HTF: Ch4
B: Ch4

Linear Classifiers

R Greiner

Cmput 466/551

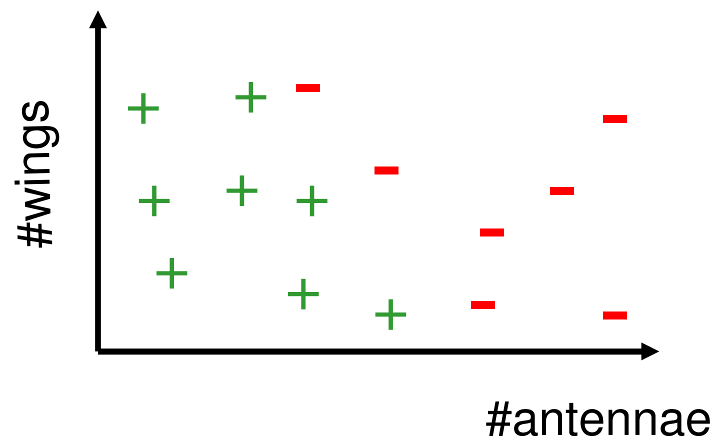
Outline

- Framework
- Exact
  - Minimize Mistakes (Perceptron Training)
  - Matrix inversion (LMS)
- Logistic Regression
  - Max Likelihood Estimation (MLE) of $P(y \mid \mathbf{x})$
  - Gradient descent (MSE; MLE)
  - Newton-Raphson
- Linear Discriminant Analysis
  - Max Likelihood Estimation (MLE) of $P(y, \mathbf{x})$
  - Direct Computation
  - Fisher's Linear Discriminant

Diagnosing Butterfly-itis

Classifier: Decision Boundaries

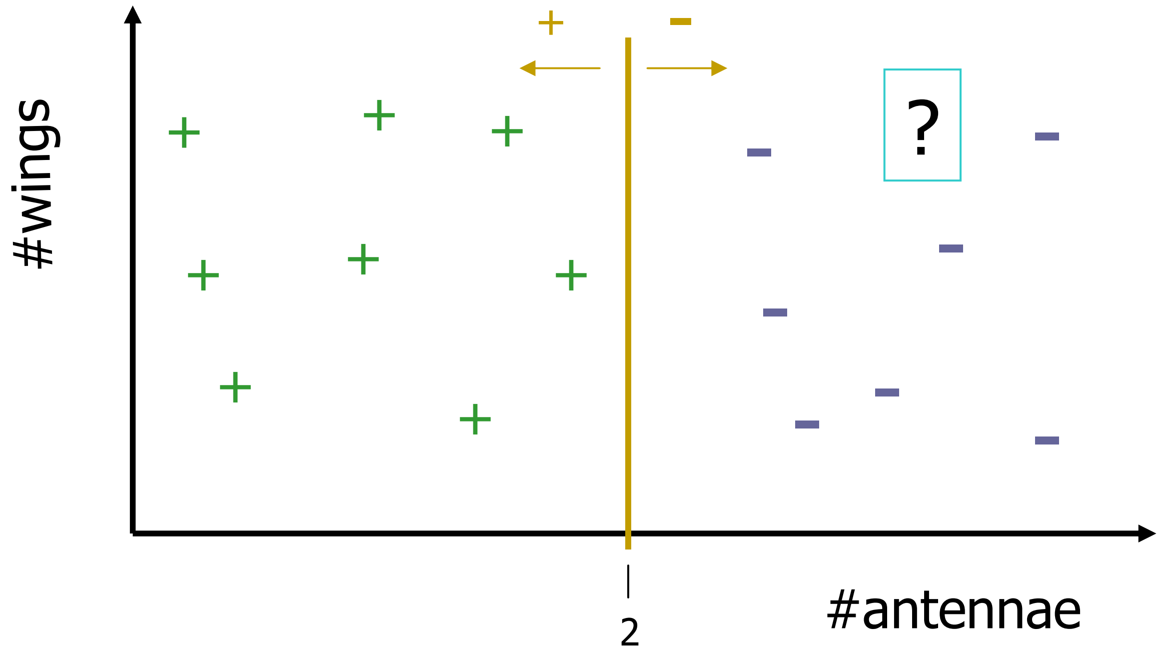
- **Classifier**: partitions input space X into "decision regions"



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

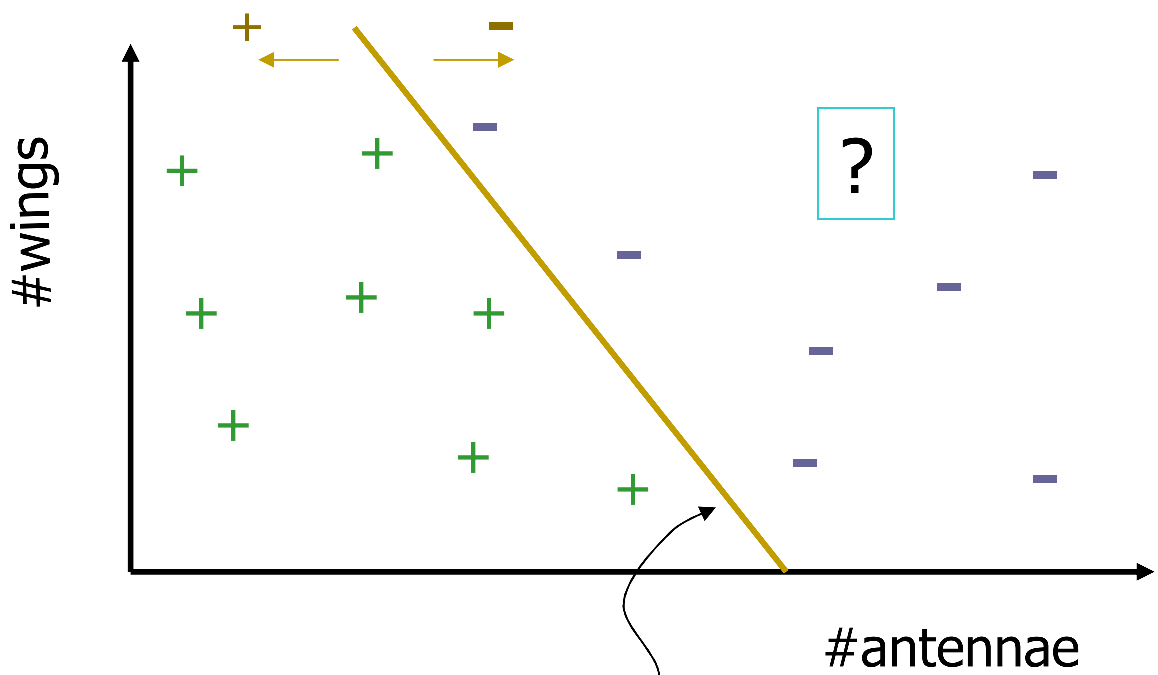
Linear Separators

- Draw "separating line"



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

Can be "angled"…



$2.3 \times \#w \; - \; 7.5 \times \#a \; + \; 1.2 \; = \; 0$

- If $\boxed{2.3} \times \#Wings - \boxed{7.5} \times \#antennae + \boxed{1.2} \; > \; 0$
  then butterfly-itis

Linear Separators, in General

- Given data (many features)

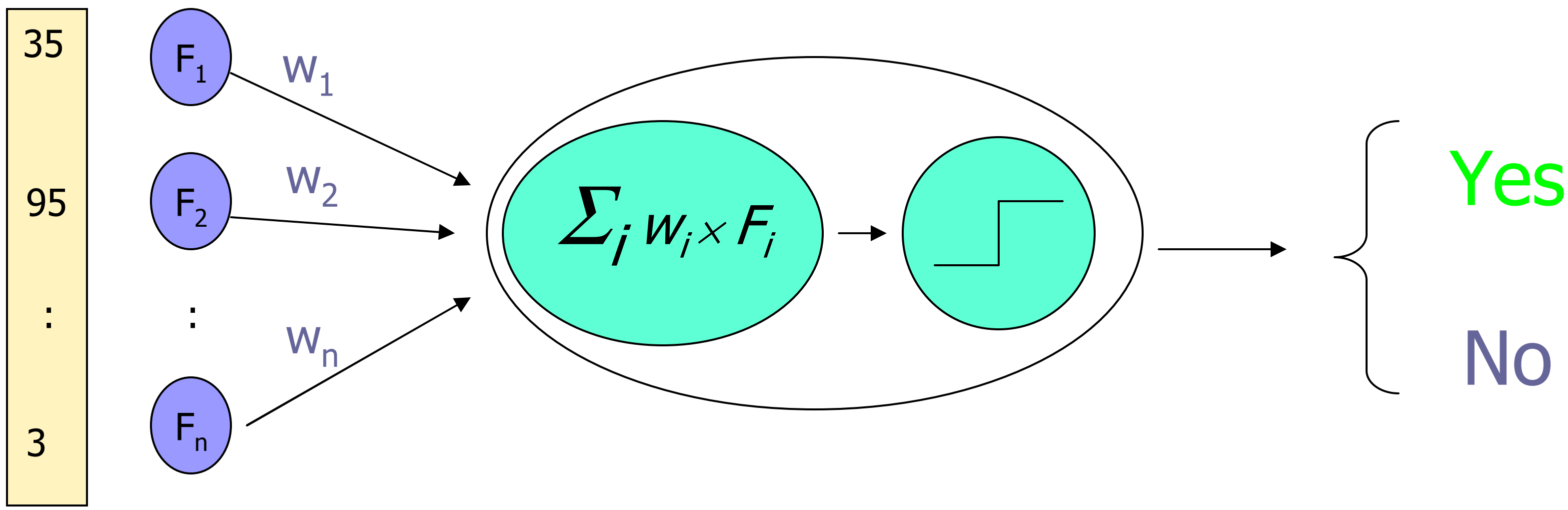
F_1	F_2	...	F_n	Class
35	95	...	3	No
22	80	...	-2	Yes
:	:		:	:
10	50	...	1.9	No

- find "weights" $\{w_1, w_2, \ldots, w_n, w_0\}$ such that

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

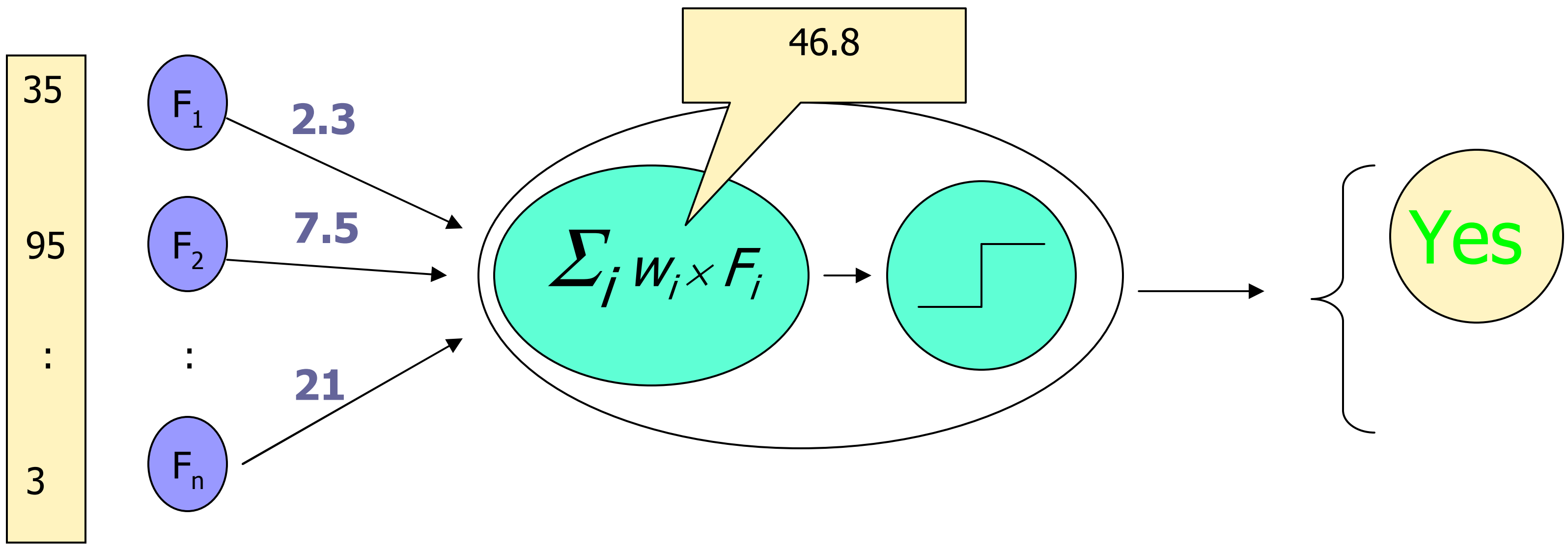
means $\boxed{Class = Yes}$

Linear Separator



Just view $F_0 = 0$, so w_0 …

Linear Separator



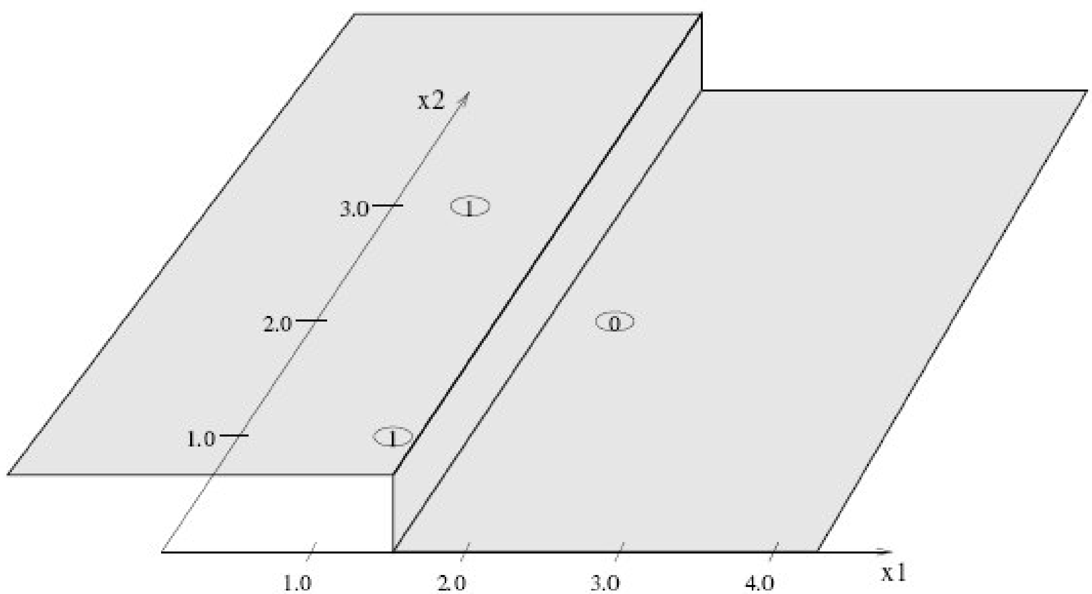
- Performance
  - Given $\{w_i\}$, and values for instance, compute response
- Learning
  - Given labeled data, find "correct" $\{w_i\}$
- Linear Threshold Unit … "Perceptron"

Geometric View

- Consider 3 training examples:

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

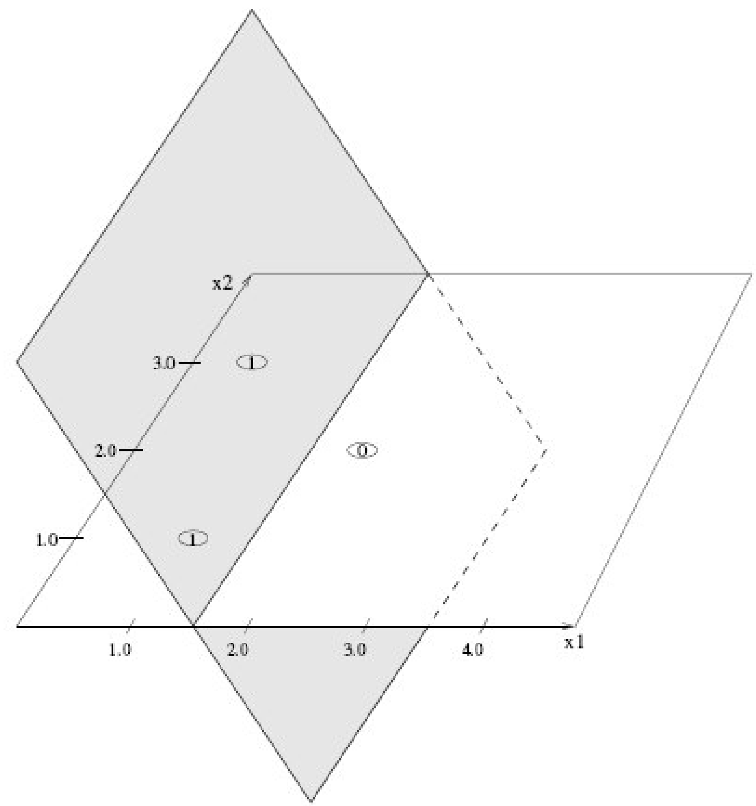
- Want classifier that looks like. . .



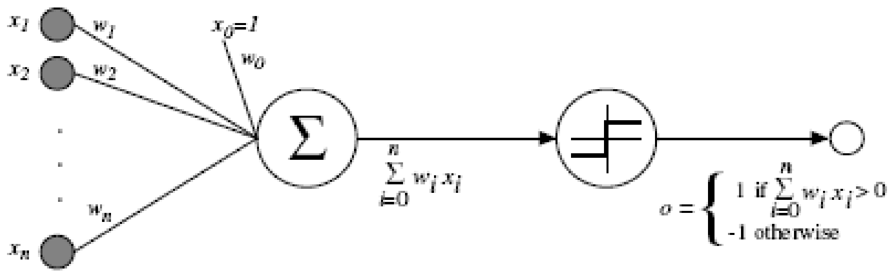
Linear Equation is Hyperplane

- Equation $\mathbf{w} \cdot \mathbf{x} = \sum_i w_i \cdot x_i$ is plane

$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ 0 & \text{otherwise} \end{cases}$$



Linear Threshold Unit: “Perceptron”



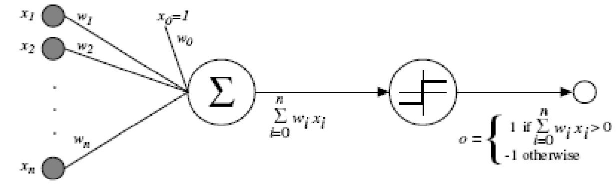
$$
\begin{aligned}
o_{\mathbf{w}}(x_1, \ldots, x_n) &= \begin{cases} 1 & \text{if } \; w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases} \\
&= \text{sign}(\; (w_0, w_1, \ldots, w_n) \cdot (1, x_1, \ldots, x_n))
\end{aligned}
$$

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

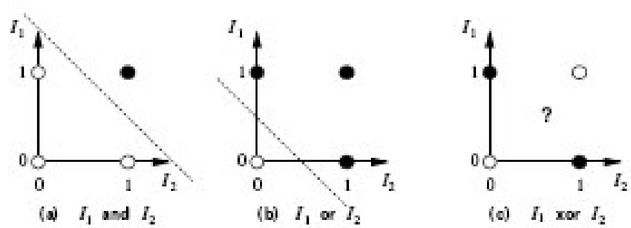
  $$\text{sgn}(r) = \begin{cases} 1 & \text{if } r > 0 \\ 0 & \text{otherwise} \end{cases}$$

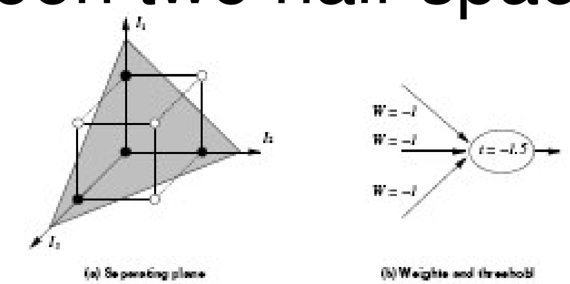
  (“Heaviside”)
- Actually $\mathbf{w} \cdot \mathbf{x} > b$ but. . .
  Create extra input x_0 fixed at 1
  Corresponding w_0 corresponds to -b

Learning Perceptrons



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





- 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 fns cannot be represented
  … killed the field temporarily!

Perceptron Learning

- Hypothesis space is. . .
  - **Fixed Size:**
    $\exists$ $O(2^{n^{\wedge}2})$ distinct perceptrons over n boolean features
  - **Deterministic**
  - **Continuous** Parameters
- Learning algorithm:
  - Various: **Local** search, **Direct** computation, . . .
  - **Eager**
  - **Online** / **Batch**

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	−
$\vdots$				$\vdots$	$\vdots$
−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	−
$\vdots$	$\vdots$				$\vdots$	$\vdots$
1	−16.2	−77.0	−1.2	$\cdots$	−4.0	+

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

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

  $$\forall i \;\; \mathrm{sgn}(\, \mathbf{w} \cdot [1, \mathbf{x}^{(i)}\,]) = y^{(i)}$$

  - . . . minimize mistakes wrt data . . .

Error Function

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

- 1. Classification error

  Perceptron Training; Matrix Inversion

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

- 2. Mean-squared error (LMS)

  Matrix Inversion; Gradient Descent

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

- 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)

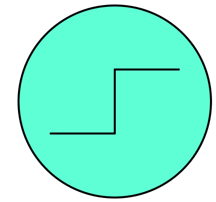
  Direct Computation

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

#1: Optimal Classification Error

- For each labeled instance $[\mathbf{x}, y]$

  $Err = y - o_{\mathbf{w}}(\mathbf{x})$

  $y = f(x)$ is target value

  $o_{\mathbf{w}}(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$ is perceptron output
- **Idea**: Move weights in appropriate direction, to push $Err \rightarrow 0$
- If $Err > 0$ (error on POSITIVE example)
  - need to increase $\mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$

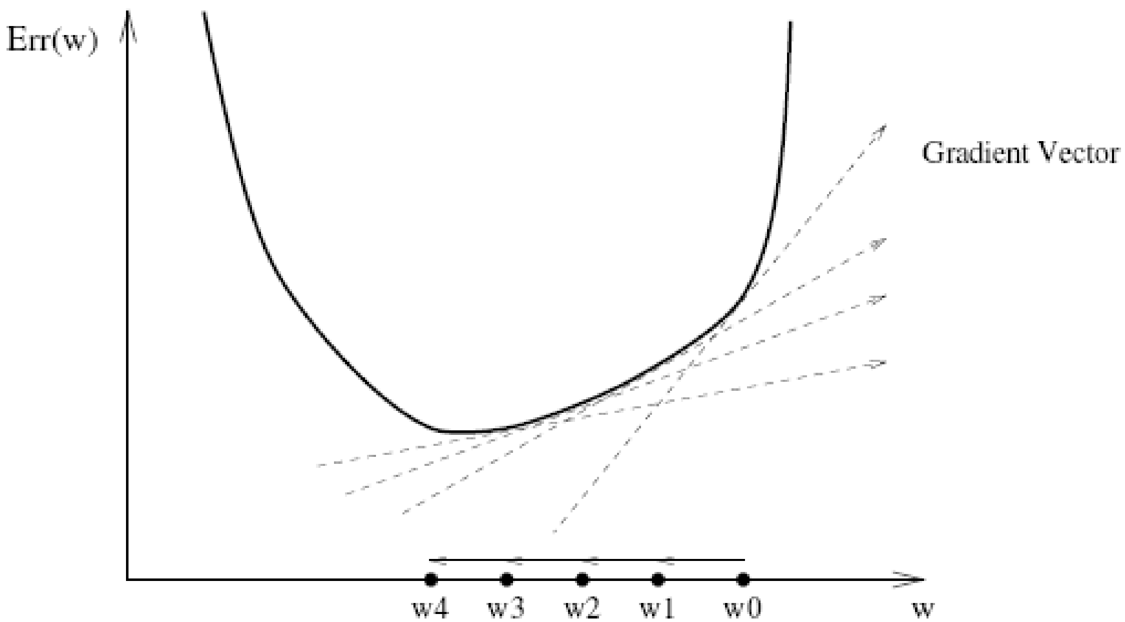
    $\Rightarrow$ need to increase $\mathbf{w} \cdot \mathbf{x}$
  - Input j contributes $w_j \cdot x_j$ to $\mathbf{w} \cdot \mathbf{x}$
    - if $x_j > 0$, increasing w_j will increase
    - if $x_j < 0$, decreasing

$\Rightarrow w_j \leftarrow w_j + x_j$

- If $Err < 0$ (error on NEGATIVE example)

  $\Rightarrow w_j \leftarrow w_j - x_j$

Local Search via Gradient Descent



Start w/ (random) weight vector $\mathbf{w}^0$.
Repeat until converged $\vee$ bored

Compute Gradient

$$\nabla \mathrm{err}(\mathbf{w}^t) = \left(\frac{\partial \mathrm{err}(\mathbf{w}^t)}{\partial w_0}, \frac{\partial \mathrm{err}(\mathbf{w}^t)}{\partial w_1}, \ldots, \frac{\partial \mathrm{err}(\mathbf{w}^t)}{\partial w_n} \right)$$

Let $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \nabla \mathrm{err}(\mathbf{w}^t)$

If CONVERGED: Return($\mathbf{w}^t$)

#1a: Mistake Bound Perceptron Alg

Initialize **w** = 0
Do until bored
  Predict "+" iff **w** · **x** > 0
    else "–"
  Mistake on y = +1: **w** ←**w** + **x**
  Mistake on y = –1: **w** ←**w** – **x**

Weights	Instance	Action
[0 0 0]	#1	

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	+

Mistake Bound Theorem

See SVM…

Theorem: [Rosenblatt 1960]

If data is consistent w/some linear threshold **w**,

then number of mistakes is $\leq (1/\Delta)^2$,

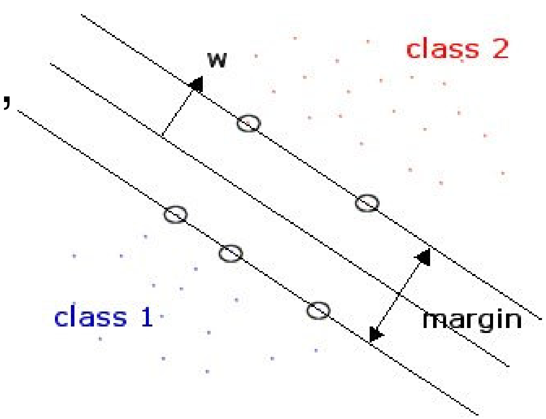
where $$\Delta = \min_x \frac{|\mathbf{w} \cdot x|}{|\mathbf{w}| \times |x|}$$

- Δ measures "wiggle room" available:

  If $|x| = 1$, then Δ is max, over all consistent planes, of minimum distance of example to that plane

- w is $\perp$ to separator, as $w \cdot x = 0$ at boundary
- So $|\mathbf{w} \cdot \mathbf{x}|$ 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:

0. Use $x_0 \equiv 1$, so target plane goes thru 0
1. Assume target plane doesn't hit any examples
2. Replace negative point $\langle\langle x_0, x_1, \ldots, x_n\rangle\ 0\rangle$
   by positive point $\langle\langle -x_0, -x_1, \ldots, -x_n\rangle\ 1\rangle$
3. Normalize all examples to have length 1

x wrong wrt **w** iff $\mathbf{w} \cdot \mathbf{x} < 0$

- Let $\mathbf{w}^*$ be unit vector rep'ning target plane
  $\Delta = \min_{\mathbf{x}} \{ \mathbf{w}^* \cdot \mathbf{x} \}$
  Let **w** be hypothesis plane
- Consider: $$\frac{(w \cdot w^*)}{|w|}$$
- On each mistake, add **x** to **w** $$\mathbf{w} = \Sigma_{\{x \mid \mathbf{x} \cdot \mathbf{w} < 0\}}\ \mathbf{x}$$

Proof (con't)

$$\boxed{\frac{(w \cdot w^*)}{|w|}}$$

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

$\mathbf{w} = \sum_{\{x \mid \mathbf{x} \cdot \mathbf{w} < 0\}} \mathbf{x}$

If w is mistake…

Numerator increases by $x \cdot w^* \geq \Delta$

(denominator)^2 becomes

$(w + x)^2 = w^2 + x^2 + 2(w \cdot x) < w^2 + 1$

as $w \cdot x < 0$

As initially $w = \langle 0, \ldots, 0 \rangle$.

After m mistakes,

numerator is $\geq m \times \Delta$

(denominator)^2 is $\leq 0 + \underbrace{1 + \ldots + 1}_{m} = m$

so denominator $\leq \sqrt{m}$

- As $(w \cdot w^*)/|w| =$ cos(angle between w and w^*)

  it must be ≤ 1, so

  numerator $\leq$ denominator

$$\Rightarrow \quad \Delta * m \leq \sqrt{m} \quad \Rightarrow \quad m \leq \frac{1}{\Delta^2}$$

#1b: Perceptron Training Rule

- For each labeled instance $[\mathbf{x}, y]$
  $\mathrm{Err}([\mathbf{x}, y]) = y - o_{\mathbf{w}}(\mathbf{x}) \in \{-1, 0, +1\}$
  - If $\mathrm{Err}([\mathbf{x}, y]) = 0$ Correct! … Do nothing!
    $\Delta w = 0 \equiv \mathrm{Err}([\mathbf{x}, y]) \cdot \mathbf{x}$
  - If $\mathrm{Err}([\mathbf{x}, y]) = +1$ Mistake on positive! Increment by +x
    $\Delta w = +x \equiv \mathrm{Err}([\mathbf{x}, y]) \cdot \mathbf{x}$
  - If $\mathrm{Err}([\mathbf{x}, y]) = -1$ Mistake on negative! Increment by -x
    $\Delta w = -x \equiv \mathrm{Err}([\mathbf{x}, y]) \cdot \mathbf{x}$

In all cases... $\Delta w^{(i)} = \mathrm{Err}([\mathbf{x}^{(i)}, y^{(i)}]) \cdot \mathbf{x}^{(i)} = [y^{(i)} - o_{\mathbf{w}}(\mathbf{x}^{(i)})] \cdot \mathbf{x}^{(i)}$

- Batch Mode: do ALL updates at once!

$$\boxed{\begin{aligned} \Delta w_j &= \textstyle\sum_i \Delta w_j^{(i)} \\ &= \textstyle\sum_i x^{(i)}_j \left(y^{(i)} - o_{\mathbf{w}}(\mathbf{x}^{(i)}) \right) \\ w_j &\mathrel{+}= \eta\, \Delta \mathbf{w}_j \end{aligned}}$$

η is **learning rate** (small pos "constant" $\ldots \approx 0.05?$)

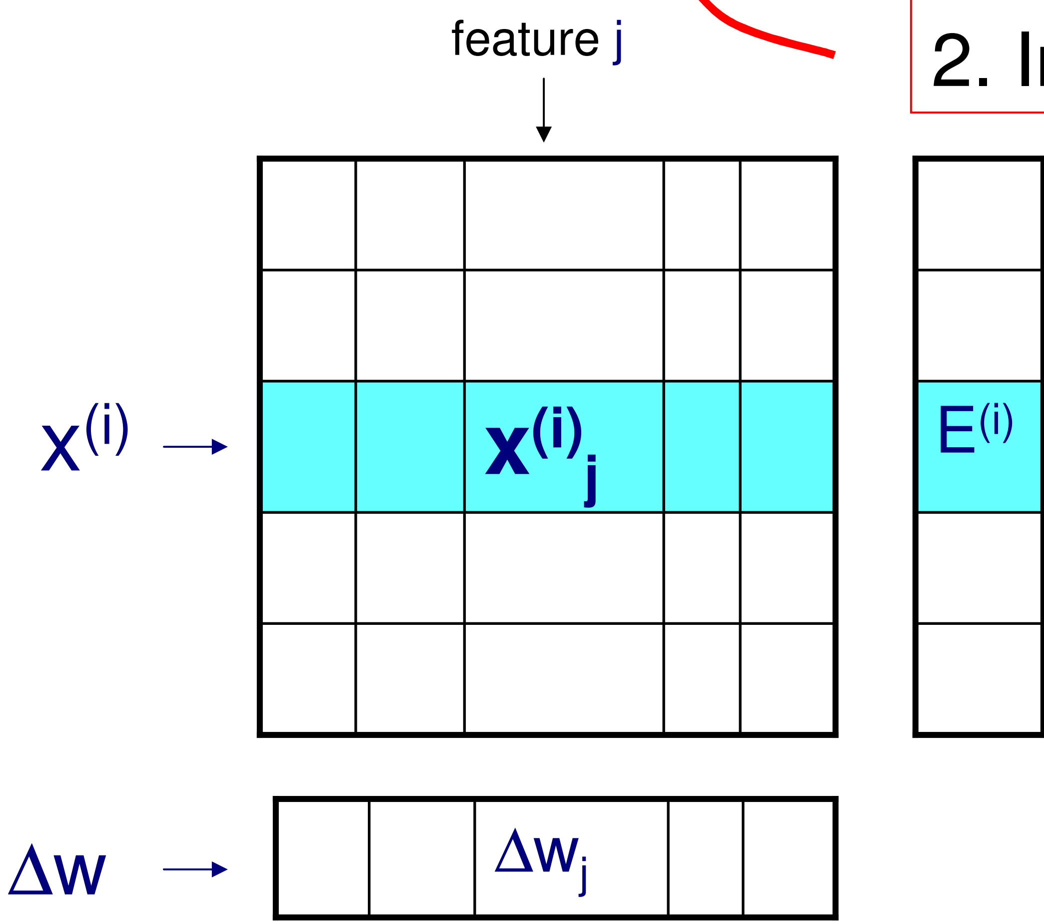
0. New $\mathbf{w}$

   $\Delta\mathbf{w} := 0$

1. For each row i, compute
   a. $E^{(i)} := y^{(i)} - o_{\mathbf{w}}(\mathbf{x}^{(i)})$
   b. $\Delta\mathbf{w} \mathrel{+}= E^{(i)}\, \mathbf{x}^{(i)}$

      $[\ \ldots\ \Delta w_j \mathrel{+}= E^{(i)}\, x^{(i)}_j \ldots\ ]$

2. Increment $\mathbf{w} \mathrel{+}= \eta\, \Delta\mathbf{w}$



Correctness

- Rule is intuitive: **Climbs in correct direction. . .**

- Thrm: Converges to correct answer, if . . .
  - training data is linearly separable
  - sufficiently small η

- Proof: Weight space has **EXACTLY 1 minimum**!

  (no non-global minima)

  $\Rightarrow$ with enough examples, finds correct function!

- Explains early popularity

- If η too large, may overshoot

  If η too small, takes too long
- So often $\eta = \eta(k)$ … which decays with # of iterations, k

#1c: Matrix Version?

- Task: Given $\{ \langle \mathbf{x}^i, y^i \rangle \}_i$
  - $y^i \in \{ -1, +1 \}$ is label

  Find $\{ w_i \}$ s.t.

$$
\begin{array}{rcl}
y^1 & = & w_0 + w_1\, x_1^1 + \cdots + w_n\, x_n^1 \\
y^2 & = & w_0 + w_1\, x_1^2 + \cdots + w_n\, x_n^2 \\
 & \vdots & \\
y^m & = & w_0 + w_1\, x_1^m + \cdots + w_n\, x_n^m
\end{array}
$$

- Linear Equalities $y = X\, \mathbf{w}$

$$
\begin{array}{rcl}
\mathbf{y} & = & [y^1, \ldots, y^m]^\top \\
\mathbf{X} & = & \begin{pmatrix} 1 & x_1^1 & \cdots & x_n^1 \\ 1 & x_1^2 & \cdots & x_n^2 \\ \vdots & \vdots & & \vdots \\ 1 & x_1^m & \cdots & x_n^m \end{pmatrix} \\
\mathbf{w} & = & [w_0, w_1, \ldots, w_n]^\top
\end{array}
$$

- Solution: $\mathbf{w} = X^{-1}\, \mathbf{y}$

Issues

Task: Given $\{ \langle \mathbf{x}^i, y^i \rangle \}$ $\quad y^i \in \{-1, +1\}$ is label
Find w_i s.t.

$$\begin{aligned} y^1 &= w_0 + w_1 x_1^1 + \cdots + w_n x_n^1 \\ y^2 &= w_0 + w_1 x_1^2 + \cdots + w_n x_n^2 \\ &\vdots \\ y^m &= w_0 + w_1 x_1^m + \cdots + w_n x_n^m \end{aligned}$$

1. Why restrict to only $y^i \in \{ -1, +1 \}$?
   - If from discrete set $y^i \in \{ 0, 1, \ldots, m \}$: General (non-binary) classification
   - If ARBITRARY $y^i \in \Re$: Regression

2. What if NO **w** works?

   ...X is singular; overconstrained ...

   Could try to minimize residual

$$\boxed{\text{NP-Hard!}}$$

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

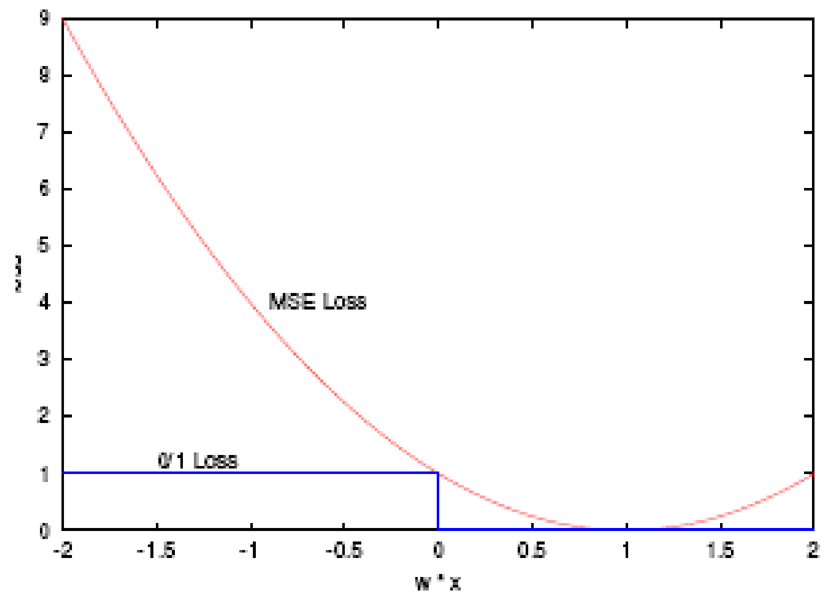
$$\| y - X\,\mathbf{w} \|_1 = \sum_i |\, y^{(i)} - \mathbf{w} \cdot \mathbf{x}^{(i)} \,|$$

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

$$\boxed{\text{Easy!}}$$

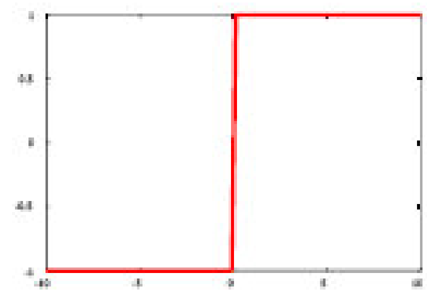
L_2 error vs 0/1-Loss

- "0/1 Loss function" not smooth, differentiable
- MSE error is smooth, differentiable… and is overbound...



Gradient Descent for Perceptron?

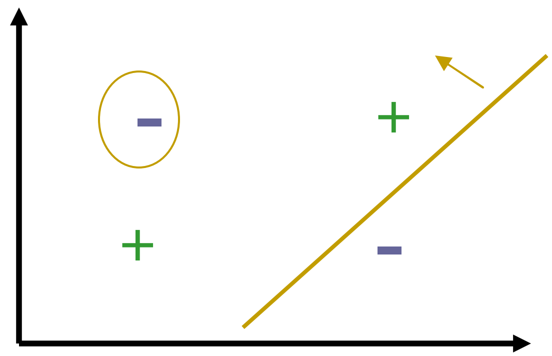
- Why not Gradient Descent
  for THRESHOLDed perceptron?
- Needs gradient (derivative), not



- Gradient Descent is General approach.
  Requires
  + continuously parameterized hypothesis
  + error must be differentiatable wrt parameters
  But. . .
  – can be slow (many iterations)
  – may only find LOCAL opt

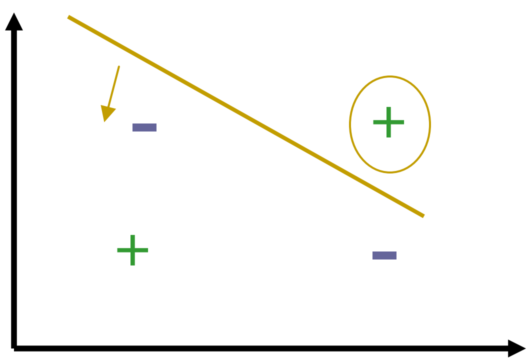
Linear Separators – Facts

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



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!

#1. LMS version of Classifier

- View as Regression
  - Find “best” linear mapping **w** from **X** to Y
    - $w^* = \text{argmin } Err_{LMS}^{(\mathbf{X},\mathbf{Y})}(w)$
    - $Err_{LMS}^{(\mathbf{X},\mathbf{Y})}(w) = \sum_i (y^{(i)} - \mathbf{w} \cdot \mathbf{x}^{(i)})^2$
  - Threshold: if $w^T x > 0.5$,
    return 1;
    else 0
- See Chapter 3…

General Idea

- Use a discriminant function $\delta_k(x)$ for each class k
  - *Eg,* $\delta_k(x)$ = P(G=k | X)
- Classification rule:

  Return k = $\text{argmax}_j\ \delta_j(x)$

- If each $\delta_j(x)$ is linear,

  decision boundaries are piecewise hyperplanes

Linear Classification using Linear Regression

- 2D Input space: $X = (X_1, X_2)$
  K-3 classes: $Y = (Y_1, Y_2, Y_3) \in \begin{cases} [1,0,0] \\ [0,1,0] \\ [0,0,1] \end{cases}$

- Training sample (N=5): $\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \\ 1 & x_{41} & x_{42} \\ 1 & x_{51} & x_{52} \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \\ y_{41} & y_{42} & y_{43} \\ y_{51} & y_{52} & y_{53} \end{bmatrix}$

- Regression output:

$$\hat{Y}((x_1, x_2)) = (1\ x_1\ \ x_2)(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} = (x^T\beta_1\ \ x^T\beta_2\ \ x^T\beta_3)$$

$$\hat{Y}_1((x_1\ x_2)) = (1\ x_1\ \ x_2)\beta_1$$
$$\hat{Y}_2((x_1\ x_2)) = (1\ x_1\ \ x_2)\beta_2$$
$$\hat{Y}_3((x_1\ x_2)) = (1\ x_1\ \ x_2)\beta_3$$

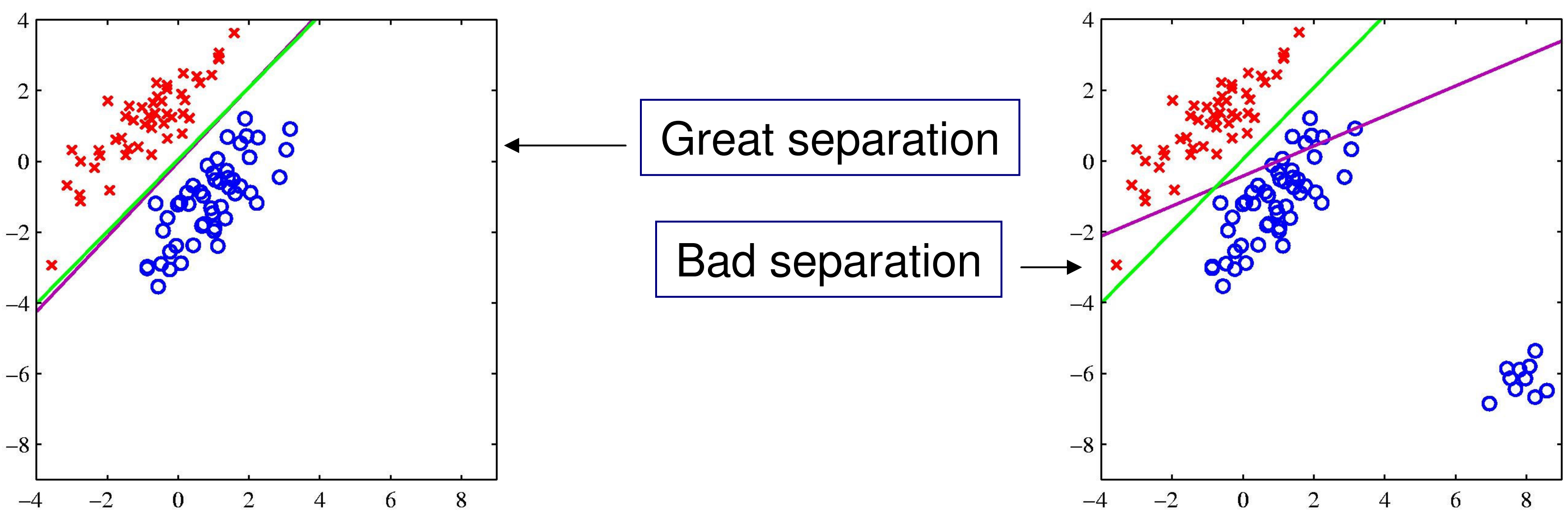
- Classification rule:

$$\hat{G}((x_1\ \ x_2)) = \arg\max_k \hat{Y}_k((x_1\ \ x_2))$$

Use Linear Regression for Classification?

- But … regression minimizes
  sum of squared errors on target function

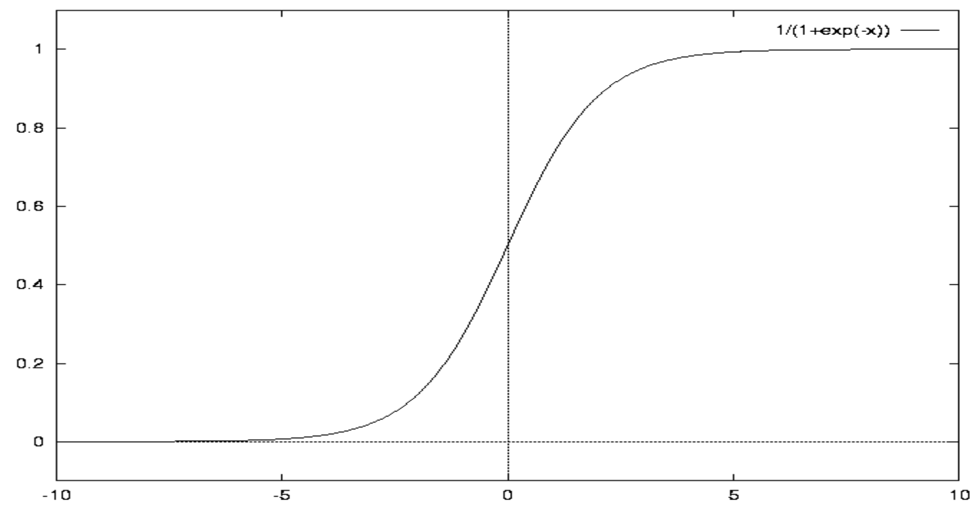
… which gives strong influence to outliers



#3: Logistic Regression

- Want to compute $P_{\mathbf{w}}(y=1|\ \mathbf{x})$
  ... based on parameters **w**
- But …
  - **w·x** has range $[-\infty, \infty]$
  - probability must be in range $\in [0; 1]$
- Need “squashing” function $[-\infty, \infty] \rightarrow [0, 1]$

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



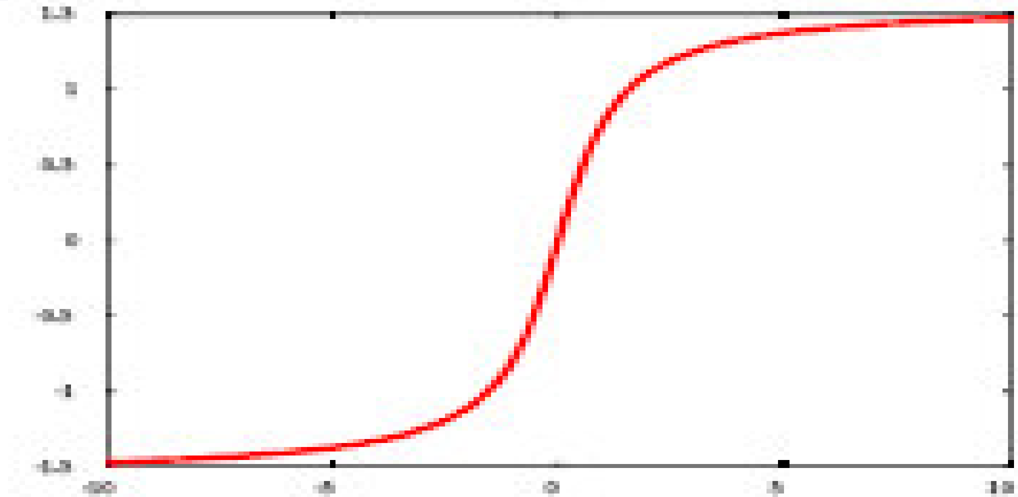
Alternative Derivation…

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

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

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

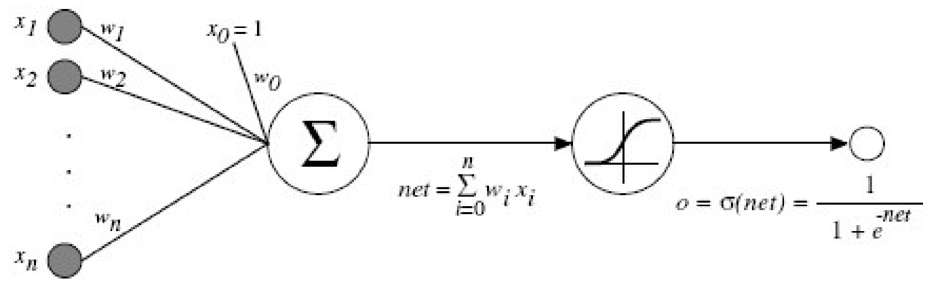
Sigmoid Unit



- Sigmoid Function: $\sigma(x) = \frac{1}{1+e^{-x}}$

- Useful properties:
  - $\sigma : \Re \rightarrow [0,1]$
  - $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\,(1-\sigma(x))$
  - If $x \approx 0$, then $\sigma(x) \approx x$

Logistic Regression (con't)



- Assume 2 classes:

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

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

- Log Odds:

$$\boxed{\log \frac{P_w(+y \mid x)}{P_w(-y \mid x)} = x \cdot w}$$

Linear

How to learn parameters $\mathbf{w}$?

- … depends on goal?
  - A: Minimize MSE?

    $\sum_i (y^{(i)} - o_{\mathbf{w}}(\mathbf{x}^{(i)}))^2$
  - B: Maximize likelihood?

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

MSError Gradient for Sigmoid Unit

- Error: $\sum_j (y^{(j)} - o_{\mathbf{w}}(\mathbf{x}^{(j)}))^2 = \sum_j E^{(j)}$

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

For single training instance

- **Input**: $\mathbf{x}^{(j)} = [x^{(j)}_1, \ldots, x^{(j)}_k]$
- **Computed Output**: $o^{(j)} = \sigma(\sum_i x^{(j)}_i \cdot w_i) = \sigma(z^{(j)})$
  - where $z^{(j)} = \sum_i x^{(j)}_i \cdot w_i$ using current $\{ w_i \}$
- **Correct output**: $y^{(j)}$

Stochastic Error Gradient (Ignore $^{(j)}$ superscript)

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

$$= (o-y)\left(\frac{\partial o}{\partial w_i}\right) = (o-y)\ \frac{\partial \sigma(z)}{\partial z}\ \frac{\partial z}{\partial w_i}$$

Derivative of Sigmoid

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

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

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

Updating LR Weights (MSE)

- $$\frac{\partial E}{\partial w_i} = (o-y)\,\frac{\partial \sigma(z)}{\partial z}\,\frac{\partial z}{\partial w_i}$$

- Using:

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)\,(1-\sigma(z)) = o(1-o)$$
$$\frac{\partial z}{\partial w_i} = \frac{\partial(\sum_i w_i \cdot x_i)}{\partial w_i} = x_i$$

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

$$\Rightarrow \boxed{\frac{\partial E^{(j)}}{\partial w_i} = (o^{(j)} - y^{(j)})\,o^{(j)}\,(1-o^{(j)})\,x_i^{(j)}}$$

Note: As already computed $o^{(j)} = \sigma(z^{(j)})$ to get answer,
trivial to compute $\sigma'(z^{(j)}) = \sigma(z^{(j)})(1-\sigma(z^{(j)}))$

- Update $w_i \mathrel{+}= \Delta w_i$ where $$\boxed{\Delta w_i = \eta \cdot \frac{\partial E^{(j)}}{\partial w_i}}$$

(LMS)

0. New **w**

   $\Delta \mathbf{w} = 0$

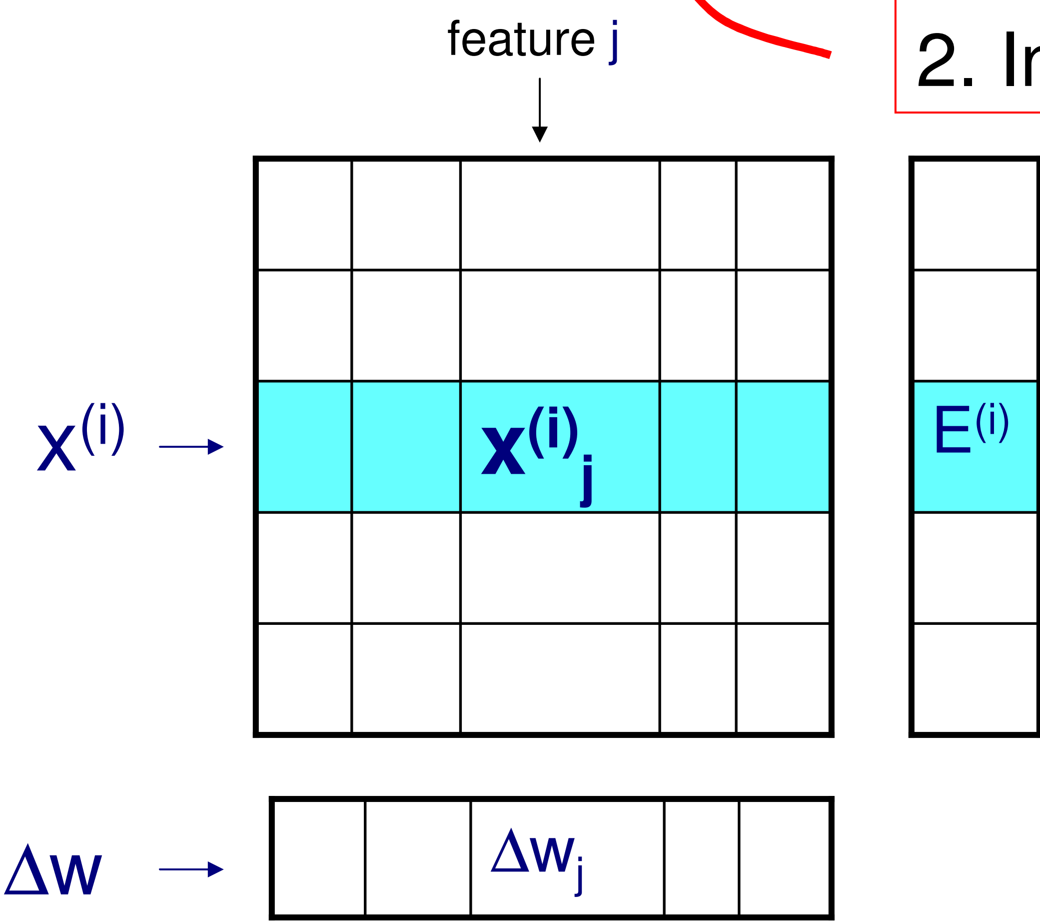
1. For each row i, compute

   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 $w \mathrel{+}= \eta \Delta \mathbf{w}$



B: Or... Learn Conditional Probability

- As fitting *probability distribution*,
  better to return probability distribution (≈ **w**)
  that is most likely, given training data, S

$$
\begin{aligned}
\text{Goal: } \mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmax}} P(\mathbf{w} \mid S) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \frac{P(S \mid \mathbf{w}) P(\mathbf{w})}{P(S)} && \text{Bayes Rules} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} P(S \mid \mathbf{w}) P(\mathbf{w}) && \text{As } P(S) \text{ does not depend on } \mathbf{w} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} P(S \mid \mathbf{w}) && \text{As } P(\mathbf{w}) \text{ is uniform} \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \log P(S \mid \mathbf{w}) && \text{As log is monotonic}
\end{aligned}
$$

ML Estimation

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

  $L(\mathbf{w}) = \log P(S \mid \mathbf{w})$
- $\mathbf{w}^* = \mathrm{argmax}_{\mathbf{w}} L(\mathbf{w})$

  is "maximum likelihood estimator" (MLE)

Computing the Likelihood

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

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

  $= \sum_i \log P_{\mathbf{w}}(y^{(i)} \mid \mathbf{x}^{(i)}) + \sum_i \log P_{\mathbf{w}}(\mathbf{x}^{(i)})$
- Here $P_{\mathbf{w}}(\mathbf{x}^{(i)}) = 1/n$ …
  not dependent on **w**, over empirical sample S
- $\mathbf{w}^* = \text{argmax}_{\mathbf{w}} \sum_i \log P_{\mathbf{w}}(y^{(i)} \mid \mathbf{x}^{(i)})$

Fit Logistic Regression… by Gradient Ascent

- Want $\mathbf{w}^* = \text{argmax}_w\ J(\mathbf{w})$
  - $J(w) = \sum_i r(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{w})$
  - For $y \in \{0, 1\}$

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

- So climb along…

$$\boxed{\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_i \frac{\partial r(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{w})}{\partial w_j}}$$

Gradient Descent …

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

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

$$\frac{\partial p_1}{\partial\, w_j} = \frac{\partial\, P_w(y=1\,|\,x)}{\partial\, w_j} = \frac{\partial}{\partial\, w_j}\big(\sigma(x\cdot w)\big)$$

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

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

$$= \sum_i (\, y^{(i)} - P_w(\, y = 1\,|\, x))\cdot x_j^{(i)}$$

Gradient Ascent for Logistic Regression (MLE)

Given: training examples $\langle \mathbf{x}^{(i)}, y^{(i)} \rangle$, $i = 1..N$
Set initial weight vector $\mathbf{w} = \langle 0, 0, 0, 0, \ldots, 0 \rangle$
Repeat until convergence
  Let gradient vector $\Delta \mathrm{w} = \langle 0, 0, 0, 0, \ldots, 0 \rangle$
  For $i = 1$ to N do
    $p_1^{(i)} = 1/(1 + \exp[\mathbf{w} \cdot \mathbf{x}^{(i)}])$
    $\text{error}_i = y^{(i)} - p_1^{(i)}$
    For $j = 1$ to n **do**
      $\Delta \mathrm{w}_j \mathrel{+}= \text{error}_i \cdot x_{ij}$
  $\mathbf{w} \mathrel{+}= \eta \, \Delta \mathrm{w}$ *% step in direction of increasing gradient*

Comments on MLE Algorithm

- This is BATCH;
  $\exists$ 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)

Use Logistic Regression for Classification

- Return YES iff

$$
\begin{aligned}
P(y=1 \mid x) &> P(y=0 \mid x) \\
\frac{P(y=1 \mid x)}{P(y=0 \mid x)} &> 1 \\
\ln \frac{P(y=1 \mid x)}{P(y=0 \mid x)} &> 0 \\
\ln \frac{1 \qquad /(1+\exp(-w \cdot x))}{\exp(-w \cdot x)/(1+\exp(-w \cdot x))} &> 0 \\
\ln \frac{1}{\exp(-w \cdot x)} = w \cdot x &> 0
\end{aligned}
$$

$\boxed{\text{Logistic Regression learns a LTU!}}$

Logistic Regression for K > 2 Classes

- To handle $K > 2$ classes
  - Let class K be "reference"
  - Represent each other class $k \neq K$ as logistic function of odds of class k versus class K:

$$\log \frac{P(y=1 \mid \mathrm{x})}{P(y=K \mid \mathrm{x})} = \mathrm{w}_1 \cdot \mathrm{x}$$

$$\log \frac{P(y=2 \mid \mathrm{x})}{P(y=K \mid \mathrm{x})} = \mathrm{w}_2 \cdot \mathrm{x}$$

$$\vdots$$

$$\log \frac{P(y=K-1 \mid \mathrm{x})}{P(y=K \mid \mathrm{x})} = \mathrm{w}_{K-1} \cdot \mathrm{x}$$

Note: k-1 different $\mathbf{w}_i$ weights, … each of dimension $|\mathbf{x}|$

- Apply gradient ascent to learn *all* w_k weight vectors, in parallel.
- Conditional probabilities:

$$P(y=k \mid \mathrm{x}) = \frac{\exp(\mathrm{w}_k \cdot \mathrm{x})}{1 + \sum_{\ell=1}^{K-1} \exp(\mathrm{w}_\ell \cdot \mathrm{x})}$$

and

$$P(y=K \mid \mathrm{x}) = \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(\mathrm{w}_\ell \cdot \mathrm{x})}$$

Learning LR Weights

Task: Given data $\langle\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\rangle$,

find w in $p_{\mathbf{w}}(y|\mathbf{x}) = \begin{cases} \dfrac{1}{1+\exp(-w \cdot x)} & \text{if } y = 1 \\ \dfrac{\exp(-w \cdot x)}{1+\exp(-w \cdot x)} & \text{if } y = 0 \end{cases}$

s.t. $p_{\mathbf{w}}(y^{(i)}|\mathbf{x}^{(i)}) > \frac{1}{2}$ iff $y^{(i)} = 1$

Approach 1: MSE – "Neural nets"

Minimize $\sum_i (o^{(i)} - y^{(i)})^2$

Gradient: $\boxed{\Delta \mathbf{w}^{(i)}_j = (o^{(i)} - y^{(i)})\, o^{(i)}\, (1 - o^{(i)})}$

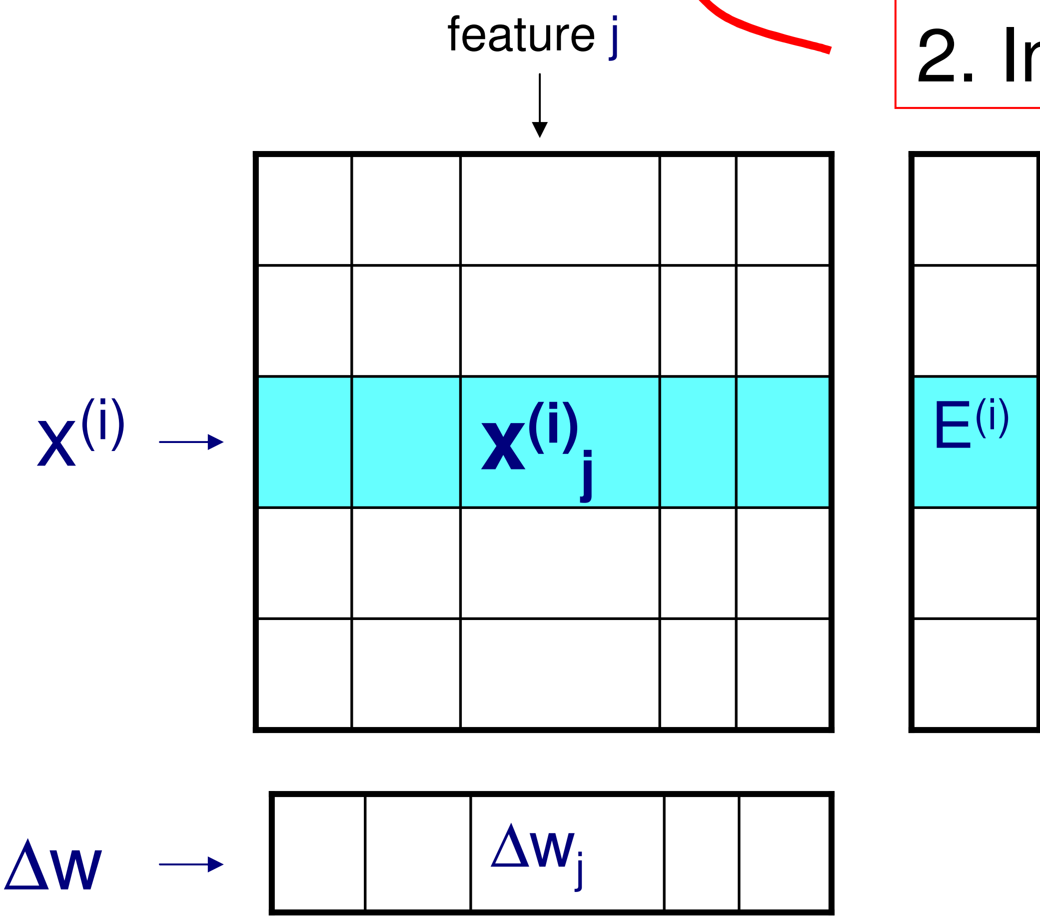
Approach 2: MLE – "Logistic Regression"

Maximize $\sum_i p_{\mathbf{w}}(y|\mathbf{x})$

Gradient: $\boxed{\Delta \mathbf{w}^{(i)}_j = (y^{(i)} - p(1|x^{(i)}))\, x^{(i)}_j}$

(MaxProb)

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



Logistic Regression Computation…

$$l(\beta) = \sum_{i=1}^{N} \{\log \Pr(G = y_i \mid X = x_i)\}$$

$$= \sum_{i=1}^{N} y_i \log(\Pr(G = 1 \mid X = x_i)) + (1 - y_i)\log(\Pr(G = 0 \mid X = x_i))$$

$$= \sum_{i=1}^{N} (y_i \beta^T x_i + (1 - y_i)\log \frac{1}{1 + \exp(\beta^T x_i)})$$

$$= \sum_{i=1}^{N} (y_i \beta^T x_i - (1 - y_i)\log(1 + \exp(\beta^T x_i)))$$

$$\boxed{\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} \left(y_i - \frac{\exp(\beta^T x)}{1 + \exp(\beta^T x)} \right) x_i = 0}$$

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

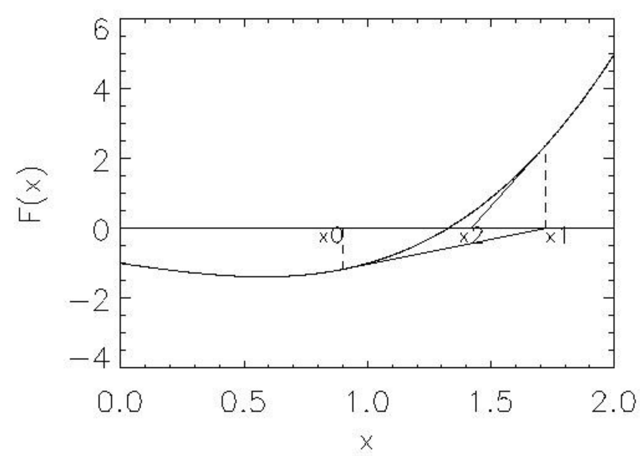
$$\beta^{new} = \beta^{old} - [\text{Jacobian}(\frac{\partial l(\beta^{old})}{\partial \beta})]^{-1} \frac{\partial l(\beta^{old})}{\partial \beta}$$

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, $f(x_{n+1}) \approx 0$

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

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:

$$f_j(x + \Delta x) = f_j(x) + \sum_{k=1}^{N} \frac{\partial f_j}{\partial x_k} \Delta x_k, \qquad j = 1, \ldots, N$$

- N-R:

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

Jacobian matrix

Newton-Raphson : Example

- Solve

$$\boxed{\begin{aligned} f_1(x_1, x_2) &= x_1^2 - \cos(x_2) &= 0 \\ f_2(x_1, x_2) &= \sin(x_1) + x_1^2 + x_2^3 &= 0 \end{aligned}}$$

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

Maximum Likelihood Parameter Estimation

- Find the unknown parameters mean & standard deviation of a Gaussian pdf,

$$\boxed{p(x;\mu,\sigma)=\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(x-\mu)^2}{2\sigma^2})}$$

given N independent samples, $\{x_1,\ldots,x_N\}$

- Estimate the parameters that maximize the likelihood function $L(\mu,\sigma)=\prod_{i=1}^{N}\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{(x_i-\mu)^2}{2\sigma^2})$

$$(\hat{\mu},\hat{\sigma})=\arg\max_{\mu,\sigma} L(\mu,\sigma)$$

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, seek **w** s.t. probability distribution $P_{\mathbf{w}}(y | x)$ is most likely given data.)

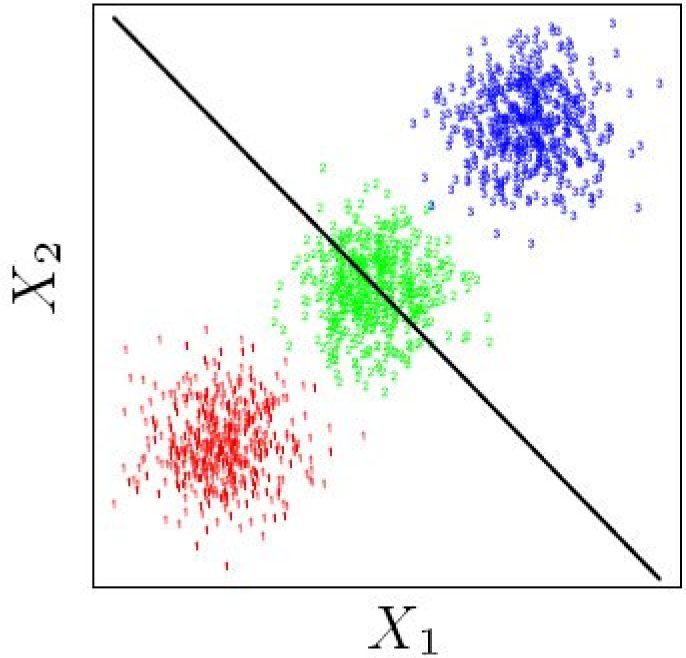
- **Eager**: Classifier constructed from training examples, which can then be discarded.

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

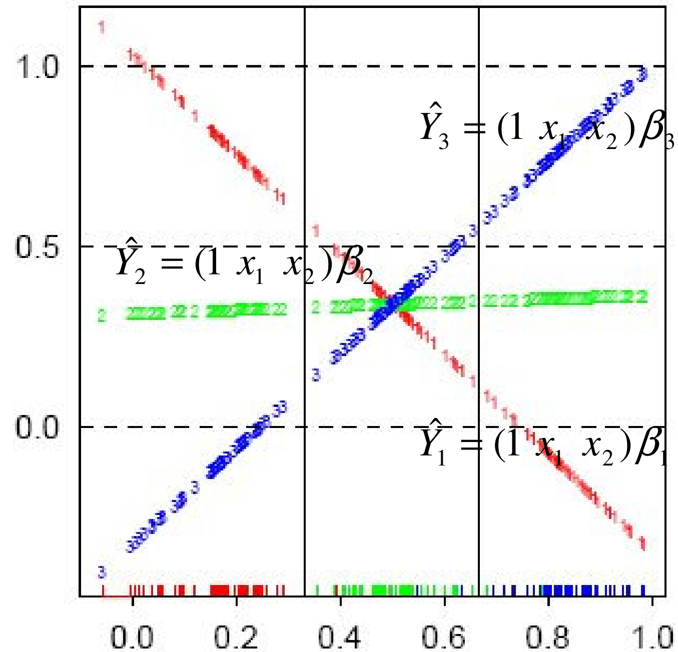
Masking of Some Class

Linear regression of the indicator matrix can lead to masking

2D input space and three classes



Masking



Viewing direction

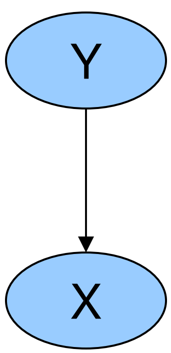
LDA can avoid this masking

#4: Linear Discriminant Analysis

- LDA learns joint distribution $P(y, \mathbf{x})$
  - As $P(y, x) \neq P(y \mid x)$;
    optimizing $P(y, x) \neq$ optimizing $P(y \mid x)$
- "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, **x**) = P(y) P(**x** | 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(**x** | y) is multivariate normal, with mean μ_k and covariance Σ

$$P(\mathrm{x} \mid y = k) =$$

$$\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\mathrm{x} - \mu_k)^\top \Sigma^{-1} (\mathrm{x} - \mu_k)\right]$$

Estimating LDA Model

- Linear discriminant analysis assumes form

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

- μ_y is mean for examples belonging to class y; covariance matrix Σ is shared by all classes !
- Can estimate LDA directly:

  m_k = #training examples in class $y = k$

  - Estimate of $P(y = k)$: $p_k = m_k / m$

$$\boxed{\hat{\mu}_k = \frac{1}{m}\sum_{\{i: y_i = k\}} x_i} \qquad \boxed{\hat{\Sigma} = \frac{1}{m}\sum_i (x_i - \hat{\mu}_{y_i})(x_i - \hat{\mu}_{y_i})^T}$$

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	–

Note: do NOT pre-pend x_0=1!

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

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

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

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

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

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

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 $\mathbf{z} := \mathbf{x} - \mu_{y(\mathbf{x})}$

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

. . .

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

… $z^{(7)}$:= …

  - 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 $\hat{\Sigma} = \frac{1}{m} \sum_i \mathbf{z}^{(i)} \mathbf{z}^{(i)\top}$

Classifying, Using LDA

- How to classify new instance, given estimates

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

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** = $[5, 14, 6]^T$?

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

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

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

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

LDA learns an LTU

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

$$\log \frac{P(y=1 \mid \mathrm{x})}{P(y=0 \mid \mathrm{x})} > 0 \quad \text{iff} \quad \log \frac{P(y=1, \mathrm{x})}{P(y=0, \mathrm{x})} > 0$$

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

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

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

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

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

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

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

LDA Learns an LTU (3)

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

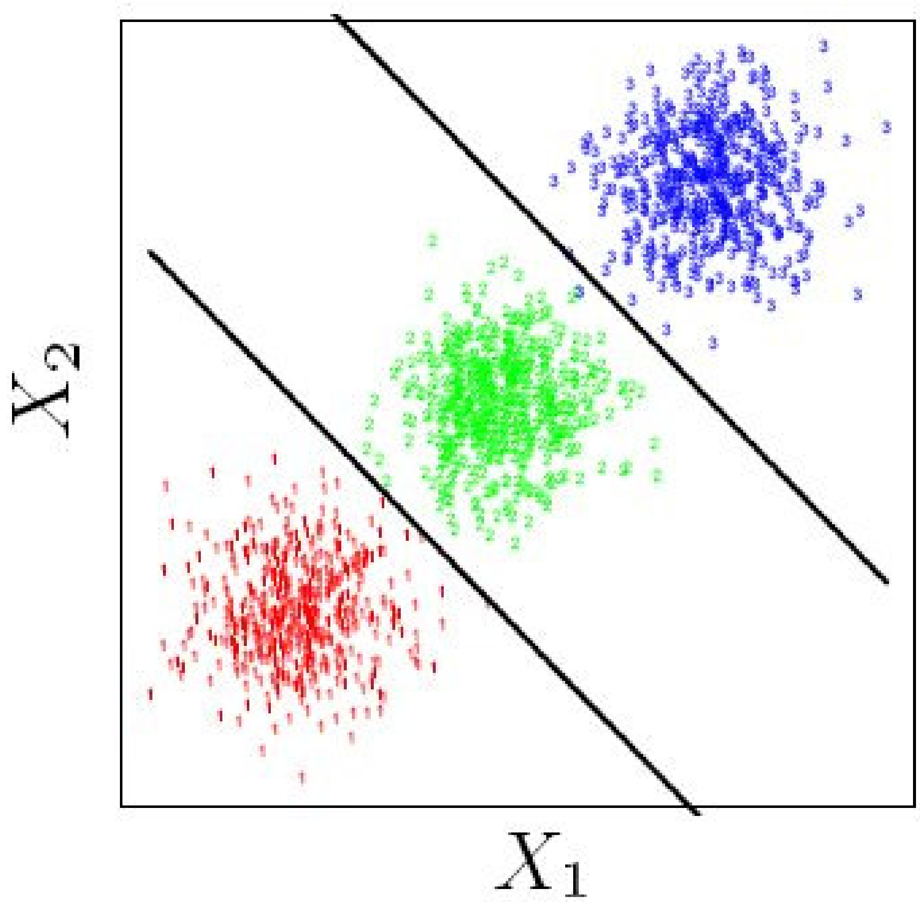
- So let…

$$\begin{aligned} \mathbf{w} &= \Sigma^{-1}(\mu_1 - \mu_0) \\ c &= \ln \frac{P(y=1)}{P(y=0)} + \frac{1}{2}\mu_0^\top \Sigma^{-1}\mu_0 - \frac{1}{2}\mu_1^\top \Sigma^{-1}\mu_1 \end{aligned}$$

- Classify $\hat{y} = 1$ iff $\mathbf{w} \cdot \mathbf{x} + c > 0$

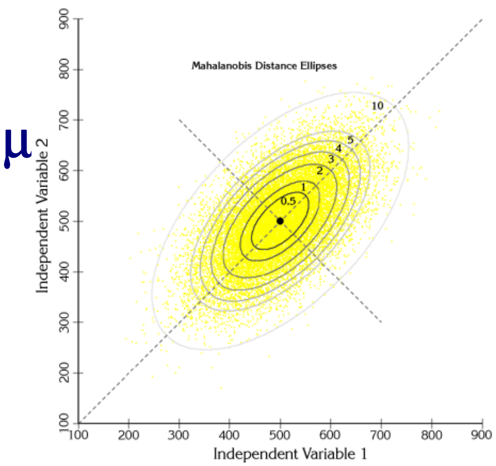
  LTU!!

LDA: Example



LDA was able to avoid masking here

View LDA wrt Mahalanobis Distance



- Squared Mahalanobis distance between $\mathbf{x}$ 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 $\mathbf{x}$ as 0 if

$$D_M^2(\mathbf{x}, \mu_\mathbf{0}) < D_M^2(\mathbf{x}, \mu_\mathbf{1})$$

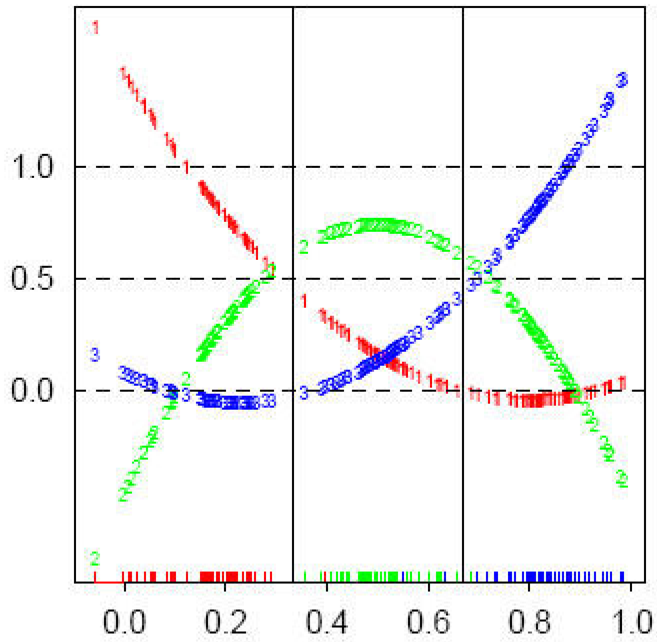
- $\log P(\mathbf{x} \mid y = k) \approx \log \pi_k - ½\, D_M^2(\mathbf{x}, \mu_\mathbf{k})$

Generalizations of LDA

- **General Gaussian Classifier: QDA**
  Allow each class k to have its own Σ_k
  $\Rightarrow$ Classifier $\equiv$ ***quadratic*** threshold unit (not LTU)
- **Naïve Gaussian Classifier**
  Allow each class *k* to have its own Σ_k
  but require each Σ_k be diagonal.
  $\Rightarrow$ within each class,
  any pair of features x_i and x_j are independent
  - Classifier is still quadratic threshold unit
    but with a restricted form
- **Most "discriminating" Low Dimensional Projection**
  - Fisher's Linear Discriminant

QDA and Masking

Better than Linear Regression in terms of handling masking:



Usually computationally more expensive than LDA

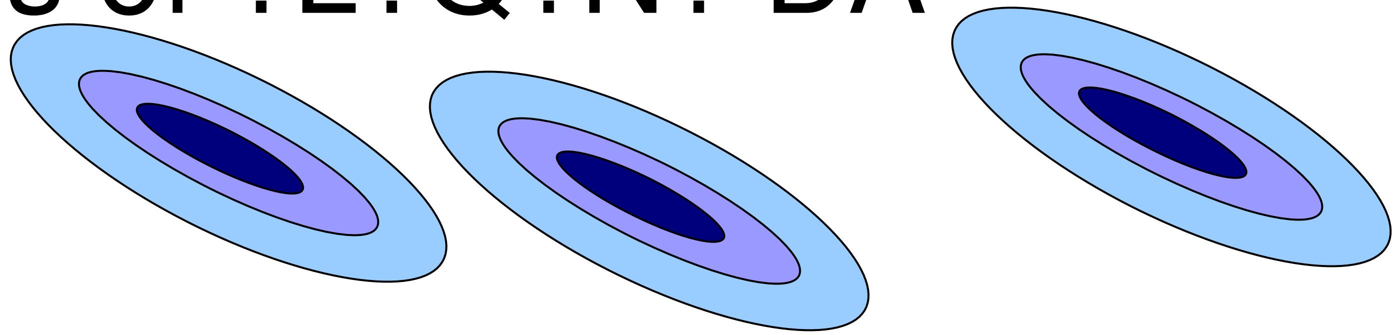
Variants of LDA

- Covariance matrix Σ
- n features; k classes

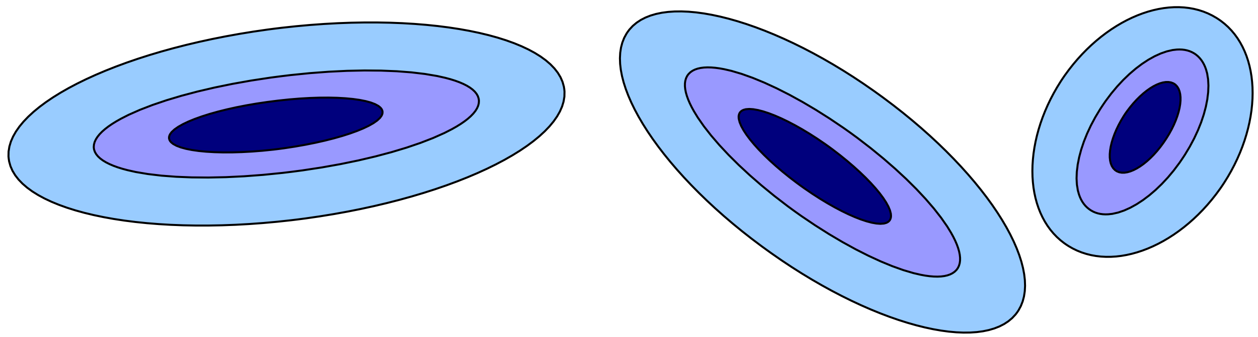
Name	Same for all classes?	Diagonal	#param's
	+	+	k
LDA	+	–	n^2
Naïve Gaussian Classifier	–	+	$k\ n$
General Gaussian Classifier	–	–	$k\ n^2$

Versions of ?L?Q?N? DA

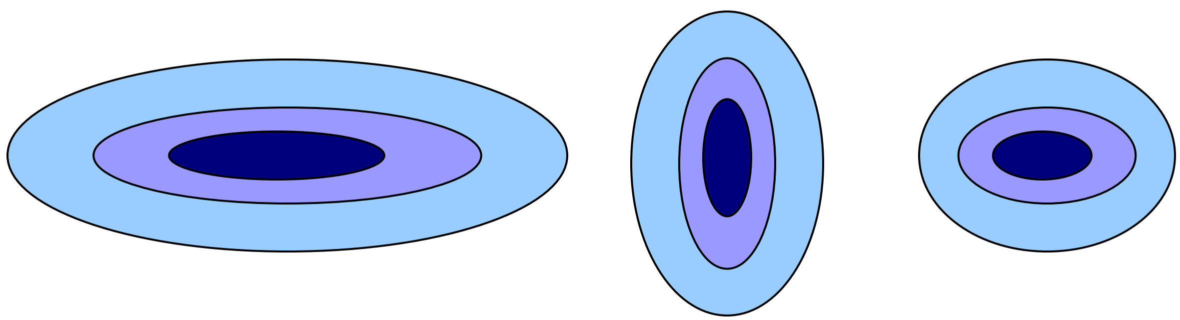
- LDA



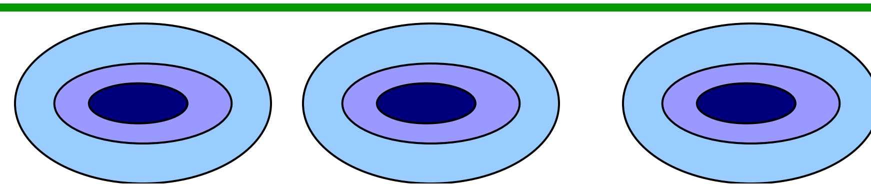
- Quadratic



- Naïve



- SuperSimple



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 $O(n^3)$
- **Eager**:

  Classifier constructed from training examples, which can then be discarded.
- **Batch**: Only a batch algorithm.

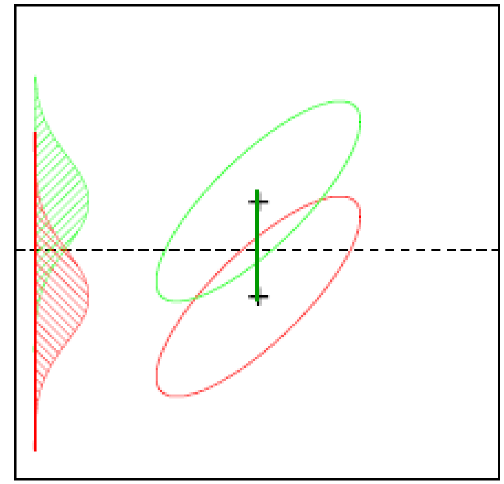
  An online LDA alg requires online alg for incrementally updating Σ^{-1}

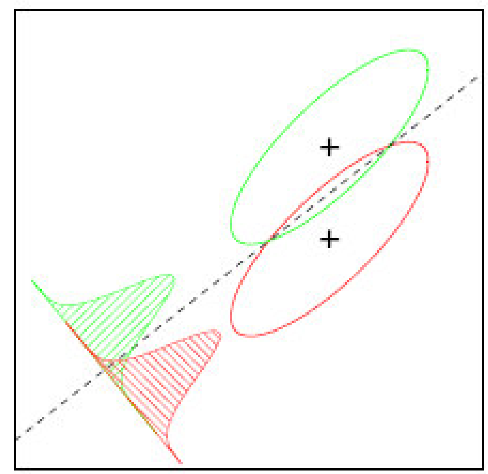
  [Easy if Σ^{-1} is diagonal. . .]

Fisher's Linear Discriminant

- LDA
  - Finds K–1 dim hyperplane
    (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





Fisher Linear Discriminant

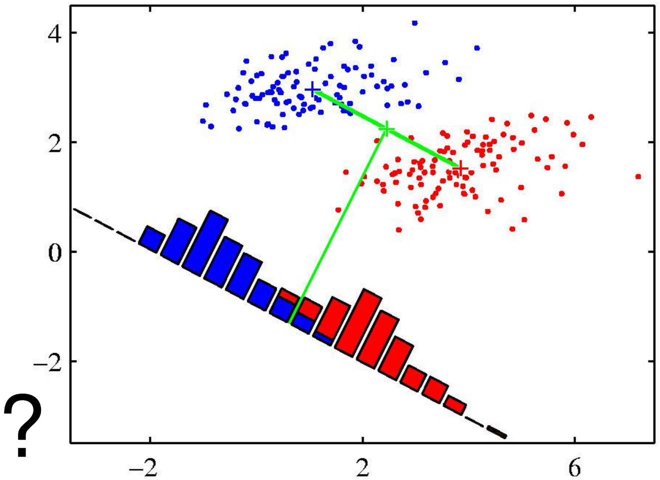
- Recall any vector **w** projects $\Re^n \rightarrow \Re$
- Goal: Want **w** that "separates" classes
  - Each $\mathbf{w} \cdot \mathbf{x}^+$ far from each $\mathbf{w} \cdot \mathbf{x}^-$

- Using $\mathbf{m}_+ = \frac{\sum_i y^{(i)} \cdot \mathbf{x}^{(i)}}{\sum_i y^{(i)}}$ $\quad \mathbf{m}_- = \frac{\sum_i (1-y^{(i)}) \cdot \mathbf{x}^{(i)}}{\sum_i (1-y^{(i)})}$

Mean of x's projections:

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

$$\mu_- = \frac{\sum_i (1-y^{(i)}) \mathbf{w}^\top \cdot \mathbf{x}^{(i)}}{\sum_i (1-y^{(i)})} = \mathbf{w}^\top \cdot \mathbf{m}_-$$



- Perhaps project onto $\mathbf{m}_+ - \mathbf{m}_-$?
- Still overlap… why?

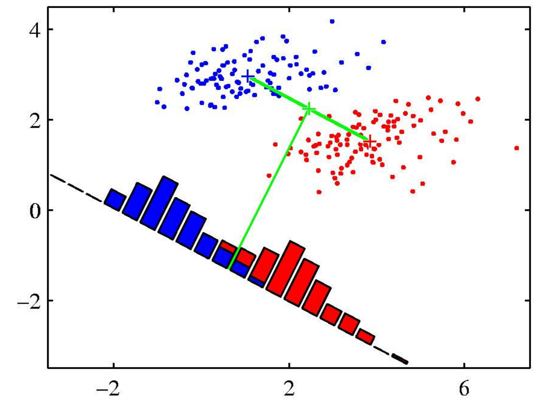
Fisher Linear Discriminant

- Using $\mathbf{m}_+ = \frac{\sum_i y^{(i)} \cdot \mathbf{x}^{(\mathrm{i})}}{\sum_i y^{(i)}}$ $\quad \mathbf{m}_- = \frac{\sum_i (1-y^{(i)}) \cdot \mathbf{x}^{(\mathrm{i})}}{\sum_i (1-y^{(i)})}$

  Mean of $\mathbf{x}$'s projections:

$$\mu_+ = \frac{\sum_i y^{(i)} \mathbf{w}^\top \cdot \mathbf{x}^{(\mathrm{i})}}{\sum_i y^{(i)}} = \mathbf{w}^\top \cdot \mathbf{m}_+$$

$$\mu_- = \frac{\sum_i (1-y^{(i)}) \mathbf{w}^\top \cdot \mathbf{x}^{(\mathrm{i})}}{\sum_i (1-y^{(i)})} = \mathbf{w}^\top \cdot \mathbf{m}_-$$



- Problem with $\mathbf{m}_+ - \mathbf{m}_-$:
- Does not consider "scatter" within class
- Goal: Want $\mathbf{w}$ that "separates" classes
  - Each $\mathbf{w} \cdot \mathbf{x}^+$ far from each $\mathbf{w} \cdot \mathbf{x}^-$
  - Positive $\mathbf{x}^+$'s: $\mathbf{w} \cdot \mathbf{x}^+$ close to each other
  - Negative $\mathbf{x}^-$'s: $\mathbf{w} \cdot \mathbf{x}^-$ close to each other

- "scatter" of +instance; –instance
  - $\mathbf{s}_+^2 = \sum_i y^{(i)} \left(\mathbf{w} \cdot \mathbf{x}^{(i)} - \mathbf{m}_+\right)^2$
  - $\mathbf{s}_-^2 = \sum_i (1 - y^{(i)}) \left(\mathbf{w} \cdot \mathbf{x}^{(i)} - \mathbf{m}_-\right)^2$

Fisher Linear Discriminant

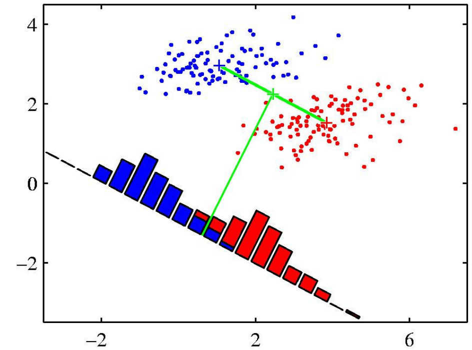
- Recall any vector **w** projects $\Re^n \rightarrow \Re$
- Goal: Want **w** that "separates" classes
  - Positive $\mathbf{x}^+$'s: $\mathbf{w} \cdot \mathbf{x}^+$ close to each other
  - Negative $\mathbf{x}^-$'s: $\mathbf{w} \cdot \mathbf{x}^-$ close to each other
  - Each $\mathbf{w} \cdot \mathbf{x}^+$ far from each $\mathbf{w} \cdot \mathbf{x}^-$

- Using $\mathbf{m}_+ = \frac{\sum_i y^{(i)} \cdot \mathbf{x}^{(i)}}{\sum_i y^{(i)}}$ $\quad \mathbf{m}_- = \frac{\sum_i (1-y^{(i)}) \cdot \mathbf{x}^{(i)}}{\sum_i (1-y^{(i)})}$

Mean of x's projections:

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

$$\mu_- = \frac{\sum_i (1-y^{(i)}) \mathbf{w}^\top \cdot \mathbf{x}^{(i)}}{\sum_i (1-y^{(i)})} = \mathbf{w}^\top \cdot \mathbf{m}_-$$



- "scatter" of +instance; –instance
  - $\mathbf{s}_+^2 = \sum_i y^{(i)} \left(\mathbf{w} \cdot \mathbf{x}^{(i)} - \mathbf{m}_+\right)^2$
  - $\mathbf{s}_-^2 = \sum_i (1 - y^{(i)}) \left(\mathbf{w} \cdot \mathbf{x}^{(i)} - \mathbf{m}_+\right)^2$

FLD, con't

- Separate means $\mathbf{m}_-$ and $\mathbf{m}_+$

  $\Rightarrow$ maximize $(\mathbf{m}_- - \mathbf{m}_+)^2$
- Minimize each spread $\mathbf{s}_+^2$, $\mathbf{s}_-^2$

  $\Rightarrow$ minimize $(\mathbf{s}_+^2 + \mathbf{s}_-^2)$
- Objective function: maximize $\boxed{J_S(w) = \dfrac{(\mu_+ - \mu_-)^2}{(s_+^2 + s_-^2)}}$

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

$\quad = \mathbf{w}^T (\mathbf{m}_+ - \mathbf{m}_-)(\mathbf{m}_+ - \mathbf{m}_-)^T \mathbf{w} \;=\; \mathbf{w}^T S_B\, \mathbf{w}$

"between-class scatter" $\boxed{S_B = (\mathbf{m}_+ - \mathbf{m}_-)\,(\mathbf{m}_+ - \mathbf{m}_-)^T}$

FLD, III

$$J_S(w) = \frac{(\mu_+ - \mu_-)^2}{(s_+^2 + s_-^2)}$$

- $\mathbf{s_+}^2 = \sum_i y^{(i)} \left(\mathbf{w} \cdot \mathbf{x}^{(i)} - \mathbf{m}_+\right)^2$
  $= \sum_i \mathbf{w}^T y^{(i)} \left(\mathbf{x}^{(i)} - \mathbf{m}_+\right) \left(\mathbf{x}^{(i)} - \mathbf{m}_+\right)^T \mathbf{w}$
  $= \mathbf{w}^T S_+ \mathbf{w}$

$$S_+ = \sum_i y^{(i)} \left(\mathbf{x}^{(i)} - \mathbf{m}_+\right) \left(\mathbf{x}^{(i)} - \mathbf{m}_+\right)^T$$

… "within-class scatter matrix" for +

$$S_- = \sum_i \left(1 - y^{(i)}\right) \left(\mathbf{x}^{(i)} - \mathbf{m}_-\right) \left(\mathbf{x}^{(i)} - \mathbf{m}_-\right)^T$$

… "within-class scatter matrix" for –

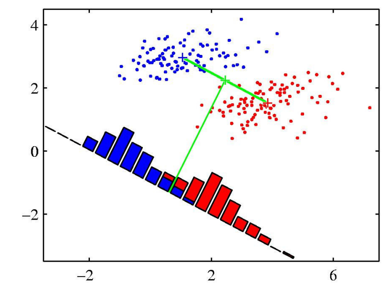
- $S_W = S_+ + S_-$ so $\mathbf{s_+}^2 + \mathbf{s_-}^2 = \mathbf{w}^T S_W \mathbf{w}$

FLD, IV

$$J_S(\mathbf{w}) = \frac{(\mu_+ - \mu_-)^2}{(s_+^2 + s_-^2)} = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}$$

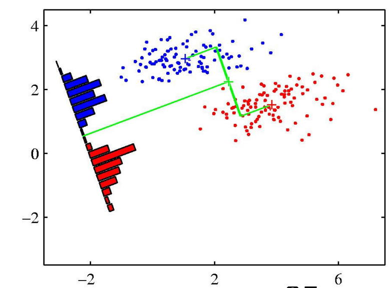
- Minimizing $J_S(\mathbf{w})$ …
  $\mathbf{w}^* = \text{argmin}_{\mathbf{w}}\ \mathbf{w}^T \mathbf{S}_B \mathbf{w}$ s.t. $\mathbf{w}^T \mathbf{S}_w \mathbf{w} = 1$
- Lagrange: $L(\mathbf{w}, \lambda) = \mathbf{w}^T \mathbf{S}_B \mathbf{w} + \lambda\,(1 - \mathbf{w}^T \mathbf{S}_w \mathbf{w}\,)$

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



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

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



FLD, V

$$J_S(\mathbf{w}) = \frac{(\mu_+ - \mu_-)^2}{(s_+^2 + s_-^2)} = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}$$

- **Optimal** w^* is eigenvector of $S_B^{-1} S_w$
- When $P(x \mid y_i) \sim N(\mu_i; \Sigma)$

  $\exists$ 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

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
- Fisher’s LD…
  - does not assume Gaussian densities
  - can be used to reduce dimensions even when multiple classes scenario

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*]
- Stay tuned...

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 changing loss functions**
  LMS must re-train the classifier when the loss function changes.
  … no retraining for generative and conditional models
- **Robustness to model assumptions**.
  Generative model usually performs poorly when the assumptions are violated.
  Eg, LDA works poorly if P(x | y) is non-Gaussian.
  Logistic Regression is more robust, … LMS is even more robust
- **Robustness to missing values and noise**.
  In many applications, some of the features x_{ij} may be missing or corrupted for some of the training examples.
  Generative models typically provide better ways of handling this than non-generative models.

Other Algorithms for learning LTUs

- **Naive Bayes** [Discuss later]

  For $K = 2$ classes, produces LTU

- **Winnow** [?Discuss later?]

  Can handle large numbers of “irrelevant" features

  - (features whose weights should be zero)

Learning Theory

Assume data is truly linearly separable. . .

- **Sample Complexity**: Given $\varepsilon, \delta \in (0, 1)$, want LTU has error rate (on new examples)
  - less than ε
  - with probability $> 1 - \delta$.

  Suffices to learn from (be consistent with)

  $$m = O\left(\frac{1}{\epsilon}\left[\ln\frac{1}{\delta} + (n+1)\ln\frac{1}{\epsilon}\right]\right)$$

  labeled training examples.

- **Computational Complexity**:
  There is a polynomial time algorithm for finding a consistent LTU
  (reduction from linear programming)

Agnostic case… different…