Linear Classifiers

R Greiner
Cmput 466/551
Outline

■ Framework
■ Exact
  □ Minimize Mistakes (Perceptron Training)
  □ Matrix inversion (LMS)
■ Logistic Regression
  □ Max Likelihood Estimation (MLE) of $P( y \mid x )$
  □ Gradient descent (MSE; MLE)
  □ Newton-Raphson
■ Linear Discriminant Analysis
  □ Max Likelihood Estimation (MLE) of $P( y, x )$
  □ Direct Computation
  □ Fisher’s Linear Discriminant
Diagnosing Butterfly-itis

Hmmm... perhaps Butterfly-it is??
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 \leq 2, then butterfly-itis

So ? is Not butterfly-itis.
Can be “angled”...

\[ 2.3 \times \#w - 7.5 \times \#a + 1.2 = 0 \]

- If \( 2.3 \times \#Wings - 7.5 \times \#antennae + 1.2 > 0 \)
  then butterfly-itis
Linear Separators, in General

- Given data (many features)

<table>
<thead>
<tr>
<th>F₁</th>
<th>F₂</th>
<th>...</th>
<th>Fₙ</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>95</td>
<td>...</td>
<td>3</td>
<td>No</td>
</tr>
<tr>
<td>22</td>
<td>80</td>
<td>...</td>
<td>-2</td>
<td>Yes</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td></td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>...</td>
<td>1.9</td>
<td>No</td>
</tr>
</tbody>
</table>

- find “weights” \( \{w₁, w₂, \ldots, wₙ, w₀\} \) such that

\[
w₁F₁ + \ldots + wₙFₙ + w₀ > 0
\]

means \( \text{Class} = \text{Yes} \)
Linear Separator

\[ \sum_j w_j \times F_i \]

Just view \( F_0 = 0 \), so \( w_0 \) ...

Yes

No
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 $\mathbf{w} \cdot \mathbf{x} = \sum_i w_i x_i$ 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”

- Squashing function:
  \[ \text{sgn}: \mathbb{R} \rightarrow \{-1, +1\} \]

\[
\text{sgn}(r) = \begin{cases} 
1 & \text{if } r > 0 \\
0 & \text{otherwise}
\end{cases}
\]  
(“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 $\exists$ 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:
    - $O(2^{n^2})$ distinct perceptrons over $n$ boolean features
  - Deterministic
  - Continuous Parameters

- Learning algorithm:
  - Various: Local search, Direct computation, . . .
  - Eager
  - Online / Batch
Task

- Input: labeled data

  Transformed to

  Output: \( w \in \mathbb{R}^{r+1} \)

  **Goal:** Want \( w \) s.t.

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

  \[\square \ldots \text{minimize mistakes wrt data} \ldots\]
Error Function

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

1. Classification error
   Perceptron Training; Matrix Inversion

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

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

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

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

   \[ LCL(w) = \frac{1}{m} \sum_{i=1}^{m} \log P_w(y^{(i)} | x^{(i)}) \]

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 \([x, y]\)
  
  \[ \text{Err} = y - o_w(x) \]
  
  \( y = f(x) \) is target value

  \( o_w(x) = \text{sgn}(w \cdot x) \) is perceptron output

- **Idea:** Move weights in appropriate direction, to push \( \text{Err} \rightarrow 0 \)

- If \( \text{Err} > 0 \) (error on POSITIVE example)
  
  - need to increase \( \text{sgn}(w \cdot x) \)
    
    \[ \Rightarrow \text{need to increase } w \cdot x \]

  - Input \( j \) contributes \( w_j \cdot x_j \) to \( w \cdot x \)
    
    - if \( x_j > 0 \), increasing \( w_j \) will increase \( w \cdot x \)
    
    - if \( x_j < 0 \), decreasing \( w_j \) will increase \( w \cdot x \)

  \[ \Rightarrow w_j \leftarrow w_j + x_j \]

- If \( \text{Err} < 0 \) (error on NEGATIVE example)

  \[ \Rightarrow w_j \leftarrow w_j - x_j \]
Local Search via Gradient Descent

\[ \text{Start } w/ \text{ (random) weight vector } w^0. \]
\[ \text{Repeat until converged } \lor \text{ bored} \]

Compute Gradient
\[ \nabla \text{err}(w^t) = \left( \frac{\partial \text{err}(w^t)}{\partial w_0}, \frac{\partial \text{err}(w^t)}{\partial w_1}, \ldots, \frac{\partial \text{err}(w^t)}{\partial w_n} \right) \]

Let \[ w^{t+1} = w^t + \eta \nabla \text{err}(w^t) \]

If CONVERGED: Return(\(w^t\))
#1a: Mistake Bound Perceptron Alg

Initialize \( \mathbf{w} = 0 \)
Do until bored
  Predict “+” iff \( \mathbf{w} \cdot \mathbf{x} > 0 \)
  else “−"
Mistake on \( y = +1 \): \( \mathbf{w} \leftarrow \mathbf{w} + \mathbf{x} \)
Mistake on \( y = -1 \): \( \mathbf{w} \leftarrow \mathbf{w} - \mathbf{x} \)

<table>
<thead>
<tr>
<th>Weights</th>
<th>Instance</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0 0 0]</td>
<td>#1</td>
<td></td>
</tr>
</tbody>
</table>

### Orig Data

<table>
<thead>
<tr>
<th>( \langle x_1, x_2 \rangle )</th>
<th>( c(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>+</td>
</tr>
<tr>
<td>1 0</td>
<td>−</td>
</tr>
<tr>
<td>1 1</td>
<td>+</td>
</tr>
</tbody>
</table>

### Data + “\( x_0 = 1 \)”

<table>
<thead>
<tr>
<th>( \langle x_0, x_1, x_2 \rangle )</th>
<th>( c(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i_1 ) : 1 0 0 0</td>
<td>+</td>
</tr>
<tr>
<td>( i_2 ) : 1 1 0 0</td>
<td>−</td>
</tr>
<tr>
<td>( i_3 ) : 1 1 1 1</td>
<td>+</td>
</tr>
</tbody>
</table>
Mistake Bound Theorem

Theorem: [Rosenblatt 1960]
If data is consistent w/some linear threshold \( \mathbf{w} \), then number of mistakes is \( \leq (1/\Delta)^2 \),
where
\[
\Delta = \min_{\mathbf{x}} \frac{|\mathbf{w} \cdot \mathbf{x}|}{|\mathbf{w}| \times |\mathbf{x}|}
\]

- \( \Delta \) measures “wiggle room” available:
  - If \( |\mathbf{x}| = 1 \), then \( \Delta \) is max, over all consistent planes, of minimum distance of example to that plane
- \( \mathbf{w} \) is \( \perp \) to separator, as \( \mathbf{w} \cdot \mathbf{x} = 0 \) at boundary
- So \( |\mathbf{w} \cdot \mathbf{x}| \) is projection of \( \mathbf{x} \) onto plane, PERPENDICULAR to boundary line
  … ie, is distance from \( \mathbf{x} \) 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 \( \langle x_0, x_1, \ldots, x_n \rangle 0 \)
   by positive point \( \langle -x_0, -x_1, \ldots, -x_n \rangle 1 \)
3. Normalize all examples to have length 1

- Let \( \mathbf{w}^* \) be unit vector rep'ning target plane
  \[ \Delta = \min_x \{ \mathbf{w}^* \cdot \mathbf{x} \} \]
  Let \( \mathbf{w} \) be hypothesis plane

- Consider:
  \[ \frac{(\mathbf{w} \cdot \mathbf{w}^*)}{|\mathbf{w}|} \]

- On each mistake, add \( \mathbf{x} \) to \( \mathbf{w} \)
  \[ \mathbf{w} = \sum_{\{\mathbf{x} \mid \mathbf{x} \cdot \mathbf{w} < 0\}} \mathbf{x} \]
Proof (con't)

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 $\leq 0 + 1 + \ldots + 1 = m$

so denominator $\leq \sqrt{m}$

• As $(w \cdot w^*)/|w| = \cos($angle between $w$ and $w^*)$
  it must be $\leq 1$, so
  numerator $\leq$ denominator

$$\Rightarrow \Delta \cdot m \leq \sqrt{m} \Rightarrow m \leq \frac{1}{\Delta^2}$$
#1b: Perceptron Training Rule

- For each labeled instance \([x, y]\)
  \[
  \text{Err(} [x, y] \text{)} = y - o_w(x) \in \{ -1, 0, +1 \}
  \]

  - If \(\text{Err(} [x, y] \text{)} = 0\)  Correct! ... Do nothing!
    \[
    \Delta w = 0 \equiv \text{Err(} [x, y] \text{)} \cdot x
    \]

  - If \(\text{Err(} [x, y] \text{)} = +1\)  Mistake on positive!  Increment by \(+x\)
    \[
    \Delta w = +x \equiv \text{Err(} [x, y] \text{)} \cdot x
    \]

  - If \(\text{Err(} [x, y] \text{)} = -1\)  Mistake on negative!  Increment by \(-x\)
    \[
    \Delta w = -x \equiv \text{Err(} [x, y] \text{)} \cdot x
    \]

In all cases...

\[
\Delta w(i) = \text{Err(} [x(i), y(i)] \text{)} \cdot x(i) = [y(i) - o_w(x(i))] \cdot x(i)
\]

- Batch Mode: do ALL updates at once!

\[
\Delta w_j = \sum_i \Delta w_{j(i)}
\]

\[
= \sum_i x_{(i)j} (y(i) - o_w(x(i)))
\]

\[
W_j += \eta \Delta w_j
\]

\(\eta\) is **learning rate** (small pos “constant” ... \(\approx 0.05\))
0. New $\mathbf{w}$

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

1. For each row $i$, compute

a. $E^{(i)} := y^{(i)} - o_w(x^{(i)})$

b. $\Delta \mathbf{w} += E^{(i)} x^{(i)}$

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

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

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

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

- Proof: Weight space has **EXACTLY 1 minimum!**
  (no non-global minima)
  $\Rightarrow$ with enough examples, finds correct function!

- Explains early popularity

- If $\eta$ too large, may overshoot
  - If $\eta$ too small, takes too long

- So often $\eta = \eta(k)$ ... which decays with # of iterations, $k$
#1c: Matrix Version?

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

Find \( \{ w_i \} \) s.t.

\[
\begin{align*}
y^1 &= w_0 + w_1 x_1^1 + \cdots + w_n x_n^1 \\
y^2 &= w_0 + w_1 x_1^2 + \cdots + w_n x_n^2 \\
&\vdots \\
y^m &= w_0 + w_1 x_1^m + \cdots + w_n x_n^m
\end{align*}
\]

- Linear Equalities \( y = X w \)

- Solution: \( w = X^{-1} y \)
Issues

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 $y^i \in \mathbb{R}$: Regression

2. What if NO $w$ works?
   ...$X$ is singular; overconstrained ...
   Could try to minimize residual

\[
\sum_i I[ y^{(i)} \neq w \cdot x^{(i)} ]
\]
\[
\| y - Xw \|_1 = \sum_i | y^{(i)} - w \cdot x^{(i)} |
\]
\[
\| y - Xw \|_2 = \sum_i ( y^{(i)} - w \cdot x^{(i)} )^2
\]

NP-Hard!

Easy!
L₂ 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 differentiable with respect to 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 separable!

Stay tuned!
#1. LMS version of Classifier

- **View as Regression**
  - Find “best” linear mapping \( \mathbf{w} \) from \( \mathbf{X} \) to \( \mathbf{Y} \)
  - \( \mathbf{w}^* = \arg\min \ Err_{LMS}^{(\mathbf{X}, \mathbf{Y})}(\mathbf{w}) \)
  - \( Err_{LMS}^{(\mathbf{X}, \mathbf{Y})}(\mathbf{w}) = \sum_i \ (y^{(i)} - \mathbf{w} \cdot \mathbf{x}^{(i)})^2 \)

- Threshold: if \( \mathbf{w}^\top \mathbf{x} > 0.5 \), 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 $k = \arg\max_j \delta_j(x)$

- 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: $Y = (Y_1, Y_2, Y_3) \in \{[1,0,0], [0,1,0], [0,0,1]\}$

- Training sample (N=5):

\[
X = \begin{bmatrix}
1 & x_{11} & x_{12} \\
1 & x_{21} & x_{22} \\
1 & x_{31} & x_{32} \\
1 & x_{41} & x_{42} \\
1 & x_{51} & x_{52}
\end{bmatrix}, \quad Y = \begin{bmatrix}
y_{11} & y_{12} & y_{13} \\
y_{21} & y_{22} & y_{23} \\
y_{31} & y_{32} & y_{33} \\
y_{41} & y_{42} & y_{43} \\
y_{51} & y_{52} & y_{53}
\end{bmatrix}
\]

- Regression output:

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

- Classification rule:

\[
\hat{G}((x_1, x_2)) = \arg\max_k \hat{Y}_k((x_1, x_2))
\]

\[
\hat{Y}_1((x_1, x_2)) = (1 \ x_1 \ x_2)\beta_1 \\
\hat{Y}_2((x_1, x_2)) = (1 \ x_1 \ x_2)\beta_2 \\
\hat{Y}_3((x_1, x_2)) = (1 \ x_1 \ x_2)\beta_3
\]
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 $P_w(y=1|\, x)$ ... based on parameters $w$
- But …
  - $w \cdot x$ 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:
  - \( \sigma : \mathbb{R} \rightarrow [0, 1] \)
  - \( \frac{\partial \sigma(x)}{\partial x} = \sigma(x) (1 - \sigma(x)) \)
  - If \( x \approx 0 \), then \( \sigma(x) \approx x \)
Logistic Regression (con’t)

Assume 2 classes:

\[ P_w(y = 1 \mid x) = \sigma(w \cdot x) = \frac{1}{1 + e^{-(x \cdot w)}} \]
\[ P_w(y = 0 \mid x) = 1 - \frac{1}{1 + e^{-(x \cdot w)}} = \frac{e^{-(x \cdot w)}}{1 + e^{-(x \cdot w)}} \]

Log Odds:

\[ \log \frac{P_w(y = 1 \mid x)}{P_w(y = 0 \mid x)} = x \cdot w \]
How to learn parameters $w$?

... depends on goal?

- A: Minimize MSE?
  \[ \sum_i ( y^{(i)} - o_w(x^{(i)}) )^2 \]

- B: Maximize likelihood?
  \[ \sum_i \log P_w(y^{(i)} | x^{(i)}) \]
MS Error Gradient for Sigmoid Unit

- **Error:** \[ \sum_j ( y^{(j)} - o_w(x^{(j)}) )^2 = \sum_j E^{(j)} \]
  - For single training instance

- **Input:** \( x^{(j)} = [x^{(j)}_1, \ldots, x^{(j)}_k] \)

- **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)**

\[
\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \left[ \frac{1}{2} (o - y)^2 \right] = \frac{1}{2} \left[ 2(o - y) \frac{\partial}{\partial w_i} (o - y) \right] = (o - y) \frac{\partial o}{\partial w_i} = (o - y) \frac{\partial \sigma(z)}{\partial z} \frac{\partial z}{\partial w_i}
\]
Derivative of Sigmoid

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

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

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

\[
\frac{\partial \sigma(z)}{\partial z} = \sigma(z) (1 - \sigma(z)) = o(1 - o)
\]

\[
\frac{\partial z}{\partial w_i} = \frac{\partial (\sum_i w_i \cdot x_i)}{\partial w_i} = x_i
\]

\[
\frac{\partial E(j)}{\partial w_i} = (o(j) - y(j)) o(j) (1 - o(j)) x_i(j)
\]

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 \( w_i += \Delta w_i \) where

\[
\Delta w_i = \eta \cdot \frac{\partial E(j)}{\partial w_i}
\]
(LMS)

0. New $w$
   \[ \Delta w = 0 \]

1. For each row $i$, compute
   a. $E^{(i)} = (o^{(i)} - y^{(i)}) \cdot o^{(i)} \cdot (1 - o^{(i)})$
   b. $\Delta w += E^{(i)} \cdot x^{(i)}$
      [ \[ \ldots \Delta w_j += E^{(i)} \cdot x^{(i)}_j \ldots \] ]

2. Increment $w += \eta \Delta w$
B: Or... Learn Conditional Probability

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

\[
\text{Goal: } w^* = \arg\max_w P(w | S) \\
= \arg\max_w \frac{P(S | w)P(w)}{P(S)} \\
= \arg\max_w P(S | w)P(w) \\
= \arg\max_w P(S | w) \\
= \arg\max_w \log P(S | w)
\]

Bayes Rules

As \( P(S) \) does not depend on \( w \)

As \( P(w) \) is uniform

As \( \log \) is monotonic
ML Estimation

- \( P(S \mid w) \equiv \text{likelihood function} \)
  \[
  L(w) = \log P(S \mid w)
  \]
- \( w^* = \arg\max_w L(w) \)
  is “maximum likelihood estimator” (MLE)
Computing the Likelihood

- As training examples \([x^{(i)}, y^{(i)}]\) are iid
  - drawn independently from same (unknown) prob \(P_w(x, y)\)
- \(\log P(S \mid w) = \log \prod_i P_w(x^{(i)}, y^{(i)})\)
  \[= \sum_i \log P_w(x^{(i)}, y^{(i)})\]
  \[= \sum_i \log P_w(y^{(i)} \mid x^{(i)}) + \sum_i \log P_w(x^{(i)})\]
- Here \(P_w(x^{(i)}) = 1/n \ldots\)
  - not dependent on \(w\), over empirical sample \(S\)
- \(w^* = \text{argmax}_w \sum_i \log P_w(y^{(i)} \mid x^{(i)})\)
Fit Logistic Regression…
by Gradient Ascent

- Want $w^* = \arg\max_w J(w)$
  - $J(w) = \sum_i r(y^{(i)}, x^{(i)}, w)$
  - For $y \in \{0, 1\}$
    $$ r(y, x, w) = \log P_w(y | x) = y \log( P_w(y=1 | x)) + (1 - y) \log(1 - P_w(y=1 | x)) $$

- So climb along…
  $$ \frac{\partial J(w)}{\partial w_j} = \sum_i \frac{\partial r(y^{(i)}, x^{(i)}, w)}{\partial w_j} $$
Gradient Descent …

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

\[
= \frac{y}{p_1} \frac{\partial p_1}{\partial w_j} + (-1) \times \frac{1 - y}{1 - p_1} \frac{\partial p_1}{\partial w_j} = \frac{y - p_1}{p_1(1 - p_1)} \frac{\partial p_1}{\partial w_j}
\]

\[
\frac{\partial p_1}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{P_w(y = 1| x)}{\partial w_j} = \frac{\partial}{\partial w_j} (\sigma(x \cdot w))
\]

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

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

\[
= \sum_i (y^{(i)} - P_w(y = 1| x)) \cdot x_j^{(i)}
\]
Gradient Ascent for Logistic Regression (MLE)

Given: training examples \( \langle x^{(i)}, y^{(i)} \rangle, \ i = 1..N \)
Set initial weight vector \( w = \langle 0, 0, 0, 0, \ldots, 0 \rangle \)
Repeat until convergence
  Let gradient vector \( \Delta w = \langle 0, 0, 0, 0, \ldots, 0 \rangle \)
  For \( i = 1 \) to \( N \) do
    \( p_1^{(i)} = 1/(1 + \exp[w \cdot x^{(i)}]) \)
    \( \text{error}_i = y^{(i)} - p_1^{(i)} \)
    For \( j = 1 \) to \( n \) do
      \( \Delta w_j \leftarrow \text{error}_i \cdot x_{ij} \)
      \( w \leftarrow w + \eta \Delta w \) % step in direction of increasing gradient
Comments on MLE 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-Rewighted Least Squares” (IRLS)
Use Logistic Regression for Classification

- Return YES iff

\[
P(y = 1 \mid x) > P(y = 0 \mid x) > 1
\]

\[
\ln \frac{P(y = 1 \mid x)}{P(y = 0 \mid x)}> 0
\]

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

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

Logistic Regression learns a LTU!
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 \( K \):

\[
\begin{align*}
\log \frac{P(y = 1 | x)}{P(y = K | x)} &= w_1 \cdot x \\
\log \frac{P(y = 2 | x)}{P(y = K | x)} &= w_2 \cdot x \\
&\vdots \\
\log \frac{P(y = K-1 | x)}{P(y = K | x)} &= w_{K-1} \cdot x
\end{align*}
\]

Note: \( k-1 \) different \( w_i \) weights, ... each of dimension \(|x|\)

- Apply gradient ascent to learn all \( w_k \) weight vectors, in parallel.

- Conditional probabilities:

\[
P(y = k | x) = \frac{\exp(w_k \cdot x)}{1 + \sum_{\ell=1}^{K-1} \exp(w_\ell \cdot x)}
\]

and

\[
P(y = K | x) = \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(w_\ell \cdot x)}
\]
Learning LR Weights

**Task:** Given data \( \langle x^{(i)}, y^{(i)} \rangle \), find \( w \) in \( p_w(y|x) = \frac{1}{1 + \exp(-w \cdot x)} \) if \( y = 1 \)
\[
\frac{\exp(-w \cdot x)}{1 + \exp(-w \cdot x)} \quad \text{if} \quad y = 0
\]
s.t. \( p_w(y^{(i)}|x^{(i)}) > \frac{1}{2} \) iff \( y^{(i)} = 1 \)

**Approach 1:** MSE – “Neural nets”
Minimize \( \sum_i (o^{(i)} - y^{(i)})^2 \)
Gradient:
\[
\Delta w^{(i)}_j = \ (o^{(i)} - y^{(i)}) \ o^{(i)} \ (1 - o^{(i)} )
\]

**Approach 2:** MLE – “Logistic Regression”
Maximize \( \sum_i p_w(y|x) \)
Gradient:
\[
\Delta w^{(i)}_j = \ (y^{(i)} - p(1|x^{(i)})) \ x^{(i)}_j
\]
0. New \( w \)
\[ \Delta w = 0 \]

1. For each row \( i \), compute
   a. \( E^{(i)} = (y^{(i)} - p(1|x^{(i)})) \)
   b. \( \Delta w += E^{(i)} x^{(i)} \)
      \[ \ldots \Delta w_j += E^{(i)} x^{(i)}_j \ldots \]

2. Increment \( w += \eta \Delta w \)
Logistic Regression Computation...

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

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

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

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

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

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

\[ \beta^{\text{new}} = \beta^{\text{old}} - \left[ \text{Jacobian}(\frac{\partial l(\beta^{\text{old}})}{\partial \beta}) \right]^{-1} \frac{\partial l(\beta^{\text{old}})}{\partial \beta} \]
Newton-Raphson Method

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

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

Iteration…
Newton-Raphson in Multi-dimensions

- To solve the equations:

\[
\begin{align*}
f_1(x_1, x_2, \ldots, x_N) &= 0 \\
f_2(x_1, x_2, \ldots, x_N) &= 0 \\
& \vdots \\
f_N(x_1, x_2, \ldots, x_N) &= 0
\end{align*}
\]

- Taylor series:

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

- N-R:

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

Jacobian matrix
Newton-Raphson : Example

**Solve**

\[
f_1(x_1, x_2) = x_1^2 - \cos(x_2) = 0
\]
\[
f_2(x_1, x_2) = \sin(x_1) + x_1^2 + x_2^3 = 0
\]

\[
\begin{bmatrix}
x_1^{n+1} \\
x_2^{n+1}
\end{bmatrix}
= \begin{bmatrix}
x_1^n \\
x_2^n
\end{bmatrix}
- \begin{bmatrix}
2x_1^n & \sin(x_2^n) \\
\cos(x_1^n) + 2x_1^n & 3(x_2^n)^2
\end{bmatrix}^{-1}
\begin{bmatrix}
(x_1^n)^2 - \cos(x_2^n) \\
\sin(x_1^n) + (x_1^n)^2 + (x_2^n)^3
\end{bmatrix}
\]
Maximum Likelihood Parameter Estimation

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

\[ p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \]

given \( N \) independent samples, \( \{x_1, \ldots, x_N\} \)

- Estimate the parameters that maximize the likelihood function

\[ L(\mu, \sigma) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right) \]

\[ (\hat{\mu}, \hat{\sigma}) = \arg \max_{\mu,\sigma} L(\mu, \sigma) \]
Logistic Regression Algs for LTUs

- Learns Conditional Probability Distribution $P(y \mid x)$

- **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 $P_w(y \mid x)$ 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

LDA can avoid this masking
#4: Linear Discriminant Analysis

- LDA learns joint distribution $P(y, x)$
  - As $P(y, x) \neq P(y | x)$; optimizing $P(y, x)$ is not equivalent to optimizing $P(y | x)$
- “generative model”
  - $P(y, x)$ model of how data is generated
  - Eg, factor
    $$P(y, x) = P(y) P(x | y)$$
    - $P(y)$ generates value for $y$; then
    - $P(x | y)$ generates value for $x$ 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 \( \mu_k \) and covariance \( \Sigma \)

\[
P(x | y = k) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (x - \mu_k)^\top \Sigma^{-1} (x - \mu_k) \right]
\]
Estimating LDA Model

- Linear discriminant analysis assumes form

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

- \(\mu_y\) is mean for examples belonging to class \(y\); covariance matrix \(\Sigma\) is shared by all classes!

- Can estimate LDA directly:
  
  \(m_k = \#\text{training examples in class } y = k\)

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

  - \(\hat{\mu}_k = \frac{1}{m} \sum_{i:y_i=k} x_i\)

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

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

- \( m=7 \) examples;
  \( m_+ = 3 \) positive; \( m_- = 4 \) negative
  \[ p_+ = \frac{3}{7} \quad p_- = \frac{4}{7} \]

- Compute \( \hat{\mu}_i \) over each class

\[
\begin{align*}
\hat{\mu}_+ &= \frac{1}{3} \sum_{i: \langle y^{(i)} = + \rangle} x^{(i)} \\
&= \frac{1}{3} \left( [13.1, 20.2, 0.4]^T + [6.0, 17.7, -4.2]^T + [8.2, 18.2, -2.5]^T \right) \\
&= [9.1, 18.7, -2.1]^T \\
\hat{\mu}_- &= \frac{1}{4} \sum_{i: \langle y^{(i)} = - \rangle} x^{(i)} = [-1.8, 12.0, 16.3]^T
\end{align*}
\]

Note: do NOT pre-pend \( x_0 = 1 \)!
Estimation...

- Compute common $\hat{\Sigma}$
  
  - “Normalize” each $z := x - \mu_y(x)$
    
    $= [4.0, 1.5, -1.7]^T$
    
    $\ldots$
    
    $z^{(4)} := [0.4, 10.1, 19.2]^T - [-1.8, 12.0, 16.3]^T$
    $= [2.2, -1.9, 2.9]^T$
    
    $\ldots z^{(7)} := \ldots$

  - Compute covariance matrix, for each $i$:
    
    For $x^{(1)}$, via $z^{(1)}$:
    
    $z^{(1)} \times z^{(1)^T} = \begin{bmatrix} 4.0 \\ 0.5 \\ -1.7 \end{bmatrix} \cdot [4.0, 0.5, -1.7]^T$
    
    $= \begin{bmatrix} 4.0 \cdot 4.0 & 4.0 \cdot 0.5 & 4.0 \cdot -1.7 \\ 0.5 \cdot 4.0 & 0.5 \cdot 0.5 & 0.5 \cdot -1.7 \\ -1.7 \cdot 4.0 & -1.7 \cdot 0.5 & -1.7 \cdot -1.7 \end{bmatrix}$
    
    $= \begin{bmatrix} 16.0 & 2.0 & -6.8 \\ 2.0 & 0.25 & -0.85 \\ -6.8 & -0.85 & -2.89 \end{bmatrix}$
    
    - Set $\hat{\Sigma} = \frac{1}{m} \sum_i z^{(i)}z^{(i)^T}$
Classifying, Using LDA

How to classify new instance, given estimates

\[ \hat{\mu}_+ = \begin{bmatrix} 9.1 \\ 18.7 \\ -2.1 \end{bmatrix}^T \]
\[ \hat{\mu}_- = \begin{bmatrix} -1.8 \\ 12.0 \\ 16.3 \end{bmatrix}^T \]
\[ \hat{\Sigma} = \begin{bmatrix} 7.22 & -1.31 & 6.35 \\ -1.31 & 2.91 & 0.32 \\ 6.35 & 0.32 & 26.03 \end{bmatrix} \]

Class for instance \( \mathbf{x} = [5, 14, 6]^T \) ?

\[
= \frac{3}{7} \times P(x = [5, 14, 6]^T | x \sim \mathcal{N}(\hat{\mu}_+, \hat{\Sigma})) \\
= \frac{3}{7} \times \frac{1}{(2\pi)^{3/2} |\hat{\Sigma}|^{1/2}} \exp \left[ -\frac{1}{2} (x - \hat{\mu}_+)^T \hat{\Sigma}^{-1} (x - \hat{\mu}_+) \right] \\
= 16.63 \times 10^{-11}
\]

\[
= \frac{4}{7} \times \frac{1}{(2\pi)^{3/2} |\hat{\Sigma}|^{1/2}} \exp \left[ -\frac{1}{2} (x - \hat{\mu}_-)^T \hat{\Sigma}^{-1} (x - \hat{\mu}_-) \right] \\
= 43.33 \times 10^{-11}
\]


\[ P(y = - | [5, 14, 6]^T) = 0.7226 \]
LDA learns an LTU

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

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

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

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

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

1. \((x - \mu_1)^T \Sigma^{-1} (x - \mu_1) - (x - \mu_0)^T \Sigma^{-1} (x - \mu_0)\)
   \[= x^T \Sigma^{-1} (\mu_0 - \mu_1) + (\mu_0 - \mu_1)^T \Sigma^{-1} x + \mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0\]

2. As \(\Sigma^{-1}\) is symmetric,
   \[... = 2 \ x^T \Sigma^{-1} (\mu_0 - \mu_1) + \mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0\]

\[
\Rightarrow \ln \frac{P(x, y = 1)}{P(x, y = 0)} = \ln \frac{P(y = 1)}{P(y = 0)} - \frac{1}{2} [(x - \mu_1)^T \Sigma^{-1} (x - \mu_1) - (x - \mu_0)^T \Sigma^{-1} (x - \mu_0)]
\]

\[
= \ln \frac{P(y=1)}{P(y=0)} + x^T \Sigma^{-1} (\mu_1 - \mu_0) + \frac{1}{2} \mu_0^T \Sigma^{-1} \mu_0 - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1
\]

\[
= \ x^T \Sigma^{-1} (\mu_1 - \mu_0) + \ln \frac{P(y=1)}{P(y=0)} + \frac{1}{2} \mu_0^T \Sigma^{-1} \mu_0 - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1
\]
LDA Learns an LTU (3)

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

- So let…

\[
\begin{align*}
w &= \Sigma^{-1}(\mu_1 - \mu_0) \\
c &= \ln \frac{P(y=1)}{P(y=0)} + \frac{1}{2} \mu_0^\top \Sigma^{-1} \mu_0 - \frac{1}{2} \mu_1^\top \Sigma^{-1} \mu_1
\end{align*}
\]

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

LTU!!
LDA: Example

LDA was able to avoid masking here
View LDA wrt Mahalanobis Distance

- Squared Mahalanobis distance between $x$ and $\mu$:

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

- $\Sigma^{-1} \approx$ linear distortion
  ... converts standard Euclidean distance into Mahalanobis distance.

- LDA classifies $x$ as 0 if

$$D_M^2(x, \mu_0) < D_M^2(x, \mu_1)$$

- $\log P(x \mid y = k) \approx \log \pi_k - \frac{1}{2} D_M^2(x, \mu_k)$
Generalizations of LDA

- **General Gaussian Classifier: QDA**
  Allow each class $k$ to have its own $\Sigma_k$
  $\Rightarrow$ Classifier $\equiv$ *quadratic* threshold unit (not LTU)

- **Naïve Gaussian Classifier**
  Allow each class $k$ to have its own $\Sigma_k$
  but require each $\Sigma_k$ 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Same for all classes?</th>
<th>Diagonal</th>
<th>#param’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA</td>
<td>+</td>
<td>+</td>
<td>$k$</td>
</tr>
<tr>
<td>Naïve Gaussian Classifier</td>
<td>—</td>
<td>+</td>
<td>$k n$</td>
</tr>
<tr>
<td>General Gaussian Classifier</td>
<td>—</td>
<td>—</td>
<td>$k n^2$</td>
</tr>
</tbody>
</table>

- Covariance matrix $\Sigma$
- $n$ features; $k$ classes
Versions of LDA

- 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 $\Sigma^{-1}$
  [Easy if $\Sigma^{-1}$ is diagonal. . . ]
Fisher's Linear Discriminant

- **LDA**
  - Finds $K-1$ dim hyperplane 
    $(K = \text{number of classes})$
  - Project $\mathbf{x}$ and $\{ \mu_k \}$ to that hyperplane
  - Classify $\mathbf{x}$ as nearest $\mu_k$ within hyperplane

- **Better:**
  Find hyperplane that maximally separates projection of $\mathbf{x}$'s wrt $\sum^{-1}$

Fisher’s Linear Discriminant
Fisher Linear Discriminant

- Recall any vector \( \mathbf{w} \) projects \( \mathbb{R}^n \to \mathbb{R} \)
- Goal: Want \( \mathbf{w} \) 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 $m_+ = \frac{\sum_i y^{(i)} \cdot x^{(i)}}{\sum_i y^{(i)}}$ and $m_- = \frac{\sum_i (1-y^{(i)}) \cdot x^{(i)}}{\sum_i (1-y^{(i)})}$

  Mean of $x$'s projections:
  $\mu_+ = \frac{\sum_i y^{(i)} w^T \cdot x^{(i)}}{\sum_i y^{(i)}} = w^T \cdot m_+$
  $\mu_- = \frac{\sum_i (1-y^{(i)}) w^T \cdot x^{(i)}}{\sum_i (1-y^{(i)})} = w^T \cdot m_-

- Problem with $m_+ - m_-$:
  - Does not consider “scatter” within class
  - Goal: Want $w$ that “separates” classes

  - Each $w \cdot x^+$ far from each $w \cdot x^-$
  - Positive $x^+$'s: $w \cdot x^+$ close to each other
  - Negative $x^-$'s: $w \cdot x^-$ close to each other

- “scatter” of $+instance; -instance$

  - $s_+^2 = \sum_i y^{(i)} \left( w \cdot x^{(i)} - m_+ \right)^2$
  - $s_-^2 = \sum_i (1 - y^{(i)}) \left( w \cdot x^{(i)} - m_- \right)^2$
Fisher Linear Discriminant

- Recall any vector $\mathbf{w}$ projects $\mathbb{R}^n \rightarrow \mathbb{R}$
- Goal: Want $\mathbf{w}$ 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 $\mathbf{m}_+ = \frac{\sum_i y(i) \cdot \mathbf{x}(i)}{\sum_i y(i)}$ \hspace{1cm} $\mathbf{m}_- = \frac{\sum_i (1-y(i)) \cdot \mathbf{x}(i)}{\sum_i (1-y(i))}$

  Mean of $\mathbf{x}$’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 $+\text{instance};\; -\text{instance}$
  - $\mathbf{s}_+^2 = \sum_i y(i) \left( \mathbf{w} \cdot \mathbf{x}(i) - \mathbf{m}_+ \right)^2$
  - $\mathbf{s}_-^2 = \sum_i (1-y(i)) \left( \mathbf{w} \cdot \mathbf{x}(i) - \mathbf{m}_+ \right)^2$
FLD, con't

- Separate means $m_-$ and $m_+$
  \[ \Rightarrow \text{maximize } (m_- - m_+)^2 \]
- Minimize each spread $s_+^2$, $s_-^2$
  \[ \Rightarrow \text{minimize } (s_+^2 + s_-^2) \]
- Objective function: maximize

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

\#1: $(\mu_- - \mu_+)^2 = (w^T m_+ - w^T m_-)^2$
\[ = w^T (m_+ - m_-)(m_+ - m_-)^T w = w^T S_B w \]

“between-class scatter”

\[
S_B = (m_+ - m_-)(m_+ - m_-)^T
\]
FLD, III

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

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

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

... “within-class scatter matrix” for +

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

... “within-class scatter matrix” for –

- \[ S_W = S_+ + S_- \] so \[ s_+^2 + s_-^2 = w^T S_W w \]
FLD, IV

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

- Minimizing \( J_S(w) \) ...
  \[ w^* = \arg\min_w w^T S_B w \quad \text{s.t.} \quad w^T S_w w = 1 \]
- Lagrange: \( L(w, \lambda) = w^T S_B w + \lambda (1 - w^T S_w w) \)
  \[ \frac{\partial L(w, \lambda)}{\partial w} = 2S_B w - \lambda (2S_w w) \]
  \[ \frac{\partial L(w, \lambda)}{\partial w} = 0 \quad \Rightarrow \quad S_B^{-1} S_w w = \frac{1}{\lambda} \]

- \( w^* \) is eigenvector of \( S_B^{-1} S_w \)
FLD, V

- **Optimal** $w^*$ is eigenvector of $S_B^{-1}S_w$

- When $P(x | y_i) \sim N(\mu_i; \Sigma)$
  - ∃ LINEAR DISCRIMINANT: $w = \Sigma^{-1}(\mu_+ - \mu_-)$
  - ⇒ 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: \textit{LMS, LR, LDA, FLD}?

Ongoing debate within machine learning community about relative merits of

- direct classifiers \([ LMS ]\)
- conditional models \(P( y \mid x )\) \([ LR ]\)
- 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 $\epsilon, \delta \in (0, 1)$, want LTU has error rate (on new examples)
  - less than $\epsilon$
  - 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...