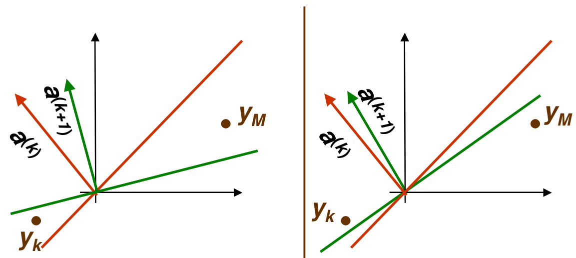
$$(a^{(k)})^t y_M \le 0$$

  - 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 respect to $\boldsymbol{y_M}$



LDF: Perceptron Single Sample Rule

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



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

η is too small, y_M is still misclassified

LDF Example: Augment feature vector

	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 $x_1, \ldots, x_n$ to augmented samples $y_1, \ldots, y_n$ by adding a new dimension of value 1

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: Perform "Normalization"

	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 examples from class c_2 by their negative

$$y_i \to -y_i \qquad \forall y_i \in c_2$$

- Seek weight vector a s.t. $a^t y_i > 0 \qquad \forall y_i$

LDF: Use Single Sample Rule

	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 $a^t y_i = \sum_{k=0}^{4} a_k y_i^{(k)} < 0$
- gradient descent single sample rule: $a^{(k+1)} = a^{(k)} + \eta^{(k)} y_M$
- Set ***fixed*** learning rate to $\eta^{(k)}$= 1: $a^{(k+1)} = a^{(k)} + y_M$

LDF: Gradient decent Example

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

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

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

LDF: Gradient decent Example

- set equal initial weights $a^{(1)}$=[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

$$a^{(2)} = a^{(1)} + y_M = [0.25 \ \ 0.25 \ \ 0.25 \ \ 0.25 \ \ 0.25] +$$
$$+ [-1 \ \ -1 \ \ -1 \ \ -1 \ \ -1] =$$
$$= [-0.75 \ \ -0.75 \ \ -0.75 \ \ -0.75 \ \ -0.75]$$

LDF: Gradient decent Example

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

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

- new weights

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

LDF: Gradient decent Example

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

name	$a^t y$	misclassified?
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

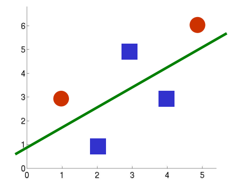
- Thus the discriminant function is

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

- Converting back to the original features ***x***:

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

LDF: Nonseparable Example



- Suppose we have 2 features and samples are:
  - Class 1: [2,1], [4,3], [3,5]
  - Class 2: [1,3] and [5,6]
- These samples are not separable by a line
- Still would like to get approximate separation by a line, good choice is shown in green
  - some samples may be "noisy", and it's ok if they are on the wrong side of the line
- Get y_1, y_2, y_3, y_4 by adding extra feature and "normalizing"

$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

LDF: Gradient decent Example

- Converting back to the original features ***x***:

$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} > 0.75 \Rightarrow grade\ A$$

$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} < 0.75 \Rightarrow grade\ F$$

good attendance ↙ tall ↙ sleeps in class ↘ chews gum ↘

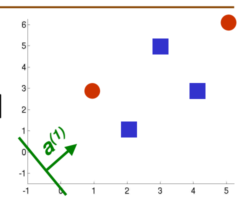
- This is just one possible solution vector
- If we started with weights $a^{(1)}$=[0,0.5, 0.5, 0, 0], solution would be [-1,1.5, -0.5, -1, -1]

$$1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} > 1 \Rightarrow grade\ A$$

$$1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} < 1 \Rightarrow grade\ F$$

  - In this solution, being tall is the least important feature

LDF: Nonseparable Example



- Let's apply Perceptron single sample algorithm
- initial equal weights $a^{(1)} = [1 \;\; 1 \;\; 1]$
  - this is line $x^{(1)} + x^{(2)} + 1 = 0$
- fixed learning rate $\eta = 1$

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

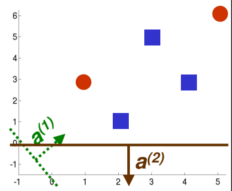
$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $y^t_1 a^{(1)}$ = [1 1 1]*[1 2 1]^t > 0 ✔
- $y^t_2 a^{(1)}$ = [1 1 1]*[1 4 3]^t > 0 ✔
- $y^t_3 a^{(1)}$ = [1 1 1]*[1 3 5]^t > 0 ✔

LDF: Nonseparable Example

$a^{(1)} = [1\ 1\ 1]$ $\quad a^{(k+1)} = a^{(k)} + y_M$

$y_1 = \begin{bmatrix}1\\2\\1\end{bmatrix}$ $y_2 = \begin{bmatrix}1\\4\\3\end{bmatrix}$ $y_3 = \begin{bmatrix}1\\3\\5\end{bmatrix}$ $y_4 = \begin{bmatrix}-1\\-1\\-3\end{bmatrix}$ $y_5 = \begin{bmatrix}-1\\-5\\-6\end{bmatrix}$



- $y^t_4 a^{(1)}$=[1 1 1]*[-1 -1 -3]^t = -5< 0

$$a^{(2)} = a^{(1)} + y_M = [1\ 1\ 1] + [-1\ -1\ -3] = [0\ 0\ -2]$$

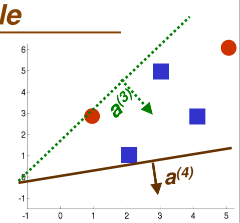
- $y^t_5 a^{(2)}$=[0 0 -2]*[-1 -5 -6]^t = 12 > 0 ✓
- $y^t_1 a^{(2)}$=[0 0 -2]*[1 2 1]^t < 0

$$a^{(3)} = a^{(2)} + y_M = [0\ 0\ -2] + [1\ 2\ 1] = [1\ 2\ -1]$$

LDF: Nonseparable Example

$a^{(4)} = [0\ 1\ -4]$ $\quad a^{(k+1)} = a^{(k)} + y_M$

$y_1 = \begin{bmatrix}1\\2\\1\end{bmatrix}$ $y_2 = \begin{bmatrix}1\\4\\3\end{bmatrix}$ $y_3 = \begin{bmatrix}1\\3\\5\end{bmatrix}$ $y_4 = \begin{bmatrix}-1\\-1\\-3\end{bmatrix}$ $y_5 = \begin{bmatrix}-1\\-5\\-6\end{bmatrix}$



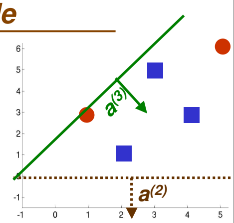
- $y^t_2 a^{(3)}$=[1 4 3]*[1 2 -1]^t =6 > 0 ✓
- $y^t_3 a^{(3)}$=[1 3 5]*[1 2 -1]^t > 0 ✓
- $y^t_4 a^{(3)}$=[-1 -1 -3]*[1 2 -1]^t = 0

$$a^{(4)} = a^{(3)} + y_M = [1\ 2\ -1] + [-1\ -1\ -3] = [0\ 1\ -4]$$

LDF: Nonseparable Example

$a^{(3)} = [1\ 2\ -1]$ $\quad a^{(k+1)} = a^{(k)} + y_M$

$y_1 = \begin{bmatrix}1\\2\\1\end{bmatrix}$ $y_2 = \begin{bmatrix}1\\4\\3\end{bmatrix}$ $y_3 = \begin{bmatrix}1\\3\\5\end{bmatrix}$ $y_4 = \begin{bmatrix}-1\\-1\\-3\end{bmatrix}$ $y_5 = \begin{bmatrix}-1\\-5\\-6\end{bmatrix}$



- $y^t_2 a^{(3)}$=[1 4 3]*[1 2 -1]^t =6 > 0 ✓
- $y^t_3 a^{(3)}$=[1 3 5]*[1 2 -1]^t > 0 ✓
- $y^t_4 a^{(3)}$=[-1 -1 -3]*[1 2 -1]^t = 0

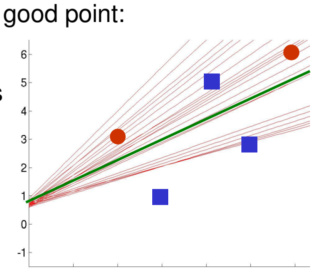
$$a^{(4)} = a^{(3)} + y_M = [1\ 2\ -1] + [-1\ -1\ -3] = [0\ 1\ -4]$$

LDF: Nonseparable Example

- we can continue this forever
  - there is no solution vector ***a*** satisfying for all ***i***

$$a^t y_i = \sum_{k=0}^{5} a_k y_i^{(k)} > 0$$

- need to stop but at a good point:



- solutions at iterations 900 through 915. Some are good some are not.
- How do we stop at a good solution?

LDF: Convergence of Perceptron rules

- If classes are linearly separable, and use fixed learning rate, that is for some constant ***c***, $\eta^{(k)}=c$
  - *both single sample and batch perceptron rules converge to a correct solution* (could be any ***a*** in the solution space)
- If classes are not linearly separable:
  - algorithm does not stop, it keeps looking for solution which does not exist
  - by choosing appropriate learning rate, can always ensure convergence: $\eta^{(k)} \to 0 \;\; as \;\; k \to \infty$
  - for example inverse linear learning rate: $\eta^{(k)} = \frac{\eta^{(1)}}{k}$
  - for inverse linear learning rate convergence in the linearly separable case can also be proven
  - no guarantee that we stopped at a good point, but there are good reasons to choose inverse linear learning rate

LDF: Perceptron Rule and Gradient decent

- Linearly separable data
  - perceptron rule with gradient decent works well
- Linearly non-separable data
  - need to stop perceptron rule algorithm at a good point, this maybe tricky

Batch Rule	*Single Sample Rule*
▪ Smoother gradient because all samples are used	▪ easier to analyze ▪ Concentrates more than necessary on any isolated “noisy” training examples