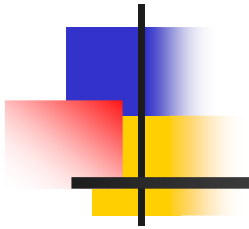
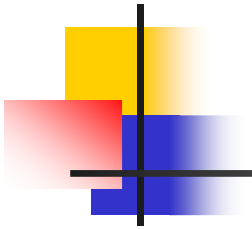
Cmput 466/551



Linear Classifiers

Covering chapters (HTF) : Ch 4

R Greiner
Department of Computing Science
University of Alberta

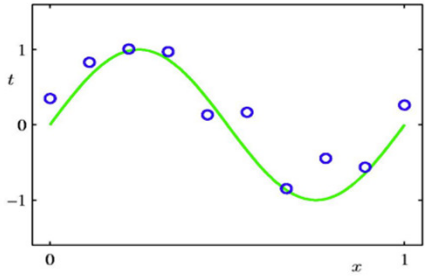


Outline

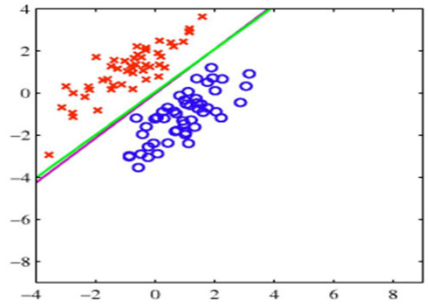
- Framework
- Exact
  - Minimize Mistakes (Perceptron Training)
- LMS
  - Minimize L_2 error
- Logistic Regression
  - Max Likelihood Estimation (MLE) of P(y | **x**)
  - Gradient descent (MSE; MLE)
  - Newton-Raphson
- Linear Discriminant Analysis
  - Max Likelihood Estimation (MLE) of P(y, **x**)
  - Direct Computation
  - Fisher's Linear Discriminant
- Support Vector Machine

Classification vs. Regression

- Same: "Learn a function from labeled examples"
- Different: Domain of label: small set {Y, N, …} vs $\mathcal{R}$
- Regression
  - "Fit the data"



- Classification
  - "Separate the data"

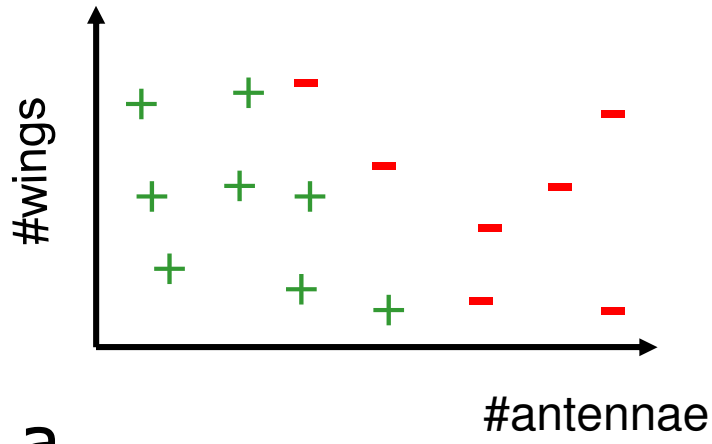


- Why make the distinction?
  - Historically, they have been studied separately
  - The type of label can determine whether an algorithm will work or not … sometimes

Diagnosing Butterfly-itis

Classifier: Decision Boundaries

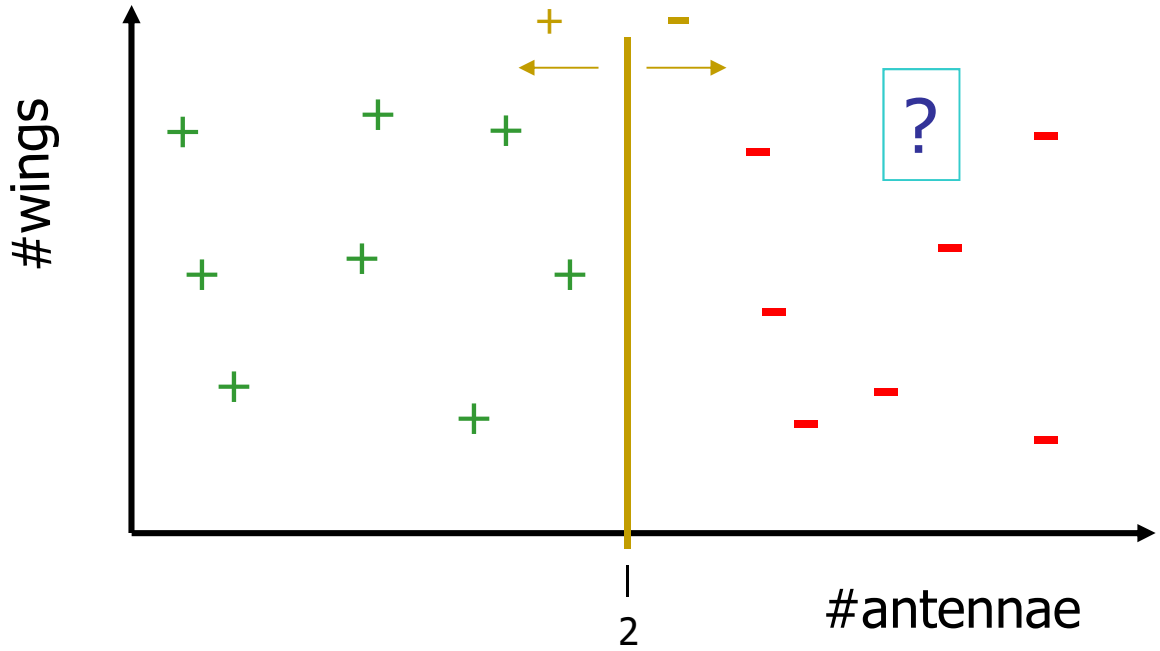
- **Classifier**: partitions input space X into "decision regions"



- *Linear threshold unit* has a linear decision boundary
- Defn: Set of points that can be separated by linear decision boundary is "linearly separable"

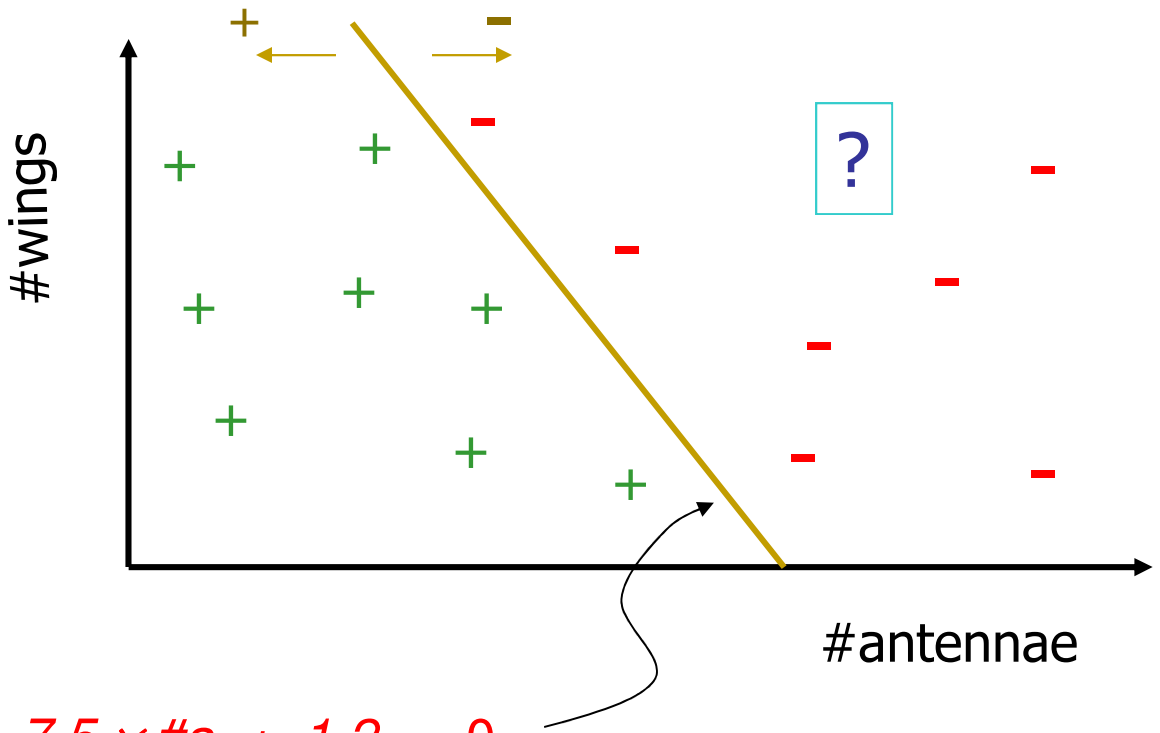
Linear Separators

- Draw "separating line"



- If #antennae ≤ 2, then butterfly-itis
- So ? is Not butterfly-itis.

Can be "angled"…



2.3 × #w − 7.5 × #a + 1.2 = 0

- If *2.3 × #wings − 7.5 × #antennae + 1.2 > 0*
  then butterfly-itis

For Regression: $\boldsymbol{\theta}$
For classification: $\mathbf{w}$

Discriminant Functions

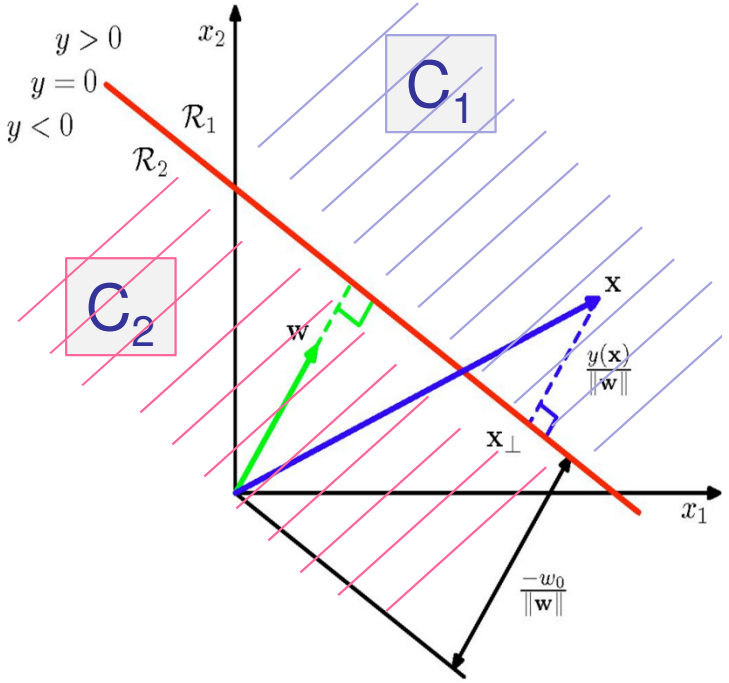
- $y(\mathbf{x}) = y_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$
- Assign $\mathbf{x}$ to C_1 if $y(\mathbf{x}) > 0$

  … otherwise to class C_2 otherwise
- Decision boundary: $y(\mathbf{x}) = 0$
- If $\mathbf{x}_A$ and $\mathbf{x}_B$ lie on decision surface:

  $y(\mathbf{x}_A) = y(\mathbf{x}_B) = 0$

  $\Rightarrow \mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) = 0$

  $\Rightarrow \mathbf{w}$ is orthogonal to the decision surface



- If $\mathbf{x}$ is on decision surface, then: $\frac{\mathbf{w}^T\mathbf{x}}{|\mathbf{w}|} = -\frac{\mathbf{w}_0}{|\mathbf{w}|}$

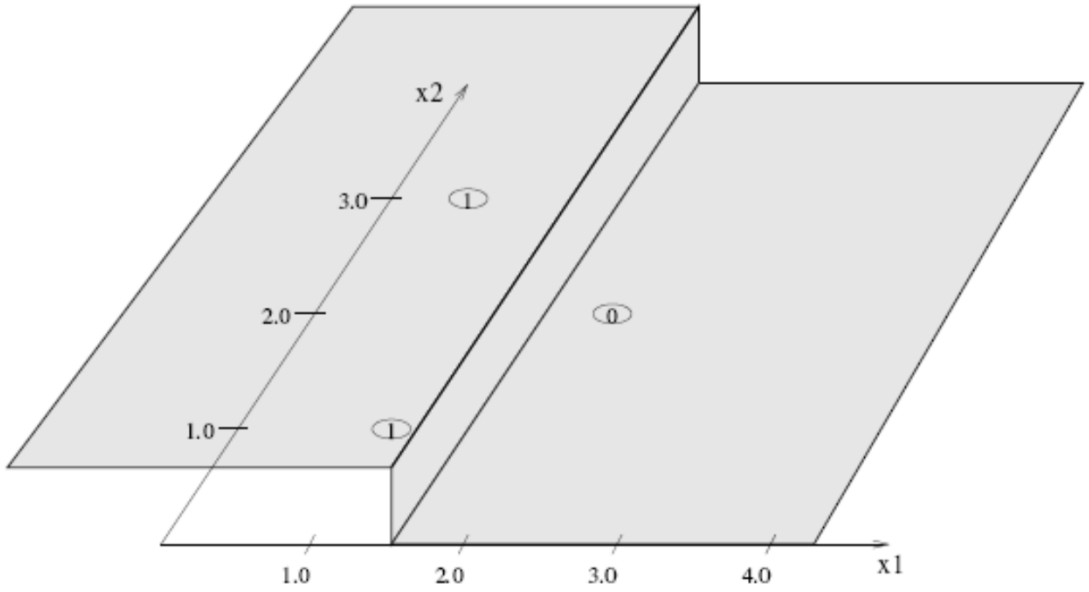
  $\Rightarrow w_0$ determines the location of the decision surface

Geometric View

- Consider 3 training examples:

([1.0, 1.0]; 1)
([0.5, 3.0]; 1)
([2.0, 2.0]; 0)

- Want classifier that looks like...



Linear Separators, in General

- Given data (many features)

F_1	F_2	…	F_n	Class
35	95	…	3	No
22	80	…	-2	Yes
:	:		:	:
10	50	…	1.9	No

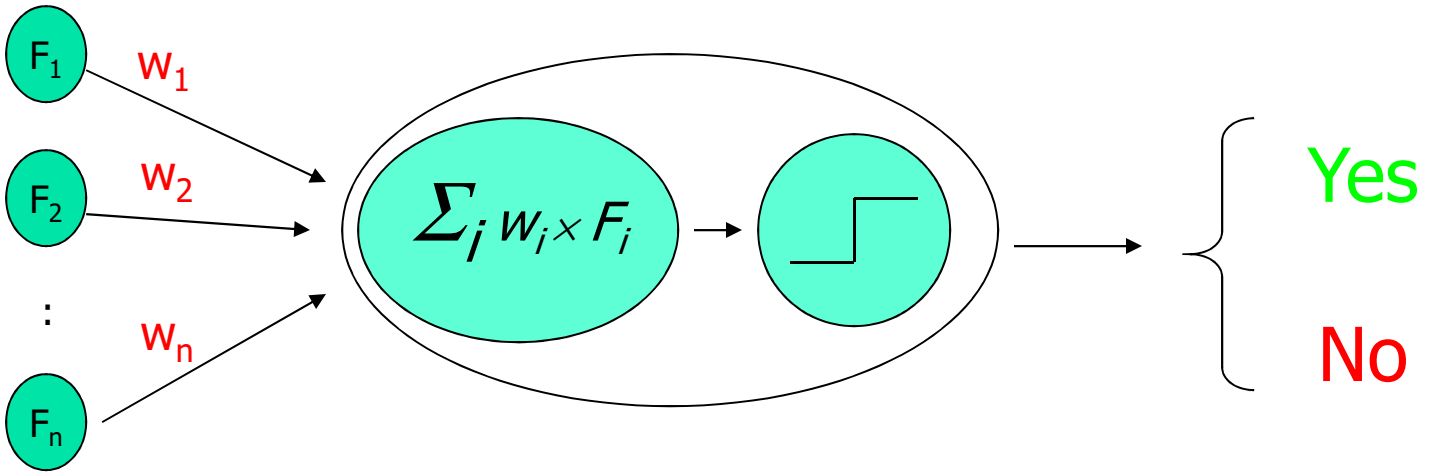
- find "weights" { w_1, w_2, …, w_n, w_0 } such that

$$w_1 \times F_1 + \ldots + w_n \times F_n + w_0 > 0$$

means

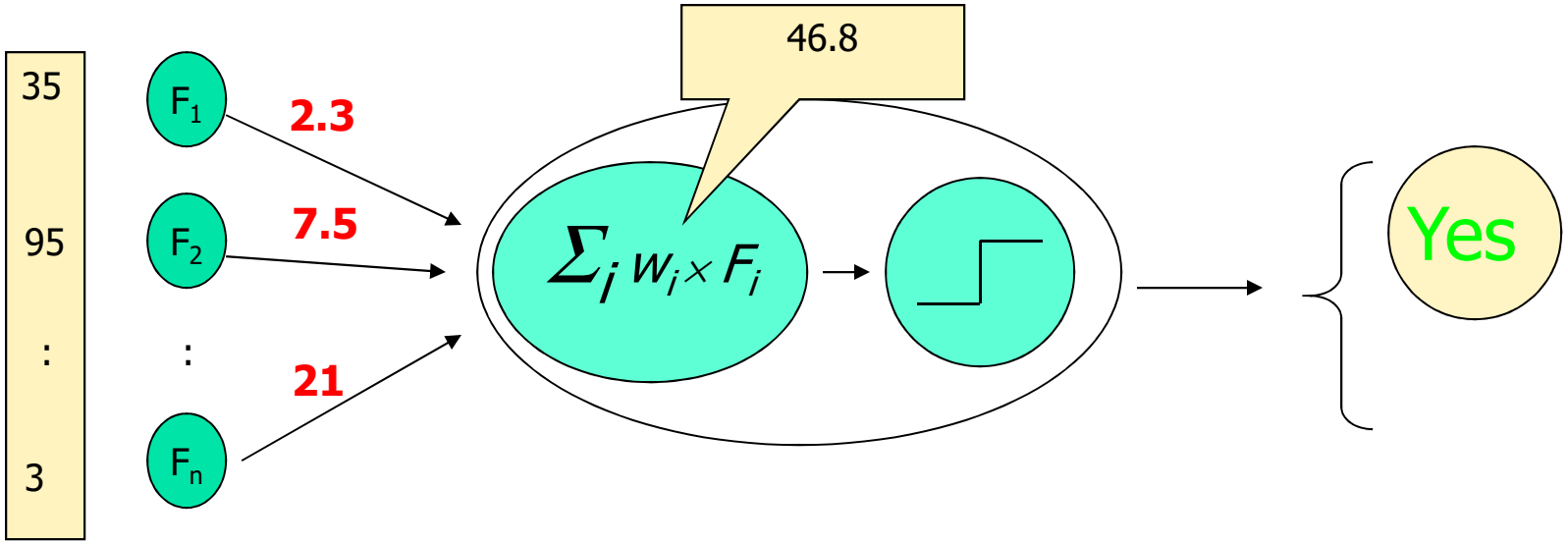
Class = Yes

Linear Separator



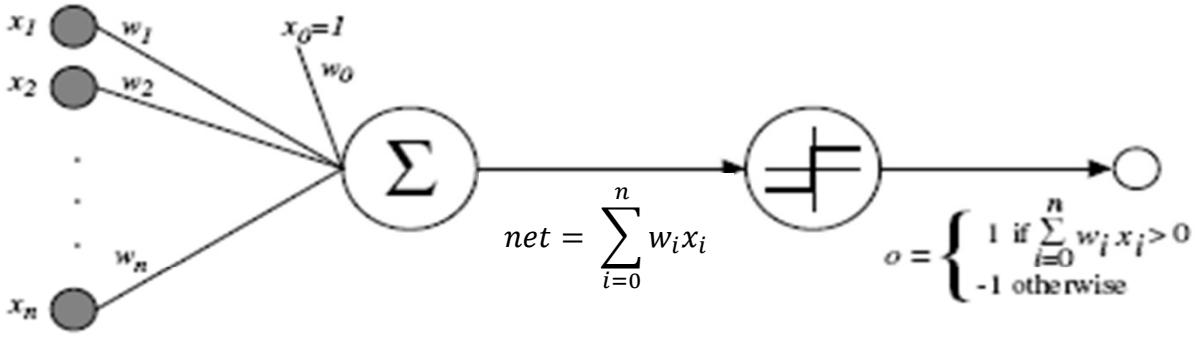
Just view $F_0 = 0$, so w_0 …

Linear Separator



- Performance
  - Given $\{w_i\}$, and values for instance, compute response
- Learning
  - Given labeled data, find "correct" $\{w_i\}$
- Linear Threshold Unit ... "Perceptron"

Linear Threshold Unit: "Perceptron"



$$o_w(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$
$$= \text{sign}(\ (w_0, w_1, \ldots, w_n) \cdot (1, x_1, \ldots, x_n)\ )$$

- Squashing function:
  sign: $\Re \rightarrow \{-1, +1\}$

$$\text{sign}(r) = \begin{cases} 1 & \text{if } r > 0 \\ 0 & \text{otherwise} \end{cases}$$

("Heaviside")

- Actually $\mathbf{w}^T \mathbf{x} > b$ but...
  Create extra input x_0 fixed at 1
  Corresponding w_0 corresponds to -b

Task

- Input: labeled data

x_1	x_2	x_3	$\cdots$	x_r	y
16.1	−22.7	0.3	$\cdots$	−4.0	+
−7.3	−5.1	9.1	$\cdots$	17.1	−
⋮				⋮	⋮
−16.2	−77.0	−1.2	$\cdots$	−4.0	+

Transformed to

x_0	x_1	x_2	x_3	$\cdots$	x_r	y
1	16.1	−22.7	0.3	$\cdots$	−4.0	+
1	−7.3	−5.1	9.1	$\cdots$	17.1	−
⋮	⋮				⋮	⋮
1	−16.2	−77.0	−1.2	$\cdots$	−4.0	+

- Output: $\mathbf{w} \in \Re^{r+1}$

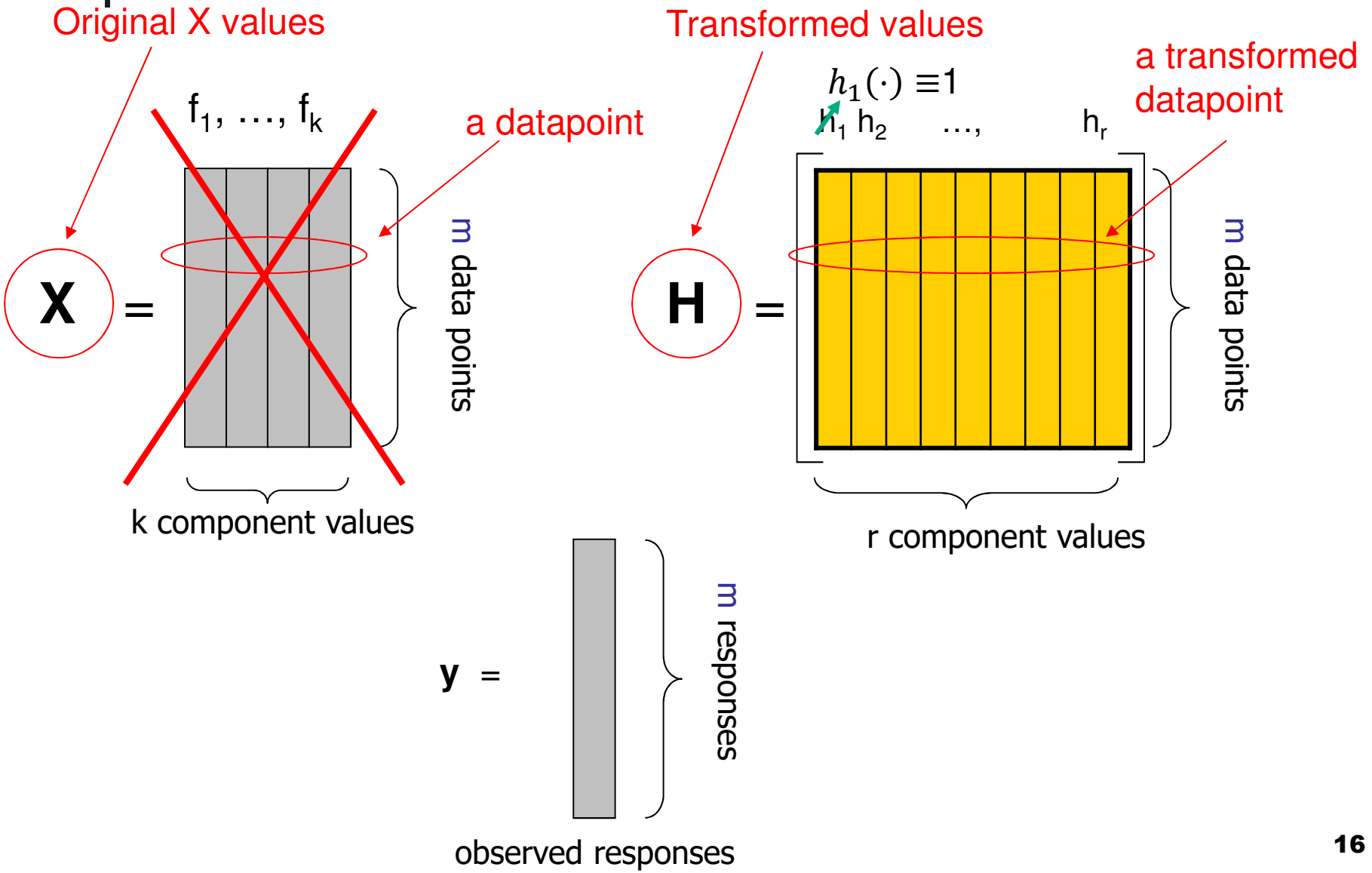
Goal: Want **w** s.t.

$\forall i \;\; \text{sign}(\mathbf{w}^T [1, \mathbf{x}^{(i)}]) = y^{(i)}$

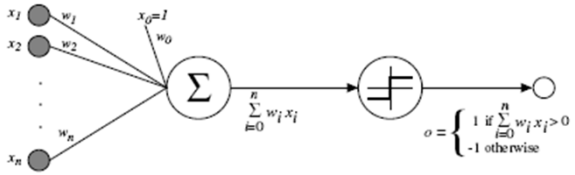
- ... minimize mistakes wrt data ... or …

General Features

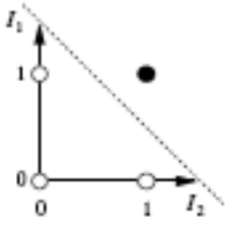
Just like Linear **Regression**



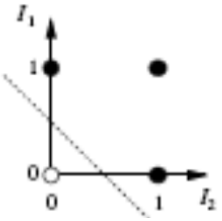
Learning Perceptrons



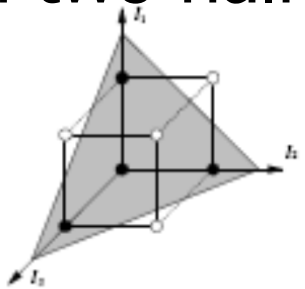
- Can represent Linearly-Separated surface
  ... any hyper-plane between two half-spaces...



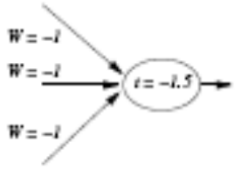
I_1 and I_2



I_1 or I_2



Separating Plane



At least k of n

- Remarkable learning algorithm: [Rosenblatt 1960]

If function *f* can be represented by perceptron,
then ∃ learning alg guaranteed to quickly converge to *f* !

⇒ enormous popularity, early / mid 60's

- But some simple functions cannot be represented
  … killed the field temporarily!

Error Function

Jump

Given data $\{ [x^{(i)}, y^{(i)}] \}_{i=1..m}$, optimize...

- 1. Classification error
  Perceptron Training

$$err_{Class}(w) = \frac{1}{m}\sum_{i=1}^{m} I[\, y^{(i)} \neq sign(w^T x^{(i)})\,]$$

- 2. Mean-squared error (LMS)
  Matrix Inversion; LMS Gradient Descent

$$err_{MSE}(w) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}[\, y^{(i)} - w^T\, x^{(i)}\,]^2$$

- 3. (Log) Conditional Probability (LR)
  MSE Gradient Descent; LCL Gradient Descent

$$LCL(w) = \frac{1}{m}\sum_{i=1}^{m} \log P_w(\, y^{(i)} \mid x^{(i)})$$

- 4. (Log) Joint Probability (LDA; FDA)
  Direct Computation

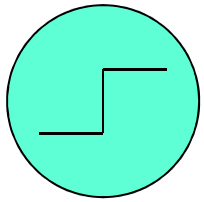
$$LL(w) = \frac{1}{m}\sum_{i=1}^{m} \log P_w(\, y^{(i)}, x^{(i)})$$

- 5. Hinge Loss (SVM)

$$HL(w) = \frac{1}{m}\sum_{i} \lambda_i \left(y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)} - 1\right)_+ + \ldots$$

#1: Optimal Classification Error

- For each labeled instance [**x**, y]
  Err = y – sign($\mathbf{w}^T\mathbf{x}$)
  - y = f(x) is target value
  - sign($\mathbf{w}^T\mathbf{x}$) is perceptron output



- **Idea**: Move weights in appropriate direction, to push Err → 0
- If Err > 0 (error on POSITIVE example)
  - need to increase sign($\mathbf{w}^T\mathbf{x}$)
    ⇒ need to increase $\mathbf{w}^T\mathbf{x}$
  - Input j, x_j, contributes $w_j \cdot x_j$ to $\mathbf{w}^T\mathbf{x}$
    - if $x_j > 0$, *increasing* w_j will increase $\mathbf{w}^T\mathbf{x}$
    - if $x_j < 0$, *decreasing* w_j will increase $\mathbf{w}^T\mathbf{x}$

  ⇒ $w_j \leftarrow w_j + x_j$
- If Err < 0 (error on NEGATIVE example)

  ⇒ $w_j \leftarrow w_j - x_j$

If Err ≠ 0 ⇒ $\mathbf{w} \leftarrow \mathbf{w} + y\,\mathbf{x}$

#1a: Mistake Bound Perceptron Alg

Initialize **w** = 0

Do until no-mistakes:

Predict $\begin{cases} + & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ - & \text{otherwise} \end{cases}$

Mistake on y = +1: **w** ←**w** + **x**

Mistake on y = –1: **w** ←**w** – **x**

Orig Data

$\langle \langle x_1$	$x_2 \rangle$	$c(x) \rangle$
0	0	+
1	0	–
1	1	+

Data + "$x_0 = 1$"

	$\langle \langle x_0$	x_1	$x_2 \rangle$	$c(x) \rangle$
i_1 :	1	0	0	+
i_2 :	1	1	0	–
i_3 :	1	1	1	+

Note: w = 2*i1 + -3*i2 + 2*i3

Weights	Instance	Action
[0 0 0]	i_1	+x
[1 0 0]	i_2	-x
[0 -1 0]	i_3	+x
[1 0 1]	i_1	OK
[1 0 1]	i_2	-x
[0 -1 1]	i_3	+x
[1 0 2]	i_1	OK
[1 0 2]	i_2	-x
[0 -1 2]	i_3	OK
[1 -1 2]	i_1	+x
[1 -1 2]	i_2	OK
[1 -1 2]	i_3	OK
[1 -1 2]	i_1	OK

Mistake Bound Theorem

See SVM…

Theorem: [Rosenblatt 1960]

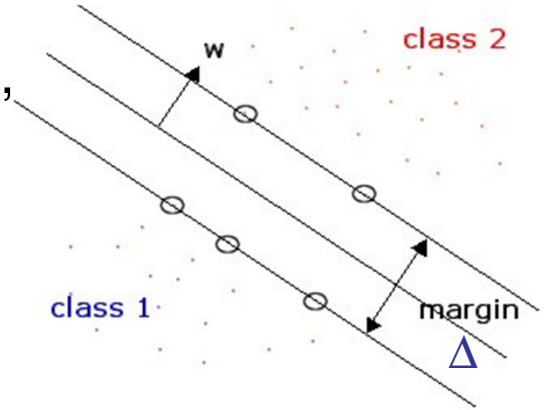
If data is consistent w/some linear threshold **w**, then number of mistakes is $\leq (1/\Delta)^2$,

where $\Delta = \min_{x} \frac{|w^T x|}{|w| \times |x|}$

- Δ measures "wiggle room" available:

  If $|\mathbf{x}| = 1$, then Δ is max, over all consistent planes, of minimum distance of example to that plane
- **w** is $\perp$ to separator, as $\mathbf{w}^T \mathbf{x} = 0$ at boundary
- So $|\mathbf{w}^T \mathbf{x}|$ is projection of **x** onto plane, PERPENDICULAR to boundary line

  … ie, is distance from **x** to that line (once normalized)



Proof of Convergence

For simplicity:
1. Use $x_0 = 1$, so target plane goes thru **0**
2. Assume target plane does not hit any instances
3. Replace each negatively-labeled instance $[[x_0, x_1, \ldots, x_n], -1]$ with positively labeled instance: $[[-x_0, -x_1, \ldots, -x_n], +1]$
4. Normalize all examples to have length 1

- At each time, let **w** be current hypothesis plane
- Example **x** is a "mistake" wrt **w** iff $\mathbf{w}^T \mathbf{x} < 0$
- On each mistake, add **x** to **w**
- Let $\mathbf{w}^*$ be *unit* vector representing target plane

  $\Delta = \min_{\mathbf{x}} \{ \mathbf{w}^{*T} \mathbf{x} \}$

- Consider: $\dfrac{\mathbf{w}^T \mathbf{w}^*}{|\mathbf{w}| \; |\mathbf{w}^*|} = \dfrac{\mathbf{w}^T \mathbf{w}^*}{|\mathbf{w}|}$ as $\mathbf{w}^*$ is *unit* vector

Proof (con't)

$$1 \geq \frac{\mathbf{w}^{\mathrm{T}}\mathbf{w}^{*}}{|\mathbf{w}|\ |\mathbf{w}^{*}|} \geq \frac{m\,\Delta}{\sqrt{m}}$$

- If **x** is mistake ...
  - Numerator increases by $\mathbf{x}^{\mathrm{T}}\mathbf{w}^{*} \geq \Delta$
  - (denominator)^2 becomes
    $(\mathbf{w}+\mathbf{x})^2 = \mathbf{w}^2 + \mathbf{x}^2 + 2(\mathbf{w}^{\mathrm{T}}\mathbf{x}) < \mathbf{w}^2 + 1$
    - as $\mathbf{w}^{\mathrm{T}}\mathbf{x} < 0$, $|\mathbf{x}| = 1$

$$\Delta = \min_{\mathbf{x}} \{\ \mathbf{x}^{\mathrm{T}}\ \mathbf{w}^{*}\ \}$$

- As initially w = (0, …, 0)
  after m mistakes:
  - numerator $\geq m \times \Delta$
  - (denominator)$^2 \leq 0 + \underbrace{1 + \cdots + 1}_{m} = m$

    …so denominator $\leq \sqrt{m}$
- As $\frac{\mathbf{w}^{\mathrm{T}}\mathbf{w}^{*}}{|\mathbf{w}|\ |\mathbf{w}^{*}|}$ = cos(angle between **w** and **w***)

  it must be ≤ 1 …

  $\Rightarrow$ numerator $\leq$ denominator
- $\Rightarrow\ \Delta \times m \leq \sqrt{m} \quad \Rightarrow \quad m \leq \frac{1}{\Delta^2}$

Observations

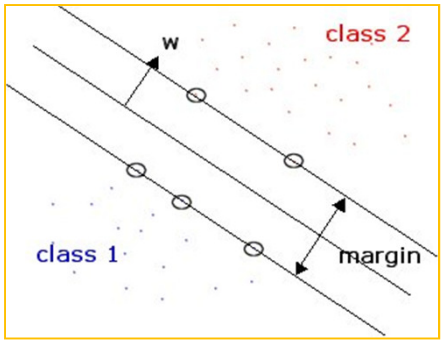
- *Weight vector* is linear sum of instances!

$$\mathbf{w} = \sum_{\{\mathbf{x} \mid y\, \mathbf{x}^{\mathbf{T}}\, w^{(k)} < \mathbf{0}\}} y\, \mathbf{x} = \sum_{i} \alpha_i\, \mathbf{x}^{(\mathbf{i})}$$

  - (Not based on tweaking feature weights { w_i })

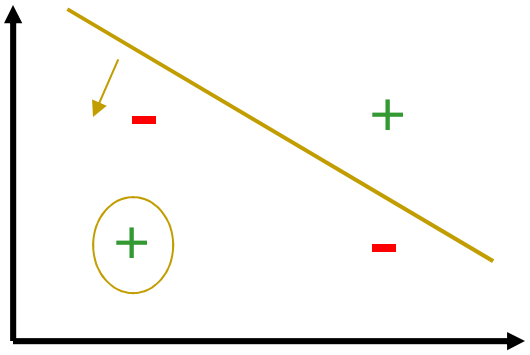
- "Margin" is relevant
  - Task is more difficult for small margins
  - Number of mistakes $\leq \frac{1}{\Delta^2}$

$$\Delta = \min_{\mathbf{x}} \frac{|\mathbf{w}^{\mathrm{T}}\mathbf{x}|}{|\mathbf{w}| \times |\mathbf{x}|}$$



Linear Separators – Facts

- GOOD NEWS:
  - If data is linearly separated,
  - Then FAST ALGORITHM finds correct $\{w_i\}$!
- But…



- Some "data sets" are NOT linearly separatable!

Stay tuned!

If "unrealizable" ?

- Task: Given $\{ [\mathbf{x}^{(i)}, y^{(i)}] \}$ $\quad y^i \in \{ +1, -1 \}$
  Find $\mathbf{w}$ s.t.
  $y^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$ ... for all i

- What if NO $\mathbf{w}$ works?
  ... overconstrained ...
- Could try to minimize residual

$$\sum_i \mathbb{I}[\, y^{(i)} \neq \text{sign}(\, \mathbf{w}^T \mathbf{x}^{(i)} \,) \,]$$

NP-Hard!

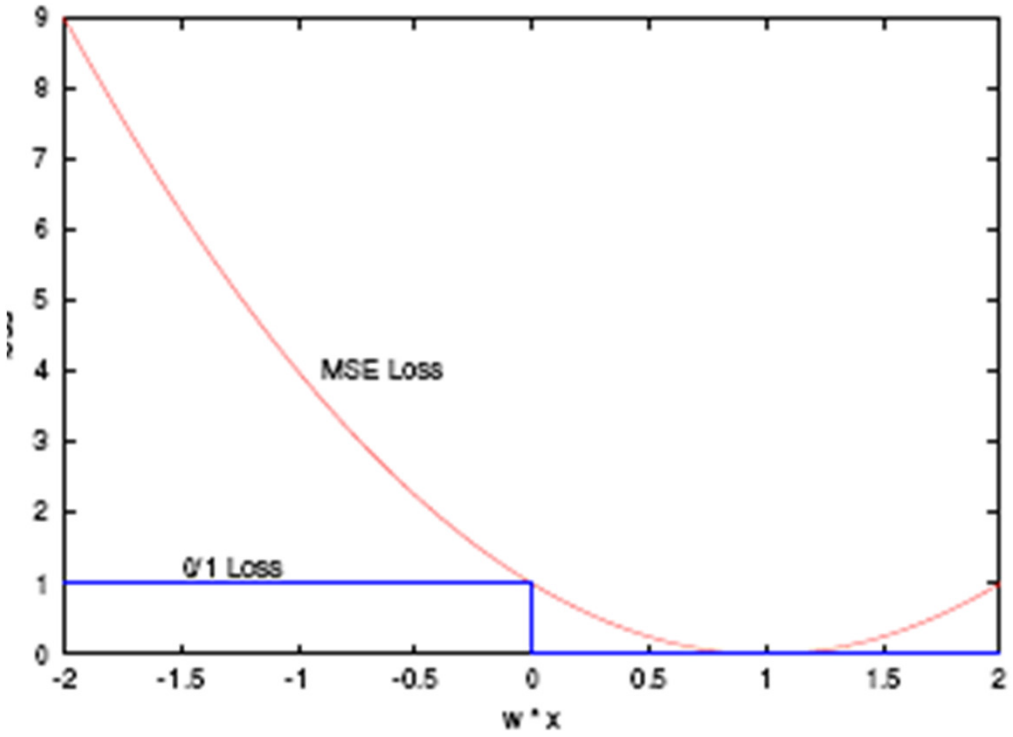
$$|| \, y - \mathbf{X}\,\mathbf{w} \, ||_1 = \sum_i | \, y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)} \, |$$

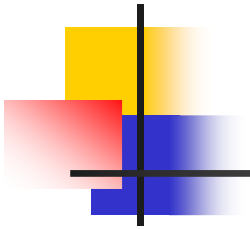
$$|| \, y - \mathbf{X}\,\mathbf{w} \, ||_2^2 = \sum_i (\, y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)} \,)^2$$

Easy!

0/1-Loss vs L_2 error

- "0/1 Loss function"
  - not smooth, not differentiable
- L_2 error (==MSE error) is
  - Smooth
  - Differentiable
  - Overbound...





Outline

- Framework
- Exact
  - Minimize Mistakes (Perceptron Training)
- LMS
  - Minimize L_2 error
- Logistic Regression
  - Max Likelihood Estimation (MLE) of P(y | **x**)
  - Gradient descent (MSE; MLE)
  - Newton-Raphson
- Linear Discriminant Analysis
  - Max Likelihood Estimation (MLE) of P(y, **x**)
  - Direct Computation
  - Fisher's Linear Discriminant
- Support Vector Machine

Error Function

Given data $\{ [x^{(i)}, y^{(i)}] \}_{i=1..m}$, optimize...

- 1. Classification error
  Perceptron Training

$$err_{Class}(w) = \frac{1}{m}\sum_{i=1}^{m} I[\, y^{(i)} \neq sign(w^T x^{(i)})\,]$$

- 2. Mean-squared error (LMS)
  Direct (Gradient Descent)

$$err_{MSE}(w) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}[\, y^{(i)} - w^T\, x^{(i)}\,]^2$$

- 3. (Log) Conditional Probability (LR)
  MSE Gradient Descent; LCL Gradient Descent

$$LCL(w) = \frac{1}{m}\sum_{i=1}^{m} \log P_w(\, y^{(i)} \mid x^{(i)})$$

- 4. (Log) Joint Probability (LDA; FDA)
  Direct Computation

$$LL(w) = \frac{1}{m}\sum_{i=1}^{m} \log P_w(\, y^{(i)}, x^{(i)})$$

- 5. Hinge Loss (SVM)

$$HL(w) = \frac{1}{m}\sum_{i} \lambda_i \left(y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)} - 1\right)_+ + \ldots$$

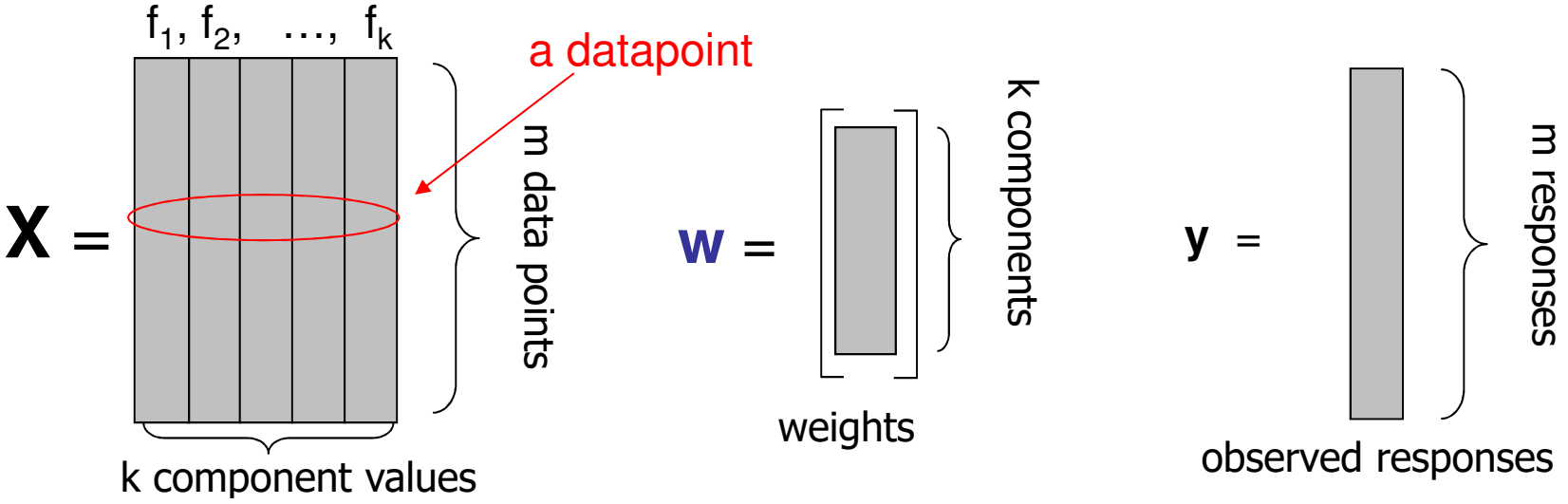
#2: LMS Approach

- Given $\mathbf{X} \in \Re^{k \times m}$, $\mathbf{y} \in \{-1, +1\}^m$,
- Find **w** to minimize

$$|| \mathbf{y} - \mathbf{X}\,\mathbf{w} ||_2^2 = \sum_i (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2$$

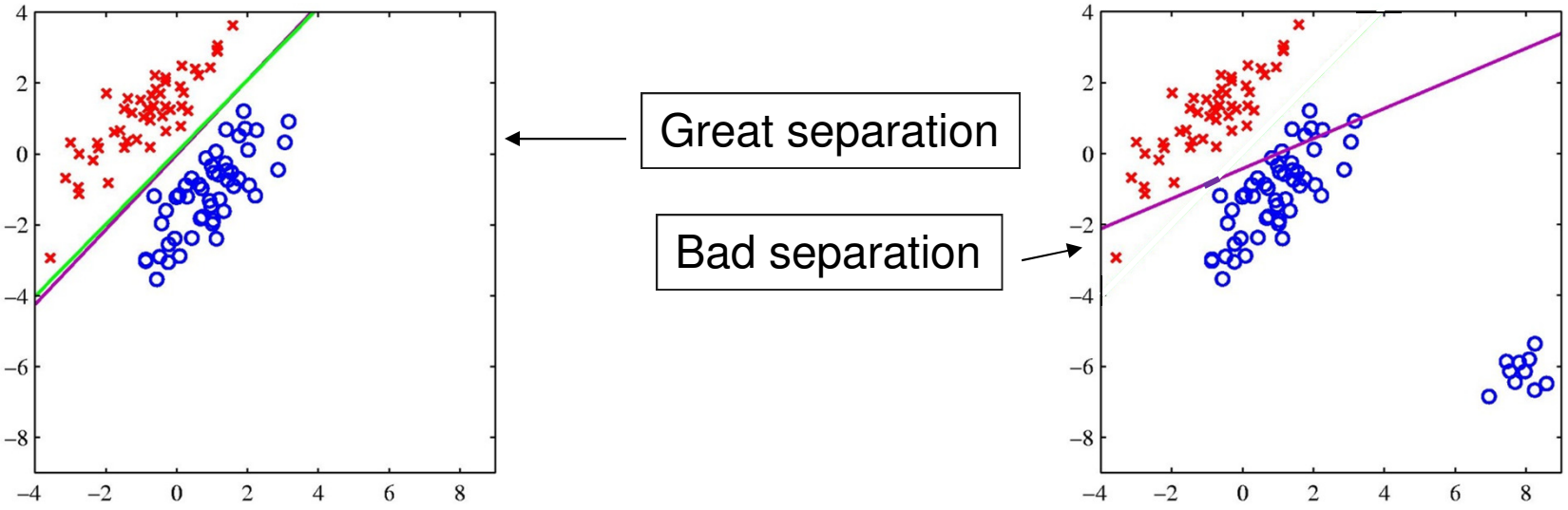
- Solution: $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

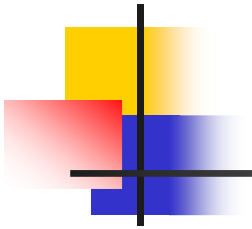
… or gradient descent or …



Use Linear Regression for Classification?

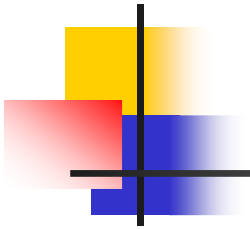
- Regression minimizes
  sum of squared errors on target function
  $\sum_i (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})^2$
- which gives strong influence to distant points …
  - even if "correct" $y^{(i)} = +1$, $\mathbf{w}^T \mathbf{x}^{(i)} = 21$ … error of 20!





for next year

- have time, with w^T x values and y values…
- and note that when y=1,
  error for w^T x of 1.3 is fine, as is 0.6…
  and even for -2, or arguably, even for -20.
- But when it is +21, note still bit error – error of 20!!
  - even though it was correct.



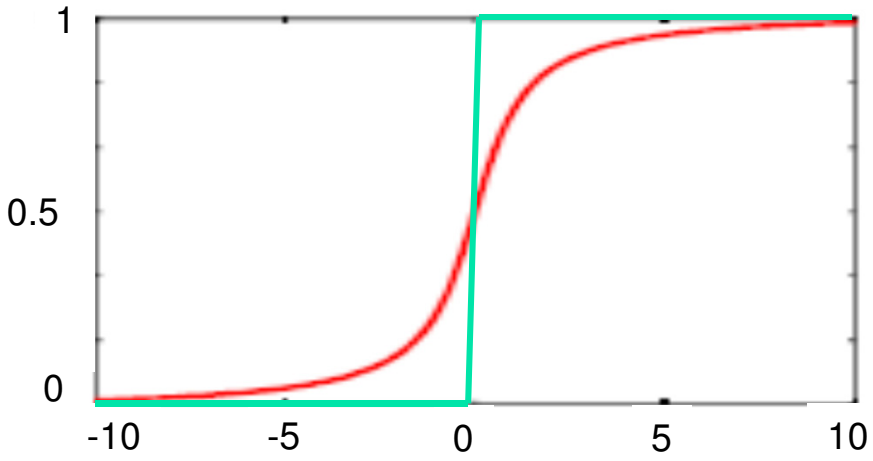
Outline

- Framework
- Exact
  - Minimize Mistakes (Perceptron Training)
- LMS
  - Minimize L_2 error
- Logistic Regression
  - Max Likelihood Estimation (MLE) of P(y | **x**)
  - Gradient descent (MSE; MLE)
  - Newton-Raphson
- Linear Discriminant Analysis
  - Max Likelihood Estimation (MLE) of P(y, **x**)
  - Direct Computation
  - Fisher's Linear Discriminant
- Support Vector Machine

#3: Logistic Regression

- Want to predict $y \in \{0,1\}$, based on linear combination of $\mathbf{x}$
  - But $\mathbf{w}^T\mathbf{x}$ has range $[-\infty, \infty]$
    … so even if "correct", still BIG error
- Approach: transform $\mathbf{w}^T\mathbf{x}$ to be in range $\in [0,1]$
- Need "squashing" function $[-\infty, \infty] \rightarrow [0, 1]$

$$\sigma(a) = \frac{1}{1+e^{-a}}$$



  - Return 1 if $\sigma(\mathbf{w}^T\mathbf{x}) \geq ½$
- Also probabilistic interpretation: $\sigma(\mathbf{w}^T\mathbf{x}) = P_{\mathbf{w}}(y=1|\ \mathbf{x})$

Error Function

Given data $\{ [x^{(i)}, y^{(i)}] \}_{i=1..m}$, optimize...

- 1. Classification error
  Perceptron Training

$$err_{Class}(w) = \frac{1}{m}\sum_{i=1}^{m} I[\, y^{(i)} \neq sign(w^T x^{(i)})\,]$$

- 2. Mean-squared error (LMS)
  Direct (Gradient Descent)

$$err_{MSE}(w) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}[\, y^{(i)} - w^T\, x^{(i)}\,]^2$$

- 3. (Log) Conditional Probability (LR)
  MSE Gradient Descent; LCL Gradient Descent

$$LCL(w) = \frac{1}{m}\sum_{i=1}^{m} \log P_w(\, y^{(i)} \mid x^{(i)})$$

- 4. (Log) Joint Probability (LDA; FDA)
  Direct Computation

$$LL(w) = \frac{1}{m}\sum_{i=1}^{m} \log P_w(\, y^{(i)}, x^{(i)})$$

- 5. Hinge Loss (SVM)

$$HL(w) = \frac{1}{m}\sum_{i} \lambda_i \left(y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)} - 1\right)_+ + \ldots$$

Probabilistic Interpretation…

Use “+y” for y=1
“-y” for y=0

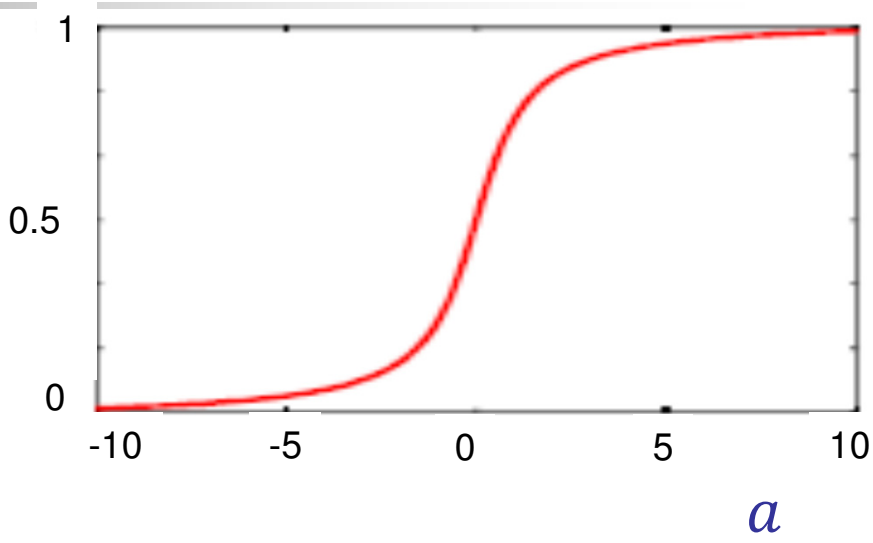
$$P(+y \mid x) = \frac{P(+y,\ x)}{P(+y,\ x) + P(-y,\ x)}$$

$$= \frac{1}{1 + \exp(-a)}$$

$$a = \ln \frac{P(+y,\ x)}{P(-y,\ x)} = \ln \frac{P(+y \mid x)}{P(-y \mid x)}$$

Logistic Function

- $\sigma(a) = \frac{1}{1+e^{-a}}$
- aka Sigmoid
- Useful properties
  - $\sigma: \Re \rightarrow [0,1]$
  - For very large positive a: $\sigma(a) \approx 1$
  - For very large negative a: $\sigma(a) \approx 0$
  - If $a \approx 0$, $\sigma(a) \approx a + \frac{1}{2}$
  - $\frac{\partial \sigma(a)}{\partial a} = \sigma(a)\,[1 - \sigma(a)]$



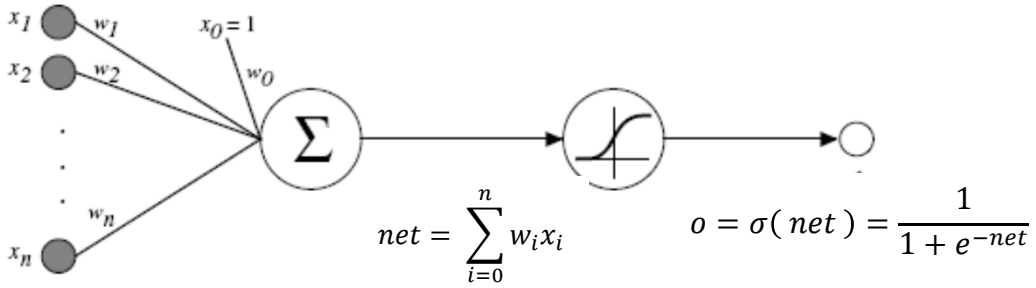
Derivative of Sigmoid

$$\frac{d}{da}\sigma(a) = \frac{d}{da}\frac{1}{(1+e^{-a})}$$

$$= \frac{-1}{(1+e^{-a})^2}\frac{d}{da}(1+e^{-a}) = \frac{-1}{(1+e^{-a})^2}(-e^{-a})$$

$$= \frac{e^{-a}}{(1+e^{-a})^2} = \frac{1}{(1+e^{-a})}\frac{e^{-a}}{(1+e^{-a})} = \sigma(a)\,[1-\sigma(a)]$$

Logistic Regression (con't)



- Assume 2 classes:

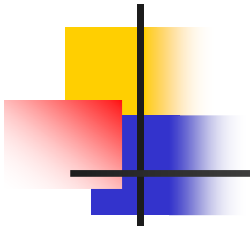
$$P_w(+y \mid x) = \sigma(w \cdot x) = \frac{1}{1+e^{-(w \cdot x)}}$$

$$P_w(-y \mid x) = 1 - \frac{1}{1+e^{-(w \cdot x)}} = \frac{e^{-(w \cdot x)}}{1+e^{-(w \cdot x)}}$$

- Log Odds:

$$\log \frac{\mathrm{P_w}(+\mathrm{y} \mid \mathbf{x})}{\mathrm{P_w}(-\mathrm{y} \mid \mathbf{x})} = \mathbf{w}^{\mathrm{T}} \mathbf{x}$$

Linear



How to learn parameters **w** ?

- … depends on goal?
  - A: Minimize mean square error (MSE)?

    $\sum_i (y^{(i)} - \sigma(\mathbf{w}^T \mathbf{x}^{(i)}))^2$
  - B: Maximize (log) conditional likelihood (LCL)?

    $\sum_i \log P_{\mathbf{w}}(y^{(i)} \mid \mathbf{x}^{(i)})$

MSError Gradient for Sigmoid Unit

- Error: $E = \sum_i \frac{1}{2} (o^{(i)} - y^{(i)})^2 = \sum_i E^{(i)}$

For single training instance

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

- **Input**: $\mathbf{x}^{(i)} = [1, x^{(i)}_1, \ldots, x^{(i)}_k]$
- **Computed Output**: $o^{(i)} = \sigma(z^{(i)})$
  - where $z^{(i)} = \sum_j x_j^{(i)} w_j$ using current $[w_0, w_1, \ldots, w_k]$
- **Correct output**: $y^{(i)}$
- Error Gradient (Ignore $^{(i)}$ superscript)

$$\frac{\partial E^{(i)}}{\partial w_j} = \frac{\partial}{\partial w_j}\left[\frac{1}{2}(o-y)^2\right] = \frac{1}{2}\left[2\,(o-y)\frac{\partial}{\partial w_j}(o-y)\right]$$

$$= (o-y)\frac{\partial o}{\partial w_j} = (o-y)\frac{\partial \sigma(z)}{\partial z}\frac{\partial z}{\partial w_j}$$

As $o = \sigma(z)$

Updating LR Weights (MSE)

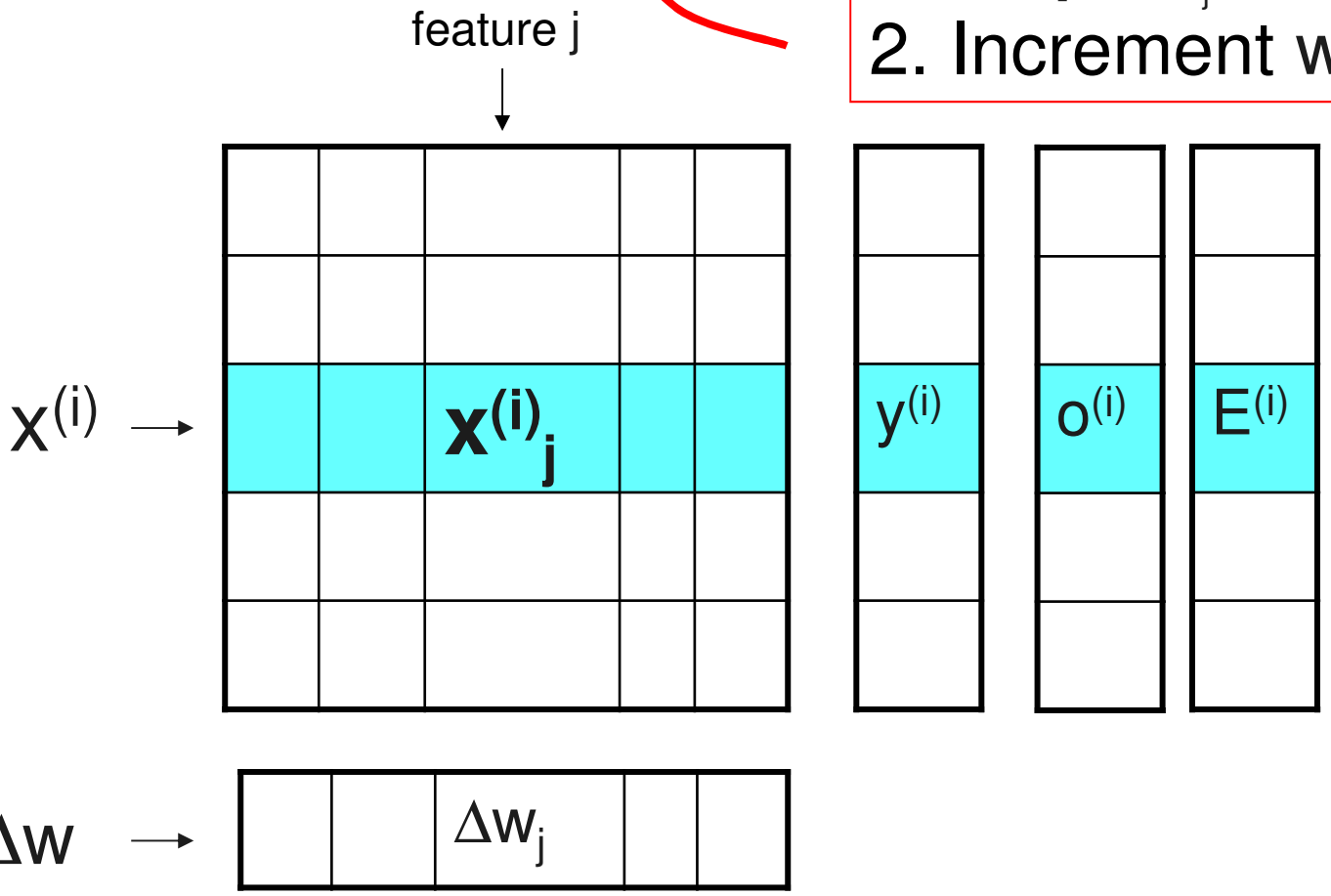
- $\frac{\partial E^{(i)}}{\partial w_j} = (o - y)\frac{\partial \sigma(z)}{\partial z}\frac{\partial z}{\partial w_j}$
- Using:
  - $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)\,[1 - \sigma(z)] = o(1-o)$
  - $\frac{\partial z}{\partial w_j} = \frac{\partial -\sum_k w_k x_k}{\partial w_j} = -x_j$
- $\frac{\partial E^{(i)}}{\partial w_j} = \left(o^{(i)} - y^{(i)}\right) o^{(i)}\left(1 - o^{(i)}\right)\left(-x_j^{(i)}\right)$
- As already computed $o^{(i)} = \sigma(z^{(i)})$ to get answer, trivial to compute $\sigma'(z^{(i)}) = \sigma(z^{(i)})\,(1 - \sigma(z^{(i)}))$
- Update $w_j \mathrel{+}= \Delta w_j$ where

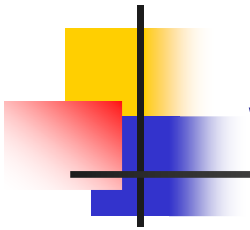
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\Delta w_j = -\eta \frac{\partial E^{(i)}}{\partial w_j}$$

#3a: MSE for LR

0. New **w**
   $\Delta\mathbf{w} := 0$
1. For each row i, compute
   a. $E^{(i)} = (o^{(i)} - y^{(i)})\, o^{(i)} (1 - o^{(i)})$
   b. $\Delta\mathbf{w} \mathrel{+}= E^{(i)}\, \mathbf{x}^{(i)}$
      [… $\Delta w_j \mathrel{+}= E^{(i)}\, x^{(i)}_j$ …]
2. Increment $w \mathrel{+}= \eta\Delta\mathbf{w}$





#3b: ML Estimation wrt Learn Conditional Probability

- $P(S \mid \mathbf{w}) \equiv$ likelihood function

  $L(\mathbf{w}) = \log P(S \mid \mathbf{w})$

- $\mathbf{w}^* = \text{argmax}_{\mathbf{w}} L(\mathbf{w})$

  is "maximum likelihood estimator"
  for Log Conditional Likelihood (LCL)

- sometimes called "MLE" …

Computing the Likelihood

- As training examples $S = \{ [\mathbf{x}^{(i)}, y^{(i)}] \}$ are iid
  - drawn independently from same (unknown) prob $P_{\mathbf{w}}(\mathbf{x}, y)$
- $\log P(S \mid \mathbf{w}) = \log \prod_i P_{\mathbf{w}}(\mathbf{x}^{(i)}, y^{(i)})$

  $= \sum_i \log P_{\mathbf{w}}(\mathbf{x}^{(i)}, y^{(i)})$

  $= \sum_i \log P_{\mathbf{w}}(y^{(i)} \mid \mathbf{x}^{(i)}) + \sum_i \log P_{\mathbf{w}}(\mathbf{x}^{(i)})$
  - As $P_{\mathbf{w}}(\mathbf{x}^{(i)})$ does not depend on $\mathbf{w}$, ...

- $\mathbf{w}^* = \text{argmax}_{\mathbf{w}} \sum_i \log P_{\mathbf{w}}(y^{(i)} \mid \mathbf{x}^{(i)})$

Optimize LCL (LR) by Gradient Ascent

- Want $\mathbf{w}^* = \text{argmax}_{\mathbf{w}} L(\mathbf{w})$
  - $L(\mathbf{w}) = \sum_i r(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{w})$ over instances [$y^{(i)}, \mathbf{x}^{(i)}$]
  - As $y \in \{0, 1\}$

$r(y, \mathbf{x}, \mathbf{w}) = \log P_{\mathbf{w}}(y \mid \mathbf{x}) =$

$y \log(P_{\mathbf{w}}(y=1 \mid \mathbf{x})) + (1 - y) \log(1 - P_{\mathbf{w}}(y=1 \mid \mathbf{x}))$

- So climb along…

$$\frac{\partial L(\mathbf{w})}{\partial w_j} = \sum_i \frac{\partial r\left(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{w}\right)}{\partial w_j}$$

Gradient Descent …

$$p_1 = P_{\mathbf{w}}(y=1 \mid \mathbf{x}) = \sigma\left(w^T x^{(i)}\right) = \frac{1}{1+\exp(-\mathbf{w}^T \mathbf{x}^{(i)})}$$

$$\frac{\partial\, r(y, \mathbf{x}, \mathbf{w})}{\partial\, w_j} = \frac{\partial}{\partial\, w_j}[\, y \log(\, p_1) + (1-\, y) \log(1-\, p_1)$$

$$= \frac{y}{p_1}\frac{\partial p_1}{\partial\, w_j} + (-1)\times\frac{1-y}{1-p_1}\frac{\partial p_1}{\partial\, w_j} = \frac{y-p_1}{p_1(1-p_1)}\frac{\partial p_1}{\partial\, w_j}$$

$$\frac{\partial p_1}{\partial\, w_j} = \frac{\partial\, P_w(y=1 \mid x)}{\partial\, w_j} = \frac{\partial}{\partial\, w_j}\left(\sigma(x \cdot w)\right)$$

$$= \sigma(x \cdot w)[1-\sigma(x \cdot w)]\frac{\partial}{\partial\, w_j}\left(-\, x \cdot w\right) = p_1(1-\, p_1)\cdot(-x_j^{(i)})$$

$$\frac{\partial\, L(w)}{\partial\, w_j} = \sum_i \frac{\partial\, r(y^{(i)}, x^{(i)}, w)}{\partial\, w_j} = \sum_i \frac{y^{(i)}-\, p_1}{p_1(1-\, p_1)} p_1(1-\, p_1)\cdot(-x_j^{(i)})$$

$$= \sum_i (\sigma(w^T x^{(i)}) - y^{(i)})\cdot x_j^{(i)}$$

Gradient Ascent for Logistic Regression (LCL)

Given: training examples $[\mathbf{x}^{(i)}, y^{(i)}]$, i = 1..N

Initialize weight vector $\mathbf{w} = [0, 0, \ldots, 0]$

Repeat until convergence

  Let gradient vector $\Delta\mathbf{w} = [0, 0, \ldots, 0]$

  For i = 1 to N **do**

    $o^{(i)} = \sigma(\mathbf{w}^{T}\mathbf{x}^{(i)})$ % p($y^{(i)}$ =1 | $x^{(i)}$)

    $err_i = o^{(i)} - y^{(i)}$

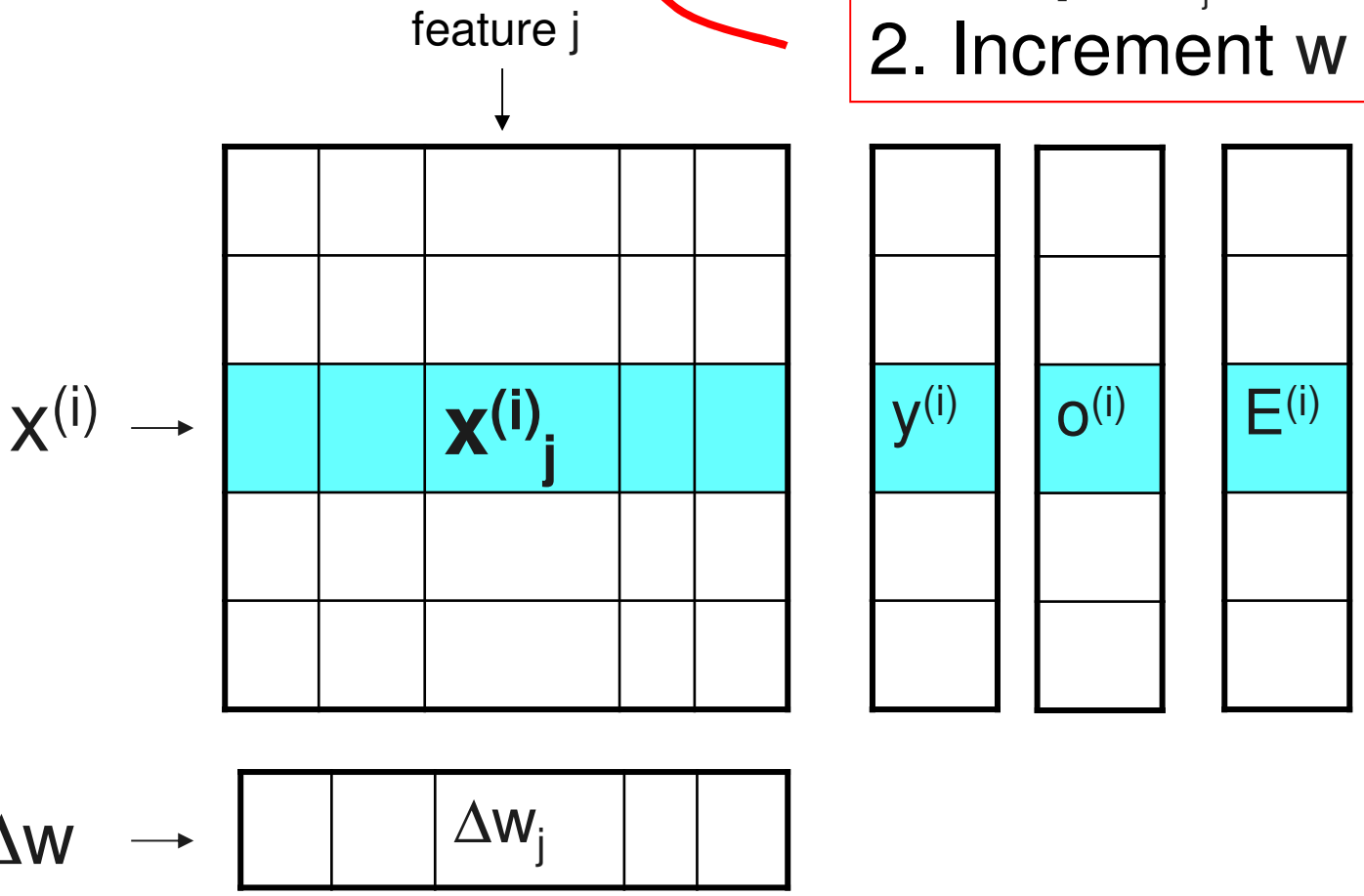
    For j = 1 to n **do**

      $\Delta\mathbf{w}_j \mathrel{+}= \mathbf{err}_i \times \mathbf{x}_j^{(i)}$

  $\mathbf{w} \mathrel{+}= \eta_t \Delta\mathbf{w}$; step in direction of increasing gradient

#3b: LCL for LR

0. New **w**
   $\Delta\mathbf{w} := 0$
1. For each row i, compute
   a. $E^{(i)} = (o^{(i)} - y^{(i)})$
   b. $\Delta\mathbf{w} \mathrel{+}= E^{(i)}\,\mathbf{x}^{(i)}$
      [… $\Delta w_j \mathrel{+}= E^{(i)}\, x^{(i)}_j$ …]
2. Increment $w \mathrel{+}= \eta\Delta\mathbf{w}$



Learning LR Weights

$$o^{(i)} = \sigma(\boldsymbol{w}^T \boldsymbol{x}^{(i)}) = \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(\mathrm{i})})}$$

- Task: Given data $\{[\boldsymbol{x}^{(i)}, y^{(i)}]\}$,
  - find **w** in $P_w(y|\boldsymbol{x}) = \begin{cases} \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x})} & \text{if } y = 1 \\ \frac{\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x})}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x})} & \text{if } y = 0 \end{cases}$
  - s.t. $\mathrm{P}_{\mathbf{w}}(\mathrm{y}^{(\mathrm{i})}|\mathbf{x}^{(\mathrm{i})}) > \frac{1}{2}$ iff $\mathrm{y}^{(\mathrm{i})} = 1$
- Approach 1: MSE ≈ Neural Nets
  - Minimize $\sum_{\mathrm{i}}(\mathrm{o}^{(\mathrm{i})} - \mathrm{y}^{(\mathrm{i})})^2$
  - Gradient: $\Delta \mathrm{w}_{\mathrm{j}}^{(\mathrm{i})} = (\mathrm{o}^{(\mathrm{i})} - \mathrm{y}^{(\mathrm{i})})\, \mathrm{o}^{(\mathrm{i})} (1 - \mathrm{o}^{(\mathrm{i})})\, \mathrm{x}_{\mathrm{j}}^{(\mathrm{i})}$
- Approach 2: LCL ≈ Probabilistic Logistic Regression
  - Maximize $\sum_{\mathrm{i}} \mathrm{P}_{\mathbf{w}}(\mathrm{y}^{(\mathrm{i})}|\mathbf{x}^{(\mathrm{i})})$
  - Gradient: $\Delta \mathrm{w}_{\mathrm{j}}^{(\mathrm{i})} = (\mathrm{o}^{(\mathrm{i})} - \mathrm{y}^{(\mathrm{i})})\, \mathrm{x}_{\mathrm{j}}^{(\mathrm{i})}$

Use Logistic Regression for Classification

- Return $y = 1$ iff

$$P(y=1|x) > P(y=0|x)$$

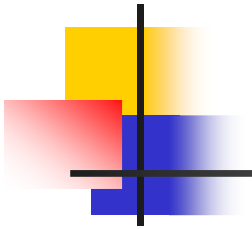
$$\frac{P(y=1|x)}{P(y=0|x)} > 1$$

$$\ln\frac{P(y=1|x)}{P(y=0|x)} > 0$$

$$\ln\frac{1/(1+\exp(-w\cdot x))}{\exp(-w\cdot x)/(1+\exp(-w\cdot x))} > 0$$

$$\ln\frac{1}{\exp(-w\cdot x)} = w\cdot x > 0$$

Logistic Regression learns a LTU!



Comments on LCL Algorithm

- This is BATCH;
  ∃ obvious online alg
  (stochastic gradient ascent)
- Can use second-order (Newton-Raphson)
  alg for faster convergence
  - weighted least squares computation;
    aka
    "Iteratively-Reweighted Least Squares" (IRLS)

Logistic Regression Computation…

$$L(\mathbf{w}) = \sum_i \log P_{\mathbf{w}}\left(y^{(i)} \mid \boldsymbol{x}^{(i)}\right)$$

$$\frac{\partial \mathrm{L}(\mathbf{w})}{\partial \mathrm{w_j}} = \sum_{\mathrm{i}=1}^{\mathrm{N}} \left(\mathrm{y}^{(\mathrm{i})} - \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(\mathbf{i})})} \right) \mathrm{x}_{\mathrm{j}}^{(\mathrm{i})} = 0$$

- $(p+1)$ non-linear equations
- Solve by Newton-Raphson method:

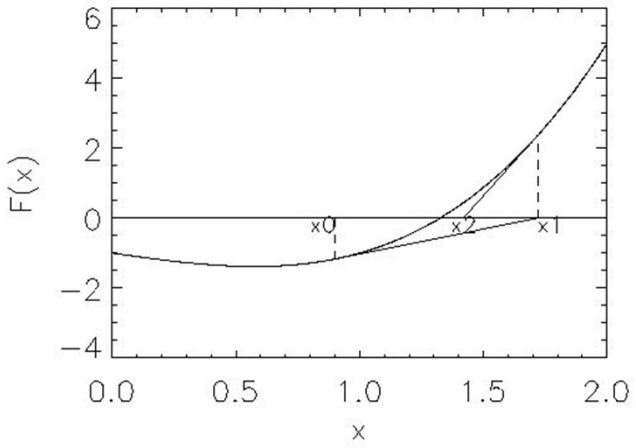
$$\mathbf{w}^{\mathrm{new}} = \mathbf{w}^{\mathrm{old}} - \left[\mathrm{Jacobian} \left(\frac{\partial L\left(\boldsymbol{w}^{old}\right)}{\partial \boldsymbol{w}} \right) \right]^{-1} \frac{\partial L\left(\boldsymbol{w}^{old}\right)}{\partial \boldsymbol{w}}$$

Newton-Raphson Method

- A gen'l technique for solving $f(x) = 0$
  - … even if non-linear
- Taylor series:
  - $f(x_{n+1}) \approx f(x_n) + (x_{n+1} - x_n) f'(x_n)$
  - $x_{n+1} \approx x_n + [f(x_{n+1}) - f(x_n)] / f'(x_n)$
- When x_{n+1} near root, $f(x_{n+1}) \approx 0$

$$\Rightarrow \quad x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$$

Update each iteration…



Newton-Raphson in Multi-dimensions

- To solve the equations:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_N) &= 0 \\ f_2(x_1, x_2, \ldots, x_N) &= 0 \\ &\vdots \\ f_N(x_1, x_2, \ldots, x_N) &= 0 \end{aligned}$$

- Taylor series: $\forall\, j = 1..N$

$$f_j(x + \Delta x) \approx f_j(x) + \sum_{k=1}^{N} \frac{\partial f_j}{\partial x_k} \Delta x_k$$

- N-R:

$$\begin{bmatrix} x_1^{n+1} \\ x_2^{n+1} \\ \vdots \\ x_N^{n+1} \end{bmatrix} = \begin{bmatrix} x_1^{n} \\ x_2^{n} \\ \vdots \\ x_N^{n} \end{bmatrix} - \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix}^{-1} \begin{bmatrix} f_1(x_1^n, x_2^n, \ldots, x_N^n) \\ f_2(x_1^n, x_2^n, \ldots, x_N^n) \\ \vdots \\ f_N(x_1^n, x_2^n, \ldots, x_N^n) \end{bmatrix}$$

Jacobian matrix

Newton-Raphson: Example

- To solve

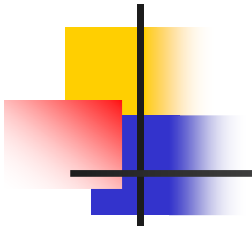
$$f_1(x_1, x_2) = x_1^2 - \cos(x_2) = 0$$

$$f_2(x_1, x_2) = \sin(x_1) + x_1^2 + x_2^3 = 0$$

- Iterate using

$$\begin{bmatrix} x_1^{n+1} \\ x_2^{n+1} \end{bmatrix} = \begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix} - \begin{bmatrix} 2x_1^n & \sin(x_2^n) \\ \cos(x_1^n) + 2x_1^n & 3(x_2^n)^2 \end{bmatrix}^{-1} \begin{bmatrix} (x_1^n)^2 - \cos(x_2^n) \\ \sin(x_1^n) + (x_1^n)^2 + (x_2^n)^3 \end{bmatrix}$$

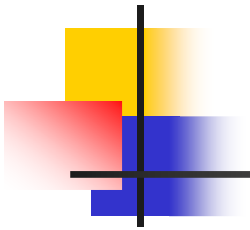
Logistic Regression Algs for LTUs



- Learns Conditional Probability Distribution P(y | **x**)

- **Local Search**:
  Begin with initial weight vector;
  iteratively modify to maximize objective function
  log likelihood of the data
  (ie, find **w** s.t.
  probability distribution $P_{\mathbf{w}}$(y | **x**) is most likely,
  given training data)

- **Online** or **batch**



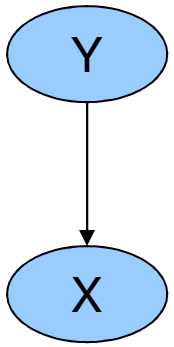
Outline

- Framework
- Exact
  - Minimize Mistakes (Perceptron Training)
- LMS
  - Minimize L_2 error
- Logistic Regression
  - Max Likelihood Estimation (MLE) of P(y | **x**)
  - Gradient descent (MSE; LCL)
  - Newton-Raphson
- Linear Discriminant Analysis
  - Max Likelihood Estimation (MLE) of P(y, **x**)
  - Direct Computation
  - Fisher's Linear Discriminant
- Support Vector Machine

#4: Linear Discriminant Analysis

- LDA learns joint distribution $P_w(y, \mathbf{x})$
  - As $P_w(y, x) \neq P_w(y \mid x)$;
    $\text{argmax}_{\mathbf{w}} P_{\mathbf{w}}(y, x) \neq \text{argmax}_{\mathbf{w}} P_{\mathbf{w}}(y \mid x)$
- "generative model"
  - $P(y, \mathbf{x})$ model of how data is generated
  - Eg, factor

    $P(y, \mathbf{x}) = P(y) P(\mathbf{x} \mid y)$
    - $P(y)$ generates value for y; then
    - $P(\mathbf{x} \mid y)$ generates value for $\mathbf{x}$ given this y
- Belief net:



Linear Discriminant Analysis, con't

- $P(y, \mathbf{x}) = P(y) P(\mathbf{x} \mid y)$
- $P(y)$ is a simple discrete distribution
  - Eg: $P(y = 0) = 0.31$; $P(y = 1) = 0.69$

    (31% negative examples; 69% positive examples)
- Assume $P(\mathbf{x} \mid y=k)$ is multivariate normal, with mean μ_k and (common) covariance Σ

$$P(\mathbf{x} \mid y = k) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

Estimating LDA Model

- Linear discriminant analysis assumes form

$$P(\mathbf{x}, y = k) = P(y = k)\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right)$$

- μ_k is *mean* of examples belonging to class C_k (ie, y=k); covariance matrix $\sum$ is shared by all classes
- Can estimate LDA directly:

  m_k = #training examples in class y = k

  - Estimate of P(y = k): $\hat{p}_k = \frac{m_k}{m}$

$$\hat{\mu}_k = \frac{1}{m_k}\sum_{\{i: y_i = k\}} x_i$$

$$\hat{\Sigma} = \frac{1}{m}\sum_i (x_i - \hat{\mu}_{y_i})(x_i - \hat{\mu}_{y_i})^T$$

m – k

(Subtract each x_i from corresponding $\hat{\mu}_{y_i}$ before taking outer product)

Example of Estimation

x_1	x_2	x_3	y
13.1	20.2	0.4	+
6.0	17.7	-4.2	+
8.2	18.2	-2.5	+
0.4	10.1	19.2	−
-4.2	12.8	5.1	−
-4.3	15.0	21.7	−
0.9	10.1	19.2	−

- m=7 examples;
  m_+ = 3 positive; m_- = 4 negative
  $\Rightarrow$ $p_+ = 3/7$ $p_- = 4/7$

- Compute $\hat{\mu}_i$ over each class

$$\hat{\mu}_+ = \frac{1}{3}\sum_{i:\,\langle y^{(i)}=+\rangle} \mathbf{x}^{(i)}$$

$$= \frac{1}{3}\begin{pmatrix} [13.1,\ 20.2,\ 0.4]^T + \\ [\ 6.0,\ 17.7,\ -4.2]^T + \\ [\ 8.2,\ 18.2,\ -2.5]^T \end{pmatrix}$$

$$= [9.1,\ 18.7,\ -2.1]^T$$

$$\hat{\mu}_- = \frac{1}{4}\sum_{i:\,\langle y^{(i)}=-\rangle} x^{(i)} = [-1.8, 12.0, 16.3]^T$$

Note: do NOT pre-pend x_0=1!

Estimation...

x_1	x_2	x_3	y
13.1	20.2	0.4	+
6.0	17.7	-4.2	+
8.2	18.2	-2.5	+
0.4	10.1	19.2	−
-4.2	12.8	5.1	−
-4.3	15.0	21.7	−
0.9	10.1	19.2	−

- Compute common $\hat{\Sigma}$
  - "Normalize" each $\mathbf{z} := \mathbf{x} - \mu_{y(\mathbf{x})}$

$$\mathbf{z}^{(1)} := [13.1, 20.2, 0.4]^T - [9.1, 18.7, -2.1]^T = [4.0, 1.5, -1.7]^T$$

$$\ldots$$

$$\mathbf{z}^{(4)} := [0.4, 10.1, 19.2]^T - [-1.8, 12.0, 16.3]^T = [2.2, -1.9, 2.9]^T$$

  - Compute covariance matrix, for each i:

For $\mathbf{x}^{(1)}$, via $\mathbf{z}^{(1)}$:

$$\mathbf{z}^{(1)} \times \mathbf{z}^{(1)^\top} = \begin{bmatrix} 4.0 \\ 0.5 \\ -1.7 \end{bmatrix} \cdot [4.0, 0.5, -1.7]$$

$$= \begin{bmatrix} 4.0 \cdot 4.0 & 4.0 \cdot 0.5 & 4.0 \cdot -1.7 \\ 0.5 \cdot 4.0 & 0.5 \cdot 0.5 & 0.5 \cdot -1.7 \\ -1.7 \cdot 4.0 & -1.7 \cdot 0.5 & -1.7 \cdot -1.7 \end{bmatrix}$$

$$= \begin{bmatrix} 16.0 & 2.0 & -6.8 \\ 2.0 & 0.25 & -0.85 \\ -6.8 & -0.85 & -2.89 \end{bmatrix}$$

  - Set $\hat{\Sigma} = \frac{1}{7-2} \sum_{i=1..7} z^{(i)} z^{(i)T}$

Classifying, Using LDA

- How to classify new instance, given estimates

$\{\hat{p}_i\}\ \{\hat{\mu}_j\}\ \hat{\Sigma}$

Eg, $\hat{p}_+ = 3/7 \quad \hat{p}_- = 4/7$

$\star\ \hat{\mu}_+ = [9.1,\ 18.7,\ -2.1]^T$

$\hat{\mu}_- = [-1.8,\ 12.0,\ 16.3]^T$

$$\star\ \hat{\Sigma} = \begin{bmatrix} 7.22 & -1.31 & 6.35 \\ -1.31 & 2.91 & 0.32 \\ 6.35 & 0.32 & 26.03 \end{bmatrix}$$

- Class for instance **x** = $[5, 14, 6]^T$?

$$P(y=+,\ x=[5,\ 14,\ 6]^T) = P(y=+)\,P([5,\ 14,\ 6]^T \mid y=+)$$
$$= \tfrac{3}{7} \times P(x=[5,\ 14,\ 6]^T \mid x \sim \mathcal{N}(\hat{\mu}_+, \hat{\Sigma}))$$
$$= \tfrac{3}{7} \times \frac{1}{(2\pi)^{3/2}|\hat{\Sigma}|^{1/2}} \exp\left[-\tfrac{1}{2}(x-\hat{\mu}_+)^T \hat{\Sigma}^{-1}(x-\hat{\mu}_+)\right]$$
$$= 16.63\text{E-}11$$

$$P(y=-,\ x=[5,\ 14,\ 6]^T) = P(y=-)\,P([5,\ 14,\ 6]^T \mid y=-)$$
$$= \tfrac{4}{7} \times \frac{1}{(2\pi)^{3/2}|\hat{\Sigma}|^{1/2}} \exp\left[-\tfrac{1}{2}(x-\hat{\mu}_-)^T \hat{\Sigma}^{-1}(x-\hat{\mu}_-)\right]$$
$$= 43.33\text{E-}11$$

- $P(y=+ \mid [5,\ 14,\ 6]^T) = \frac{P(y=+,\ [5,\ 14,\ 6]^T)}{P(y=+,\ [5,\ 14,\ 6]^T)+P(y=-,\ [5,\ 14,\ 6]^T)} = 0.2774$

$P(y=- \mid [5,\ 14,\ 6]^T) = 0.7226$

LDA learns a LinearThresholdUnit

- Consider 2-class case with a 0/1 loss function
- Classify $\hat{y} = 1$ if

$$\ln \frac{P(y=1|x)}{P(y=0|x)} > 0 \quad \text{iff} \quad \ln \frac{P(y=1, x)}{P(y=0, x)} > 0$$

$$\frac{P(x, y=1)}{P(x, y=0)} = \frac{P(y=1)\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_1)^\top \Sigma^{-1}(x-\mu_1)\right]}{P(y=0)\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_0)^\top \Sigma^{-1}(x-\mu_0)\right]}$$

$$= \frac{P(y=1) \exp\left[-\frac{1}{2}(x-\mu_1)^\top \Sigma^{-1}(x-\mu_1)\right]}{P(y=0) \exp\left[-\frac{1}{2}(x-\mu_0)^\top \Sigma^{-1}(x-\mu_0)\right]}$$

$$\ln \frac{P(x, y=1)}{P(x, y=0)} = \ln \frac{P(y=1)}{P(y=0)} - \frac{1}{2}\left[(x-\mu_1)^\top \Sigma^{-1}(x-\mu_1) - (x-\mu_0)^\top \Sigma^{-1}(x-\mu_0)\right]$$

LDA Learns an LTU (2)

- $(x-\mu_1)^T \Sigma^{-1} (x-\mu_1) - (x-\mu_0)^T \Sigma^{-1} (x-\mu_0)$
  $= x^T \Sigma^{-1} (\mu_0 - \mu_1) + (\mu_0 - \mu_1)^T \Sigma^{-1} x +$
  $\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0$

- As Σ^{-1} is symmetric,
  $\ldots = 2\, x^T \Sigma^{-1} (\mu_0 - \mu_1) + \mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0$

$$\Rightarrow \ln \frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0)} =$$

$$\ln \frac{P(y=1)}{P(y=0)} - \frac{1}{2}\left[(\mathbf{x}-\mu_1)^\top \Sigma^{-1}(\mathbf{x}-\mu_1) - (\mathbf{x}-\mu_0)^\top \Sigma^{-1}(\mathbf{x}-\mu_0)\right]$$

$$= \ln \tfrac{P(y=1)}{P(y=0)} + \mathbf{x}^\top \Sigma^{-1}(\mu_1-\mu_0) + \tfrac{1}{2}\mu_0^\top \Sigma^{-1}\mu_0 - \tfrac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1$$

$$= \mathbf{x}^\top \Sigma^{-1}(\mu_1-\mu_0) + \ln \tfrac{P(y=1)}{P(y=0)} + \tfrac{1}{2}\mu_0^\top \Sigma^{-1}\mu_0 - \tfrac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1$$

LDA Learns an LTU (3)

$$\ln \frac{P(\mathbf{x}, y=1)}{P(\mathbf{x}, y=0)} = \mathbf{x}^\top \Sigma^{-1}(\mu_1 - \mu_0) + \ln \frac{P(y=1)}{P(y=0)} + \frac{1}{2}\mu_0^\top \Sigma^{-1}\mu_0 - \frac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1$$

- So let…

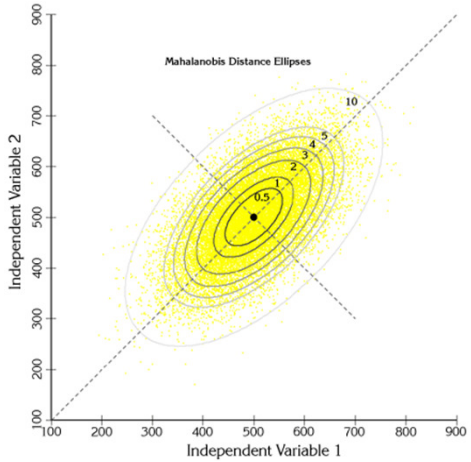
$$\mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_0)$$

$$w_0 = \ln \frac{P(y=1)}{P(y=0)} + \frac{1}{2}\mu_0^\top \Sigma^{-1}\mu_0 - \frac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1$$

- Classify $\hat{y} = 1$ iff $\mathbf{w}^\top \mathbf{x} + w_0 > 0$

  Linear Threshold Unit !!

View LDA wrt Mahalanobis Distance



- Squared Mahalanobis distance between **x** and μ

$$D_M^2(\mathbf{x}, \mu) = (\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)$$

  - $\Sigma^{-1} \approx$ linear distortion

  … converts standard Euclidean distance into Mahalanobis distance.

- LDA classifies **x** as Class1 if

$$D_M^2(\mathbf{x}, \mu_1) < D_M^2(\mathbf{x}, \mu_2)$$

- $\log P(\mathbf{x} \mid y = k) \approx \log p_k - \frac{1}{2} D_M^2(\mathbf{x}, \mu_k)$

Generalizations of LDA

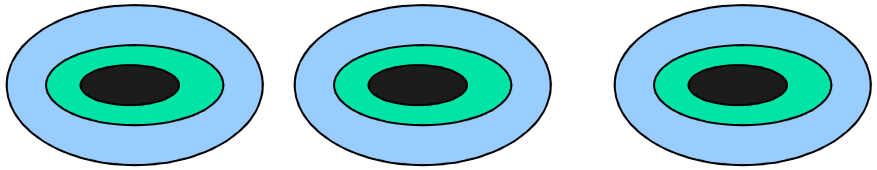
- **SuperSimple Classifier**
  - Same $\sum$ for all classes
  - This $\sum$ is diagonal

    $\Rightarrow$ within each class, each pair of features x_i and x_j is independent
- **Naïve Gaussian Classifier**
  - Allow each class *k* to have its own $\sum_k$

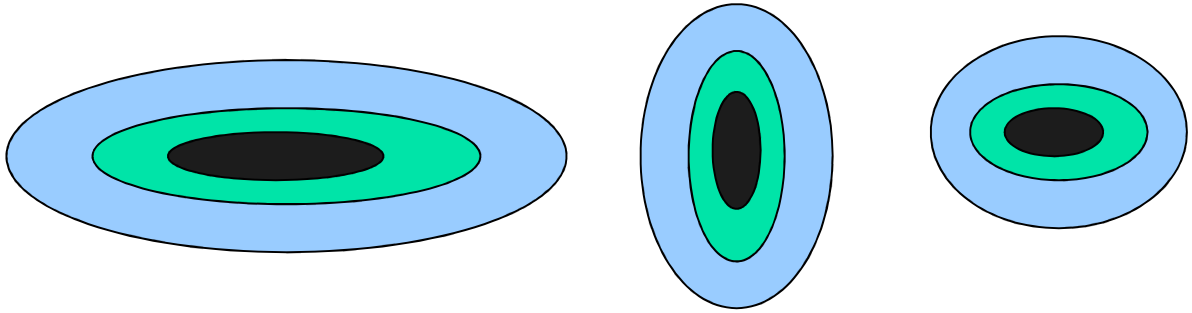
    $\Rightarrow$ Classifier $\equiv$ ***quadratic*** threshold unit (not LTU)
  - Each $\sum_k$ is diagonal
- **Linear Discriminate Analysis**
  - Same $\sum$ for all classes
  - This $\sum$ is arbitrary
- **General Gaussian Classifier (QDA)**
  - Allow each class k to have its own $\sum_k$
  - Each $\sum_k$ is arbitrary

Versions of ?L?Q?N? DA

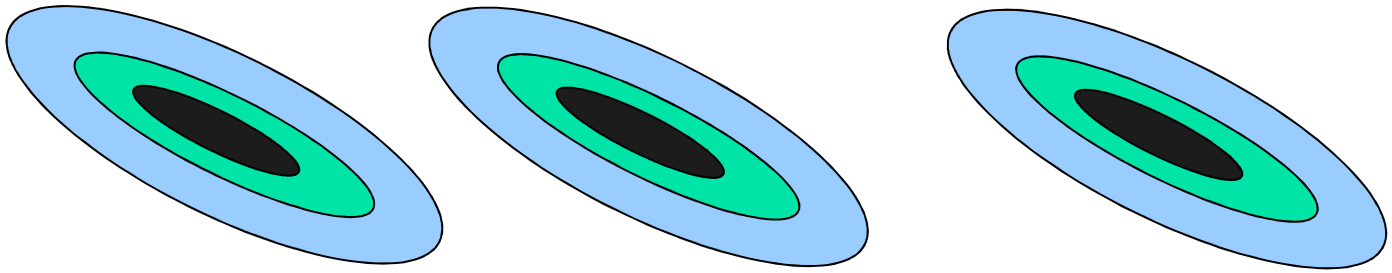
- SuperSimple



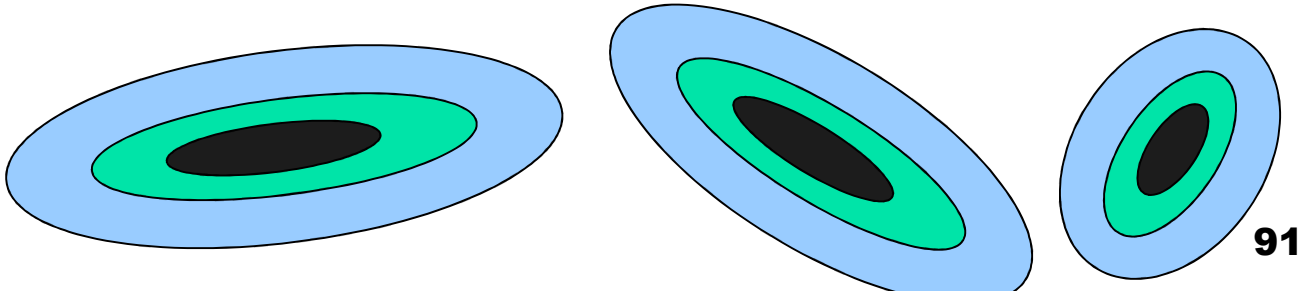
- Naïve



- LDA



- Quadratic



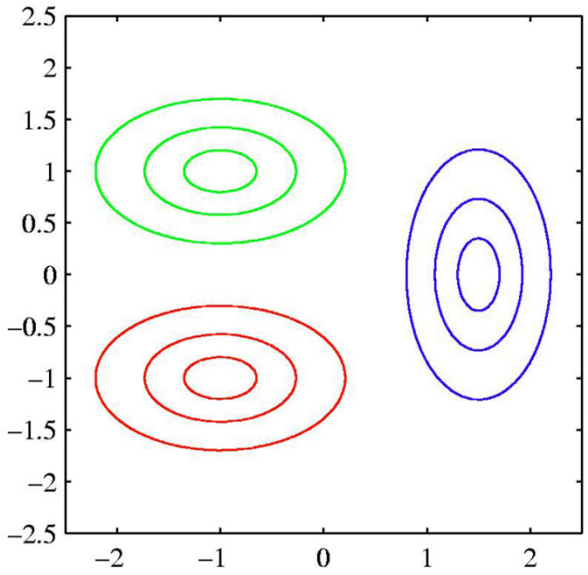
Variants of LDA

- Covariance matrix Σ
- n features; k classes

Name	Same for all classes?	Diagonal	#param's
SuperSimple	+	+	k
Naïve Gaussian Classifier	–	+	k n
LDA	+	–	n^2
General Gaussian Classifier	–	–	k n^2

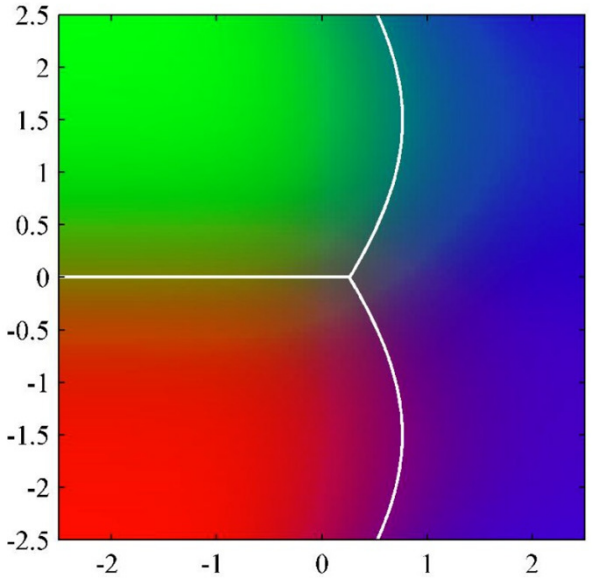
Quadratic Discriminant

- The decision boundary is linear when the covariance matrices $\Sigma = \Sigma_1 = \ \ldots = \Sigma_k$ are the same
- … BUT quadratic when they are not



Class-conditional densities for three classes;

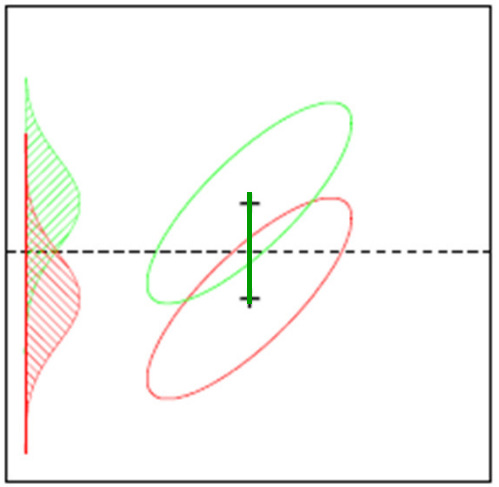
$\Sigma_{red} \neq \Sigma_{blue}$

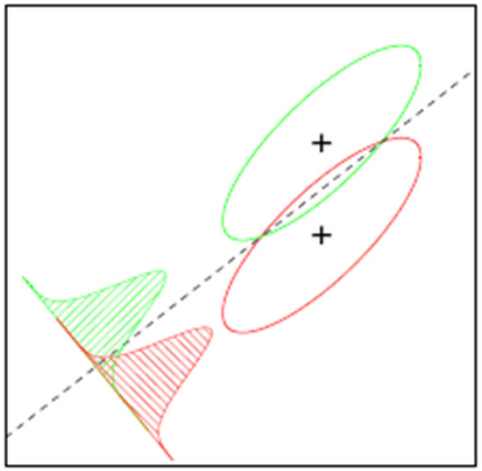


The corresponding posterior probabilities for three classes.

Fisher's Linear Discriminant

- LDA
  - Finds K–1 n-dim hyperplanes
    (K = number of classes)
  - Project **x** and $\{ \mu_k \}$ to that hyperplane
  - Classify **x** as nearest μ_k within hyperplane
- Better:
  Find hyperplane that maximally separates projection of **x**'s wrt $\sum^{-1}$

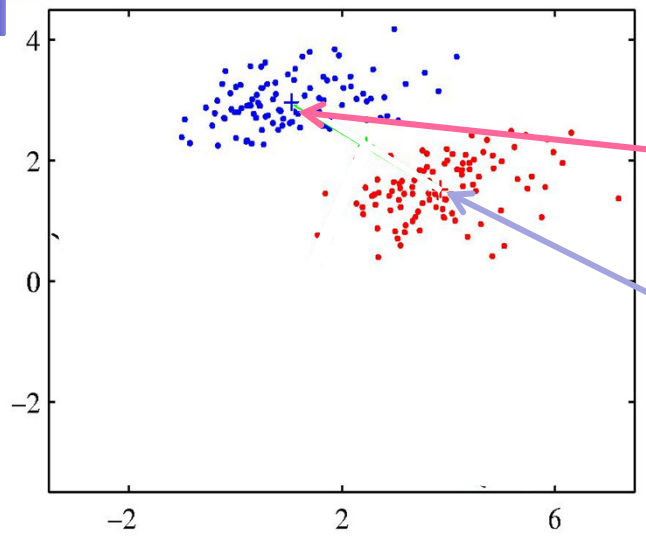




⇒ Fisher's Linear Discriminant

Skip

Fisher Linear Discriminant

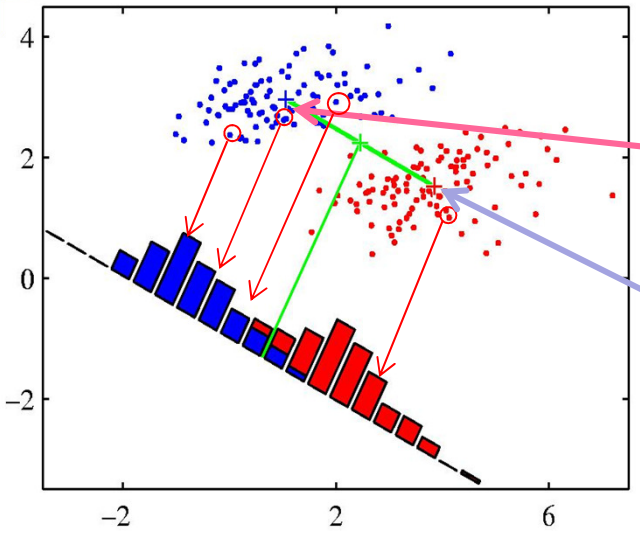


$$\mathbf{m}_+ = \frac{\sum_i y^{(i)} \mathbf{x}^{(i)}}{\sum_i y^{(i)}}$$

$$\mathbf{m}_- = \frac{\sum_i (1 - y^{(i)})\, \mathbf{x}^{(i)}}{\sum_i (1 - y^{(i)})}$$

- Recall any vector **w** projects $\Re^n \rightarrow \Re$
- Goal: Want **w** that "separates" classes
  - Each $\mathbf{w}^T \mathbf{x}^+$ far from each $\mathbf{w}^T \mathbf{x}^-$
- Perhaps project onto $\mathbf{m}_+ - \mathbf{m}_-$?

Fisher Linear Discriminant

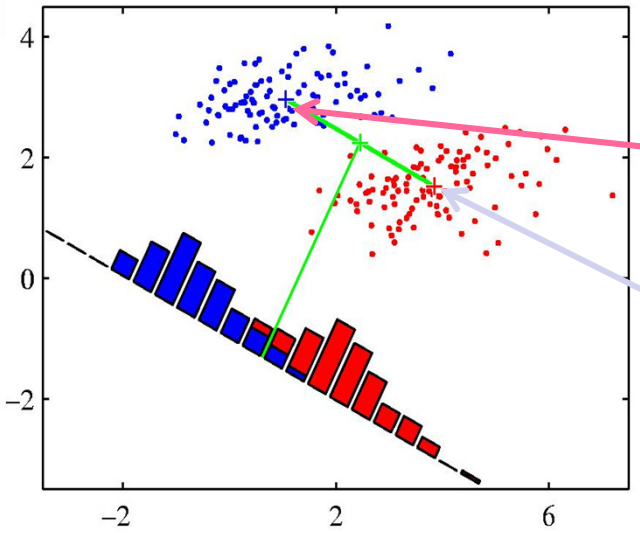


$$\mathbf{m}_+ = \frac{\sum_i y^{(i)}\mathbf{x}^{(i)}}{\sum_i y^{(i)}}$$

$$\mathbf{m}_- = \frac{\sum_i (1 - y^{(i)})\,\mathbf{x}^{(i)}}{\sum_i (1 - y^{(i)})}$$

- Recall any vector **w** projects $\Re^n \rightarrow \Re$
- Goal: Want **w** that "separates" classes
  - Each $\mathbf{w}^T \mathbf{x}^+$ far from each $\mathbf{w}^T \mathbf{x}^-$
- Perhaps project onto $\mathbf{m}_+ - \mathbf{m}_-$?
- Still overlap… why?

Fisher Linear Discriminant

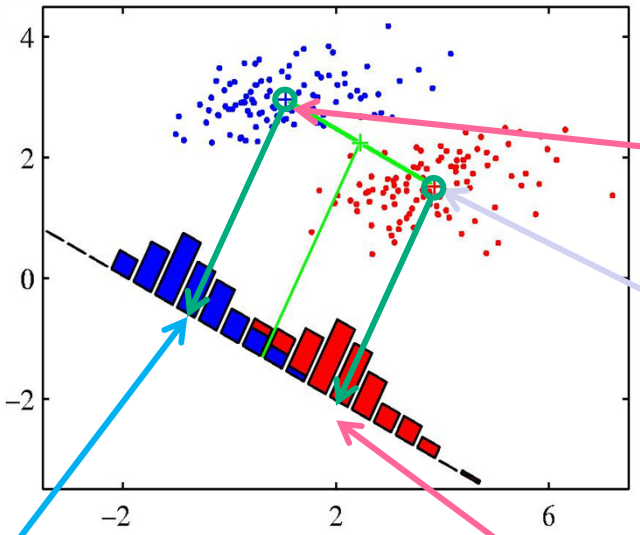


$$\mathbf{m}_+ = \frac{\sum_i y^{(i)} \mathbf{x}^{(i)}}{\sum_i y^{(i)}}$$

$$\mathbf{m}_- = \frac{\sum_i (1 - y^{(i)})\, \mathbf{x}^{(i)}}{\sum_i (1 - y^{(i)})}$$

- Problem with $\mathbf{m}_+ - \mathbf{m}_-$:
  - Does not consider "scatter" within class
- Also want $\mathbf{w}$ that "separates" classes
  - Positive $\mathbf{x}^+$'s: $\{ \mathbf{w}^T \mathbf{x}^+ \}$ close to each other
  - Negative $\mathbf{x}^-$'s: $\{ \mathbf{w}^T \mathbf{x}^- \}$ close to each other

Fisher Linear Discriminant



$$\mathbf{m}_+ = \frac{\sum_i y^{(i)}\mathbf{x}^{(i)}}{\sum_i y^{(i)}}$$

$$\mathbf{m}_- = \frac{\sum_i (1 - y^{(i)})\,\mathbf{x}^{(i)}}{\sum_i (1 - y^{(i)})}$$

$$\mu_+ = \frac{\sum_i y^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}}{\sum_i y^{(i)}}$$

$$\mu_- = \frac{\sum_i (1 - y^{(i)})\mathbf{w}^T\mathbf{x}^{(i)}}{\sum_i (1 - y^{(i)})}$$

- Also want **w** that "separates" classes
  - Positive $\mathbf{x}^+$'s: $\{ \mathbf{w}^T \mathbf{x}^+ \}$ close to each other
  - Negative $\mathbf{x}^-$'s: $\{ \mathbf{w}^T \mathbf{x}^- \}$ close to each other
- "scatter" of +instances; –instances
  - $\mathbf{s}_+^2 = \sum_i \; y^{(i)} \; [\mathbf{w}^T \mathbf{x}^{(i)} - \mathbf{w}^T \mathbf{m}_+]^2 = \sum_i y^{(i)} [\mathbf{w}^T \mathbf{x}^{(i)} - \mu_+]^2$
  - $\mathbf{s}_-^2 = \sum_i (1 - y^{(i)}) [\mathbf{w}^T \mathbf{x}^{(i)} - \mathbf{w}^T \mathbf{m}_-]^2 = \sum_i (1 - y^{(i)}) [\mathbf{w}^T \mathbf{x}^{(i)} - \mu_-]^2$

FLD, con't

1. Separate means μ_+ and μ_-

   $\Rightarrow$ maximize $(\mu_+ - \mu_-)^2$

2. Minimize each spread $\mathbf{s_+^2}$, $\mathbf{s_-^2}$

   $\Rightarrow$ minimize $(\mathbf{s_+^2} + \mathbf{s_-^2})$

- Objective function:

  maximize $J_S(\mathbf{w}) = (\mu_+ - \mu_-)^2$
  s.t. $s_+^2 + s_-^2$ is small

#1: $(\mu_- - \mu_+)^2 = (\mathbf{w}^T \mathbf{m_+} - \mathbf{w}^T \mathbf{m_-})^2$

$= \mathbf{w}^T (\mathbf{m_+} - \mathbf{m_-})(\mathbf{m_+} - \mathbf{m_-})^T \mathbf{w} = \mathbf{w}^T S_B \mathbf{w}$

"between-class scatter" $S_B = (\mathbf{m_+} - \mathbf{m_-})(\mathbf{m_+} - \mathbf{m_-})^T$

FLD, III

maximize $J_S(\mathbf{w}) = (\mu_+ - \mu_-)^2$
s.t. $s_+^2 + s_-^2$ is small

- $\mathbf{s_+^2} = \sum_i y^{(i)} [\mathbf{w}^T (\mathbf{x^{(i)}} - \mathbf{m_+})]^2$
  $= \sum_i \mathbf{w}^T y^{(i)} (\mathbf{x^{(i)}} - \mathbf{m_+}) (\mathbf{x^{(i)}} - \mathbf{m_+})^T \mathbf{w}$
  $= \mathbf{w}^T S_+ \mathbf{w}$

$$S_+ = \sum_i y^{(i)} (\mathbf{x^{(i)}} - \mathbf{m_+}) (\mathbf{x^{(i)}} - \mathbf{m_+})^T$$

… "within-class scatter matrix" for +

$$S_- = \sum_i (1 - y^{(i)}) (\mathbf{x^{(i)}} - \mathbf{m_-}) (\mathbf{x^{(i)}} - \mathbf{m_-})^T$$

… "within-class scatter matrix" for –

- $S_R = S_+ + S_-$ so $\mathbf{s_+^2} + \mathbf{s_-^2} = \mathbf{w}^T S_R \mathbf{w}$

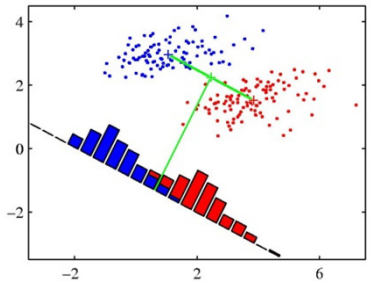
FLD, IV

$$J_S(\mathbf{w}) = \mathbf{w}^T S_B^{-1} S_R \mathbf{w}$$

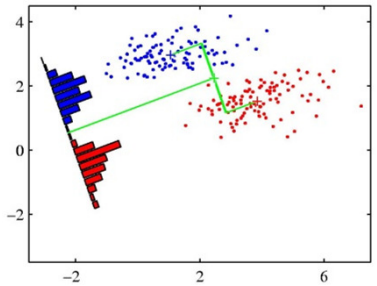
- Minimizing $J_S(\mathbf{w})$ …
  $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\ \mathbf{w}^T\mathbf{S_B}\mathbf{w}$ s.t. $\mathbf{w}^T\mathbf{S_R}\mathbf{w} = 1$
- Lagrange: $L(\mathbf{w}, \lambda) = \mathbf{w}^T\mathbf{S_B}\mathbf{w} + \lambda\,(1 - \mathbf{w}^T\mathbf{S_R}\mathbf{w})$

$$\frac{\partial L(\mathbf{w},\lambda)}{\partial \mathbf{w}} = 2S_B\mathbf{w} - \lambda(2S_R\mathbf{w})$$

$$\frac{\partial L(\mathbf{w},\lambda)}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad S_B^{-1}S_R\mathbf{w} = \frac{1}{\lambda}\mathbf{w}$$



- … $\mathbf{w}^*$ is eigenvector of $S_B^{-1}S_R$



FLD, IV

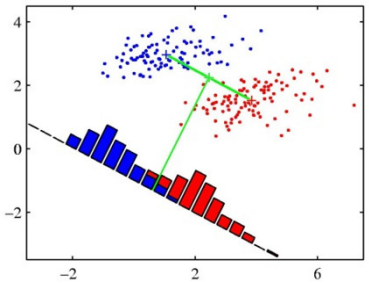
$$J_S(\mathbf{w}) = \mathbf{w}^T S_B^{-1} S_R \mathbf{w}$$

- Want $\mathbf{w}^T\mathbf{S}_R\mathbf{w}$ to be small … say $\mathbf{w}^T\mathbf{S}_R\mathbf{w} = 1$
- Minimizing $J_S(\mathbf{w})$ …

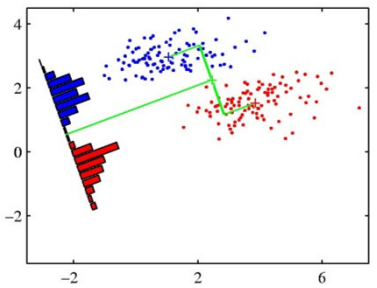
  $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\ \mathbf{w}^T\mathbf{S}_B\mathbf{w}$ s.t. $\mathbf{w}^T\mathbf{S}_R\mathbf{w} = 1$
- Lagrange: $L(\mathbf{w}, \lambda) = \mathbf{w}^T\mathbf{S}_B\mathbf{w} + \lambda\,(1 - \mathbf{w}^T\mathbf{S}_R\mathbf{w})$

$$\frac{\partial L(\mathbf{w},\lambda)}{\partial \mathbf{w}} = 2S_B\mathbf{w} - \lambda(2S_R\mathbf{w})$$

$$\frac{\partial L(\mathbf{w},\lambda)}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad S_B^{-1}S_R\mathbf{w} = \frac{1}{\lambda}\mathbf{w}$$



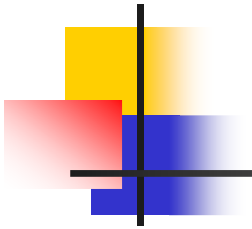
- … $\mathbf{w}^*$ is eigenvector of $S_B^{-1}S_R$



FLD, V

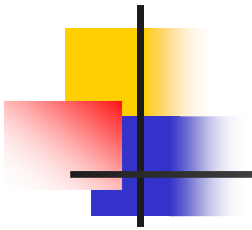
$$J_S(\mathbf{w}) = \mathbf{w}^T S_B^{-1} S_R \mathbf{w}$$

- **Optimal $\mathbf{w}^*$** is eigenvector of $S_B^{-1}S_w$
- When $P(x \mid y_k) \sim N(\mu_k; \Sigma)$
  $\exists$ LINEAR DISCRIMINANT: $\mathbf{w} = \Sigma^{-1}(\mu_+ - \mu_-)$
  $\Rightarrow$ FLD is optimal classifier,
  if classes normally distributed
- Can use even if not Gaussian:
  After projecting *d*-dim to 1,
  just use any classification method
- Analogous derivation for K > 2 classes



Fisher's LD vs LDA

- Fisher's LD = LDA when…
  - Prior probabilities are same
  - Each class conditional density is multivariate Gaussian
  - … with common covariance matrix
- But … Fisher's LD…
  - does not assume Gaussian densities
  - can be used to reduce dimensions even when multiple classes scenario

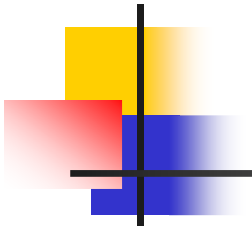


Summary of Linear Discriminant Analysis

- **Learns Joint Probability Distr'n** P(y, **x**)

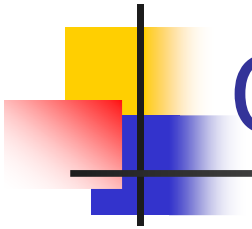
- **Direct Computation**.
  MLEstimate of P(y, **x**) computed directly from data...
  without search
  But need to invert matrix, which is $O(p^3)$

- **Batch**: Only a batch algorithm
  - An online LDA alg requires online alg for incrementally updating $\sum^{-1}$
  - [Easy if $\sum^{-1}$ is diagonal; otherwise ...]



Other issues wrt Linear Classifiers

- Yet other linear models
- What if >2 classes
- What is “evaluation” … “best”?
- Which model is best?
  What are the issues?

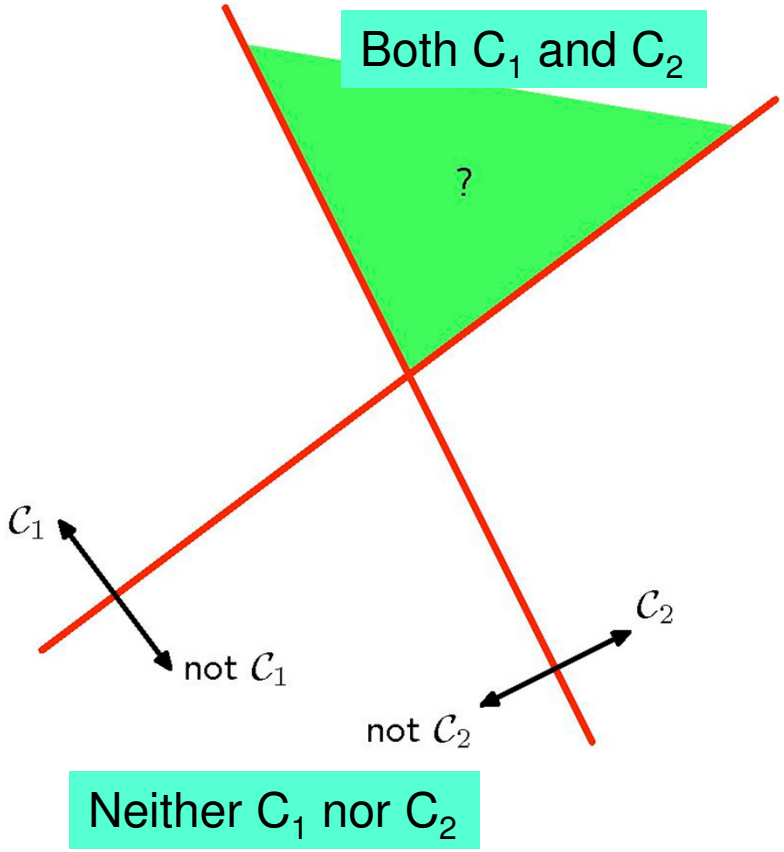


Other Algorithms for learning LTUs

- **Support Vector Machine** (SVM)
  - Linear kernel… next topic!
- **Naive Bayes** [Discuss later]
  For K = 2 classes, produces LTU
- **LASSO, LARS,** …
  Remove features
- **Winnow** [?Discuss later?]
  Can handle large numbers of "irrelevant" features
  - (features whose weights should be zero)

Multiple Classes

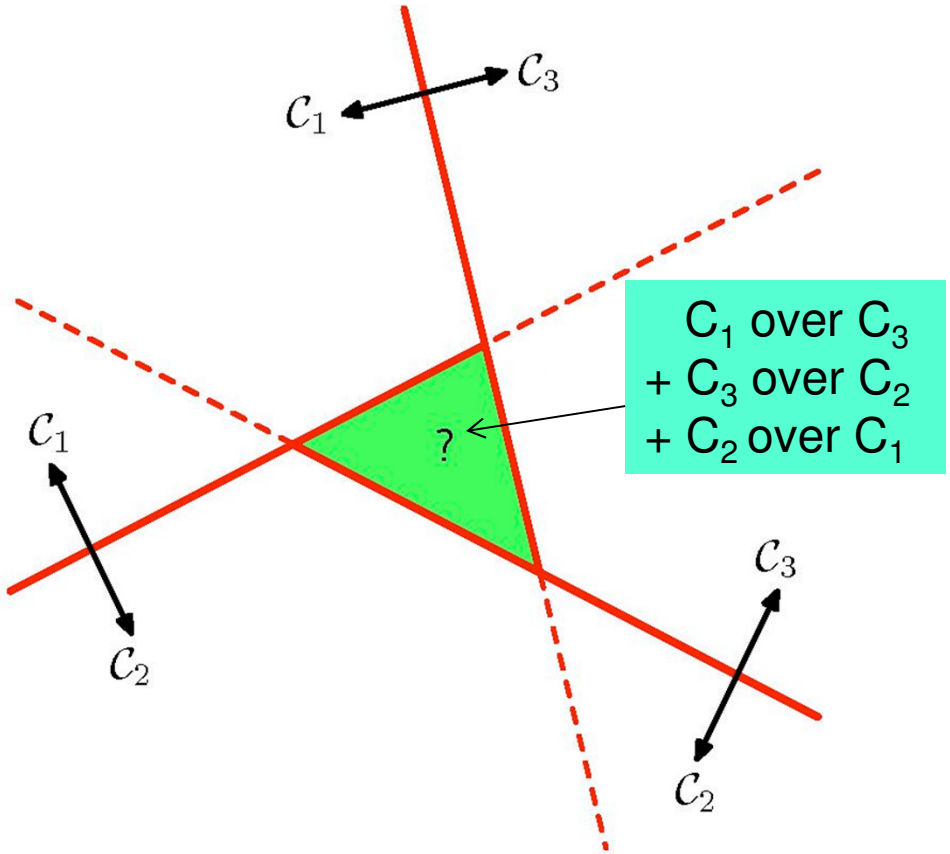
- How to extend linear discriminants to K>2 classes?
- ? perhaps use K-1 *One-versus-the-rest* classifiers
  - each separates points in class C_k from points not in that class



- NO: ∃ ambiguous regions in input space

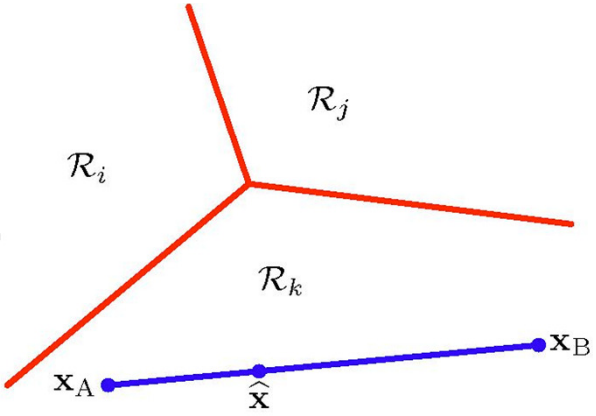
Multiple Classes

- How to extend linear discriminants to K>2 classes?
- ? perhaps use K(K-1)/2 *binary discriminators*
  - each discriminates between 2 classes
  - take majority vote



- NO: ∃ ambiguous regions in input space

Simple Solution



- Find K linear functions of the form:

  $y_k(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_k + w_{k0}$ for k=1..K

  - Assign $\mathbf{x}$ to class $C_k = \text{argmax}_k\ y_j(\mathbf{x})$
- Decision regions are singly connected, convex
  - Consider any two points in R_k:

    $\forall j \neq k$: $y_k(\mathbf{x}_A) > y_j(\mathbf{x}_A)$, $y_k(\mathbf{x}_B) > y_j(\mathbf{x}_B)$
  - As discriminant functions are linear:

$$y_k(\alpha\,\mathbf{x}_A + (1-\alpha)\mathbf{x}_B) =$$
$$\alpha\ y_k(\mathbf{x}_A) + (1-\alpha) y_k(\mathbf{x}_B)$$
$$> \alpha\ y_j(\mathbf{x}_A) + (1-\alpha) y_j(\mathbf{x}_B) = y_j(\alpha\,\mathbf{x}_A + (1-\alpha)\mathbf{x}_B)$$

- Challenge: learning these K functions…

What is Evaluation?

- So far, thinking about Discriminative Task
  - Comparable # of positives, negatives
  - Penalty for False **positive** ≈ Penalty for False **negative**
  - Misclassification costs:

0	1
1	0

0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

- But what if…
  - Imbalanced data … eg, 99% are +'s ?
    - Get accuracy = 99% trivially!
  - But not what you want, if HUGE penalty for false –'s

Comparing LMS, Logistic Regression, LDA, FLD

- Which is best: *LMS*, *LR*, *LDA*, *FLD* ?
- Ongoing debate within machine learning community about relative merits of
  - direct classifiers [*LMS*]
  - conditional models P(y | **x**) [*LR*]
  - generative models P(y, **x**) [*LDA, FLD*]

Issues in Debate

- **Statistical efficiency**
  If generative model P(y, x) is correct, then …
  usually gives better accuracy, particularly if training sample is small
- **Computational efficiency**
  Generative models typically easiest to compute
  (LDA/FLD computed directly, without iteration)
- **Robustness to model assumptions**
  Generative model usually performs poorly when assumptions are violated
  - Eg, LDA works poorly if P(x | y) is non-Gaussian
  - Logistic Regression makes fewer assumptions… ⇒ more robust
    … LMS is even more robust
- **Robustness to missing values and noise**
  Some $x^{(i)}_j$ values may be missing or corrupted
  Generative models typically provide better ways of handling this
- **Robustness to changing loss functions**
  LMS must re-train the classifier when the *loss function* changes
  … retraining not needed for generative and conditional models