

# Linear Threshold Units - II

Logisitic Regression and  
Linear Discriminant Analysis

# Logistic Regression

- Learn the conditional distribution  $P(y | \mathbf{x})$
- Let  $p_y(\mathbf{x}; \mathbf{w})$  be our estimate of  $P(y | \mathbf{x})$ , where  $\mathbf{w}$  is a vector of parameters (adjustable). Assume two classes  $y=0$  and  $y=1$

$$p(y = 1 | \mathbf{x}; \mathbf{w}) = p_1(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

$$p(y = 0 | \mathbf{x}; \mathbf{w}) = 1 - p_1(\mathbf{x})$$

- You will show in homework that, this is equivalent to

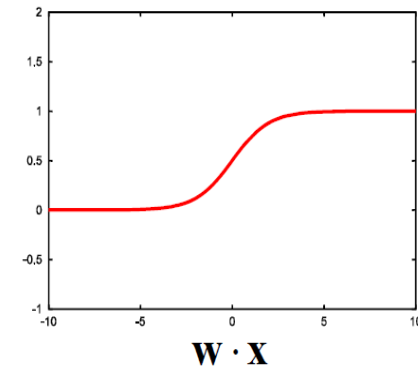
$$\log \frac{p(y = 1 | \mathbf{x}; \mathbf{w})}{p(y = 0 | \mathbf{x}; \mathbf{w})} = \mathbf{w} \cdot \mathbf{x}$$

i.e., the log odds of class 1 is a linear function of  $\mathbf{x}$ .

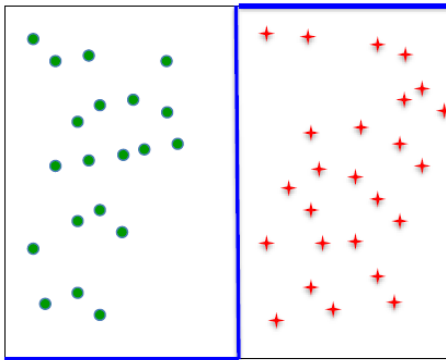
# Why the exp function?

- Differentiable
- Easy to learn
- Handles noisy labels naturally

$$g(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w} \cdot \mathbf{x})}$$

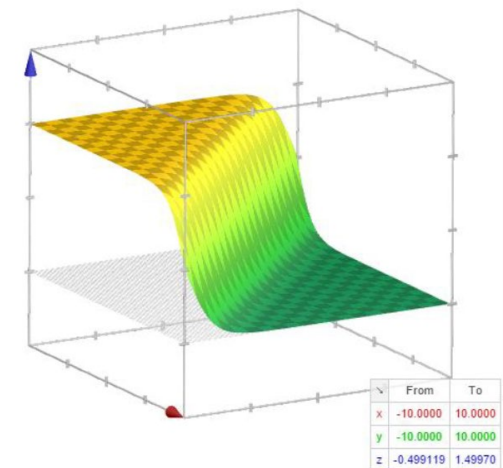


$$p(Y = 1|x) = 0$$



$$p(Y = 1|x) = 1$$

A linear function has a range from  $[-\infty, \infty]$ , the logistic function transforms the range to  $[0, 1]$  to be a probability.



Not differentiable!

# Deriving a Learning Algorithm

- Since we are fitting a conditional probability distribution, we no longer seek to minimize the loss on the training data. Instead, we are interested in finding the distribution  $h$  that is most likely given the training data
- Let  $S$  be the training sample. Our goal is to find  $h$  to maximize  $P(h|S)$

$$\begin{aligned}
 \arg \max_h P(h|S) &= \arg \max_h \frac{P(S|h)P(h)}{P(S)} && \text{by Bayes' Rule} \\
 &= \arg \max_h P(S|h)P(h) && \text{because } P(S) \text{ doesn't depend on } h \\
 &= \arg \max_h P(S|h) && \text{assuming } P(h) \text{ is uniform} \\
 &= \arg \max_h \log P(S|h) && \text{Because log is monotonic}
 \end{aligned}$$

The distribution of  $P(S|h)$  is called the likelihood function. The log likelihood is frequently used as the objective function for learning. It is often written as  $l(\mathbf{w})$ .

The  $h$  that maximizes the likelihood on the training data is called the maximum likelihood estimator (MLE)

# Computing the Likelihood

- In our framework, we assume that each training example is iid
- $\log P(S|h) = \log \prod_i P(\mathbf{x}_i, y_i|h)$   
 $= \sum_i \log P(\mathbf{x}_i, y_i|h)$
- This shows that the log likelihood of S is the sum of the log likelihoods of the individual training examples

# Computing the Likelihood

- Recall that any joint distribution  $P(\mathbf{a}, \mathbf{b})$  can be factored as  $P(\mathbf{a} | \mathbf{b})P(\mathbf{b})$ . Hence, we can write

$$\begin{aligned}\arg \max_h \log P(S | h) &= \arg \max_h \sum_i \log P(\mathbf{x}_i, y_i | h) \\ &= \arg \max_h \sum_i \log P(y_i | \mathbf{x}_i, h) P(\mathbf{x}_i | h)\end{aligned}$$

- In our case,  $P(\mathbf{x} | h) = P(\mathbf{x})$ , since it does not depend on  $h$

$$\arg \max_h \log P(S | h) = \arg \max_h \sum_i \log P(y_i | \mathbf{x}_i, h)$$

# Computing the Likelihood

- Consider an example  $(\mathbf{x}_i, y_i)$ 
  - If  $y_i=0$  the log likelihood is  $\log [1 - p_1(\mathbf{x}; \mathbf{w})]$
  - If  $y_i=1$  the log likelihood is  $\log [p_1(\mathbf{x}; \mathbf{w})]$
- Note that these cases are mutually exclusive and hence can be combined

$$l(y_i; \mathbf{x}_i, \mathbf{w}) = \log P(y_i | \mathbf{x}_i, \mathbf{w}) = (1 - y_i) \log[1 - p_1(\mathbf{x}_i, \mathbf{w})] + y_i \log[p_1(\mathbf{x}_i, \mathbf{w})]$$

- The goal of our learning algorithm will be to find  $\mathbf{w}$  to maximize

$$J(\mathbf{w}) = \sum_i l(y_i; \mathbf{x}_i, \mathbf{w})$$

# Fitting Logistic Regression by Gradient Descent

On the board

Over all gradient is

$$\frac{\partial J(w)}{\partial w_j} = \sum_i (y_i - p_1(\mathbf{x}_i; \mathbf{w})) x_{ij}$$



# Batch Gradient Ascent for Logistic Regression

**Given:** Training examples  $(\mathbf{x}_i, y_i), i = 1, \dots, N$

Let  $\mathbf{w} = (0, 0, \dots, 0)$  be the initial weight vector

**Repeat** until convergence

Let  $\mathbf{g} = (0, \dots, 0)$  be the initial gradient vector

For  $i = 1$  to  $N$  do

$$p_i = 1 / (1 + \exp[\mathbf{w} \cdot \mathbf{x}_i])$$

$$error = y_i - p_i$$

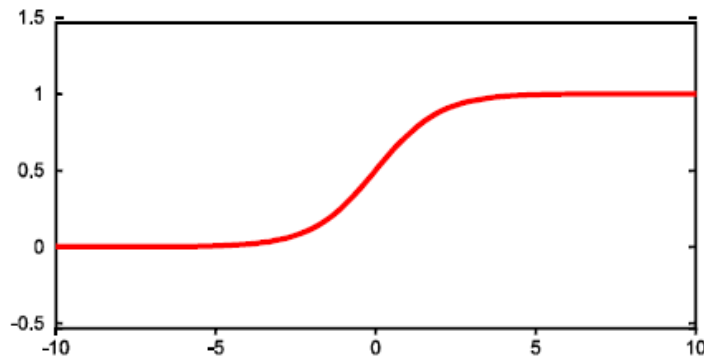
For  $j = 1$  to  $n$  do

$$g_j = g_j + error \cdot x_{ij}$$

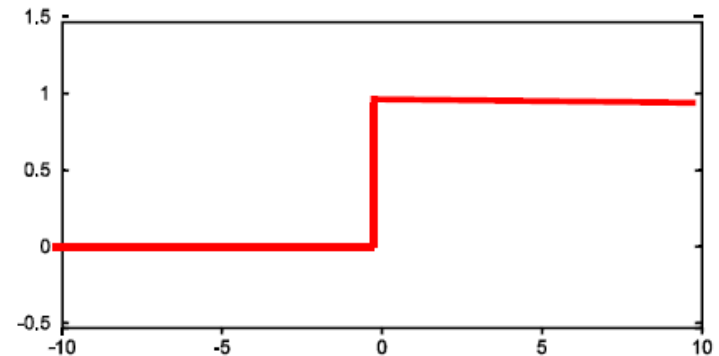
$$\mathbf{w} := \mathbf{w} + \eta \mathbf{g}$$

# Connection between Logistic Regression and Perceptron Algorithm

If we replace the logistic function with a step function:



$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$



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

# Multi-Class Logistic Regression

- Choose class  $K$  to be the “reference class” and represent each of the other classes as a logistic function of the odds of class  $k$  versus class  $K$ :

$$\begin{aligned}\log \frac{P(y = 1 \mid \mathbf{x})}{P(y = K \mid \mathbf{x})} &= \mathbf{w}_1 \cdot \mathbf{x} \\ \log \frac{P(y = 2 \mid \mathbf{x})}{P(y = K \mid \mathbf{x})} &= \mathbf{w}_2 \cdot \mathbf{x} \\ &\vdots \\ \log \frac{P(y = K - 1 \mid \mathbf{x})}{P(y = K \mid \mathbf{x})} &= \mathbf{w}_{K-1} \cdot \mathbf{x}\end{aligned}$$

- Gradient ascent can be applied to simultaneously train all weight vectors  $\mathbf{w}_k$

# Multi-Class Logistic Regression

- Conditional probability for class  $k \neq K$  can be computed as

$$P(y = k \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_k \cdot \mathbf{x})}{1 + \sum_{l=1}^{K-1} \exp(\mathbf{w}_l \cdot \mathbf{x})}$$

- For class  $K$ , the conditional probability is

$$P(y = K \mid \mathbf{x}) = \frac{1}{1 + \sum_{l=1}^{K-1} \exp(\mathbf{w}_l \cdot \mathbf{x})}$$

# Logistic Regression Implements a Linear Discriminant Function

White board time!

# Summary of Logistic Regression

- Learns conditional probability distribution  $P(y | \mathbf{x})$
- Local Search
  - begins with initial weight vector. Modifies it iteratively to maximize the log likelihood of the data
- Online or Batch
  - both online and batch variants of the algorithm exist

# Linear Discriminant Analysis

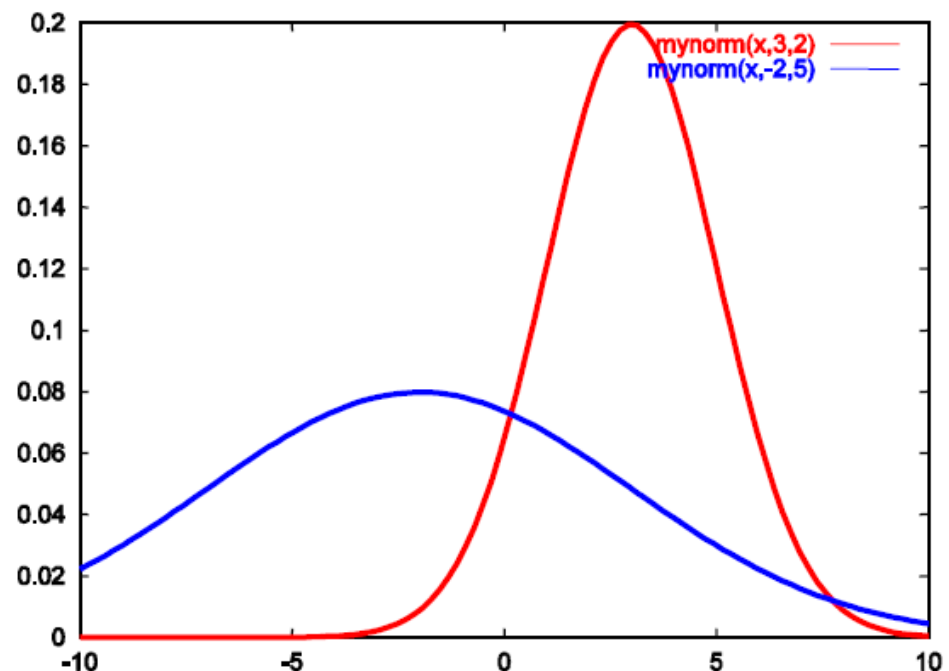
- Learn  $P(\mathbf{x}, y)$ . This is often called as generative approach, because we think of  $P(\mathbf{x}, y)$  as a model of how the data has been generated.
  - For example, if we factor the joint distribution  $P(\mathbf{x}, y) = P(y) P(\mathbf{x} | y)$
  - We can think of  $P(y)$  as generating a value for  $y$  according to the prior. Then we can think of  $P(\mathbf{x} | y)$  as generating a value for  $\mathbf{x}$  given the previously-generated value of  $y$



## LDA (2)

- $P(y)$  is a discrete normal distribution
  - Ex:  $P(y=0) = 0.31$  will generate 31% negative examples and 69% positive examples
- Recall that the univariate gaussian has the formula

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





# Multivariate Gaussian

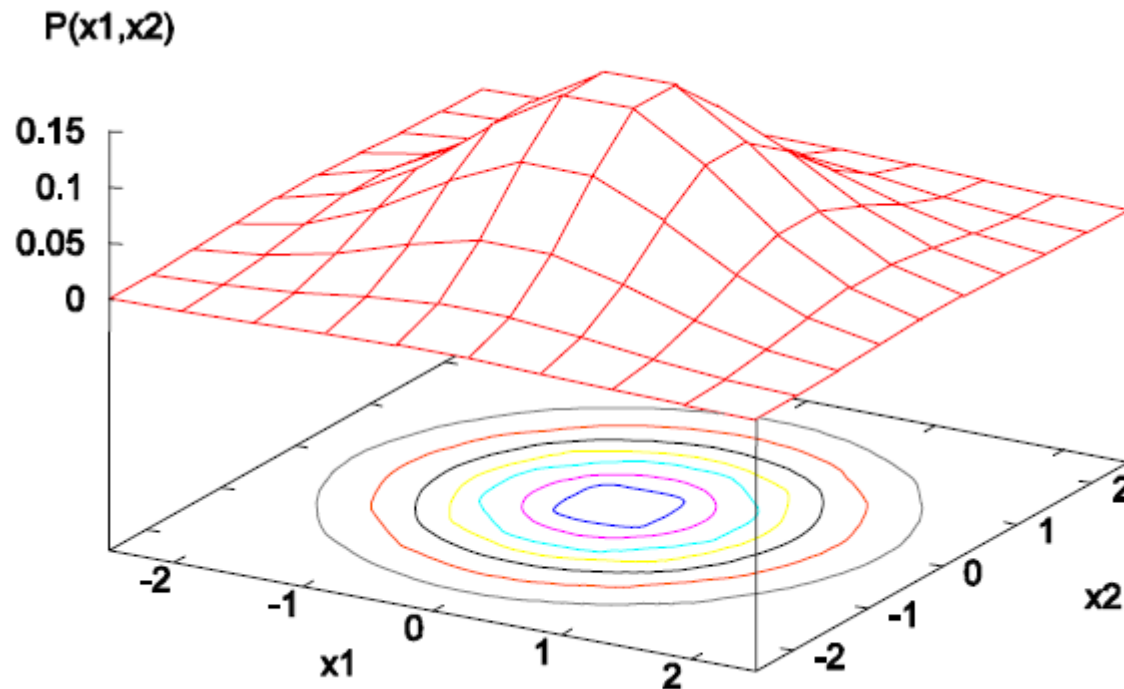
- A 2-dimensional Gaussian is defined by a mean vector  $\mu = (\mu_1, \mu_2)$  and a covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_{1,1}^2 & \sigma_{1,2}^2 \\ \sigma_{2,1}^2 & \sigma_{2,2}^2 \end{bmatrix}$$

where  $\sigma_{i,j}^2 = E[(\mathbf{x}_i - \mu_i)((\mathbf{x}_j - \mu_j))]$  is the variance (if  $i=j$ ) or the covariance if ( $i \neq j$ ).

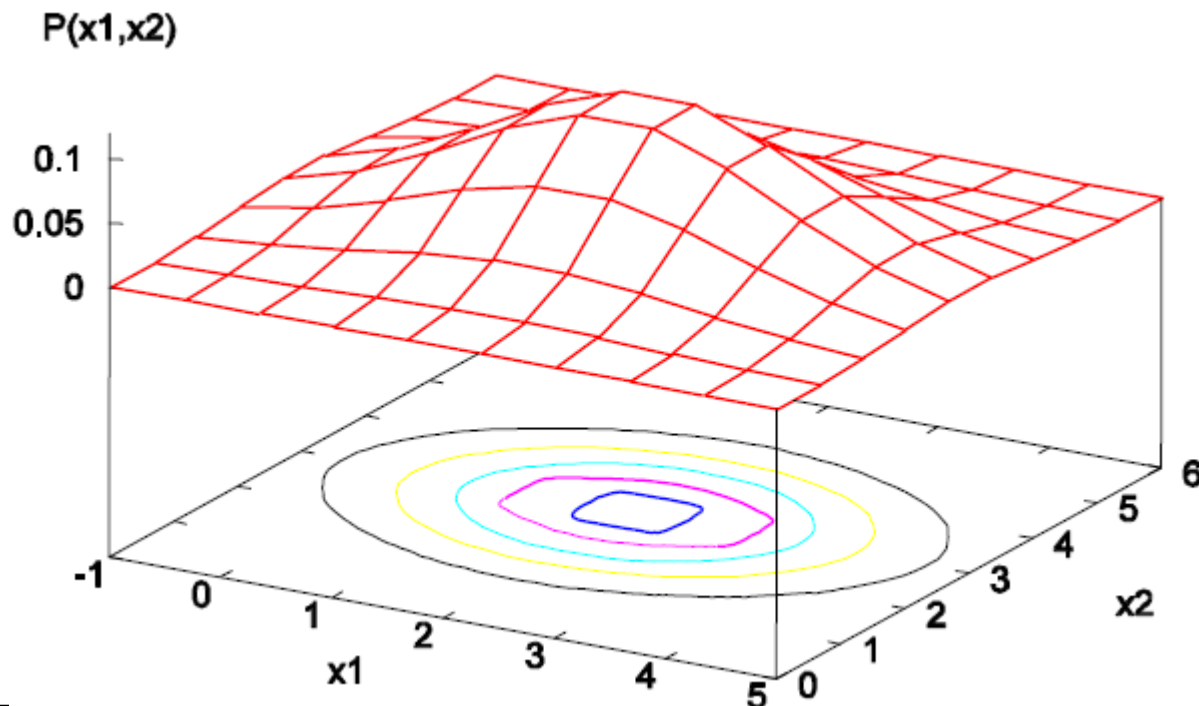
# Multivariate Gaussian(2)

If  $\Sigma$  is the identity matrix, i.e.,  $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  and  $\mu = (0,0)$  then we get the standard normal distribution



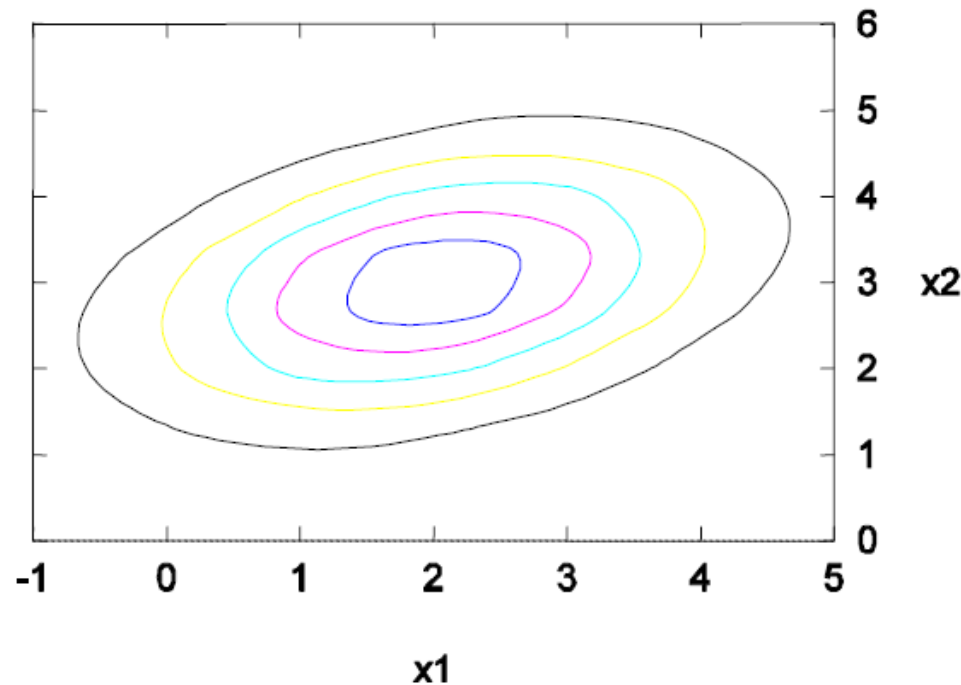
## Multivariate Gaussian (3)

- If  $\Sigma$  is a diagonal matrix, for ex,  $\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$  and  $\mu = (2,3)$ , the two variates  $x_1$  and  $x_2$  are independent random variables. In this case, we obtain



## Multivariate Gaussian (4)

- If  $\Sigma$  is an arbitrary matrix, for ex,  $\Sigma = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 1 \end{bmatrix}$  and  $\mu = (2,3)$ , the two variates  $x_1$  and  $x_2$  are dependent variables. In this case, the lines are tilted to the coordinate axes.



# Estimating a Multivariate Gaussian

- Given a set of  $N$  data points  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  we can compute the maximum likelihood estimate for the multivariate gaussian

$$\hat{\mu} = \frac{1}{N} \sum_i \mathbf{x}_i$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (\mathbf{x}_i - \hat{\mu}) \cdot (\mathbf{x}_i - \hat{\mu})^T$$

- Note that the dot product in the second equation is an outer product. The outer product of two vectors is a matrix

$$x \cdot y^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \cdot [y_1 \ y_2 \ y_3] = \begin{bmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \\ x_3 y_1 & x_3 y_2 & x_3 y_3 \end{bmatrix}$$

- For comparison the usual dot product is written as  $\mathbf{x}^T \cdot \mathbf{y}$

# LDA Model

- LDA assumes that the joint distribution has the form

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

Where each  $\mu_y$  is the mean of a multivariate Gaussian for examples belonging to class  $y$  and  $\Sigma$  is a single covariance matrix shared by all classes.

# Fitting the LDA Model

- It is easy to learn the LDA model in a single pass through the data
  - Let  $\widehat{\pi}_k$  be our estimate of  $P(y=k)$
  - Let  $N_k$  be the number of training examples belonging to class  $k$

$$\hat{\pi}_k = \frac{N_k}{N}$$

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{i; y_i=k} \mathbf{x}_i$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (\mathbf{x}_i - \hat{\mu}_{y_i}) \cdot (\mathbf{x}_i - \hat{\mu}_{y_i})^T$$

- Note that each  $\mathbf{x}_i$  is subtracted from its corresponding  $\hat{\mu}_{y_i}$  prior to taking the outer product. This gives the “pooled” estimate of  $\Sigma$
- It is easy to prove that LDA learns a LTU

# Two Geometric Views of LDA

## View 1: Mahalanobis Distance

- The term  $D_M(x, \mathbf{u})^2 = (x - \mathbf{u})^T \Sigma^{-1} (x - \mathbf{u})$  is known as the (squared) Mahalanobis distance between  $\mathbf{x}$  and  $\mathbf{u}$ . We can think of the matrix  $\Sigma^{-1}$  as a linear distortion of the coordinate system that converts the standard Euclidean distance to Mahalanobis distance
- Note that

$$\log P(\mathbf{x} | y = k) \propto \log \pi_k - \frac{1}{2} [(x - \mu_k)^T \Sigma^{-1} (x - \mu_k)]$$

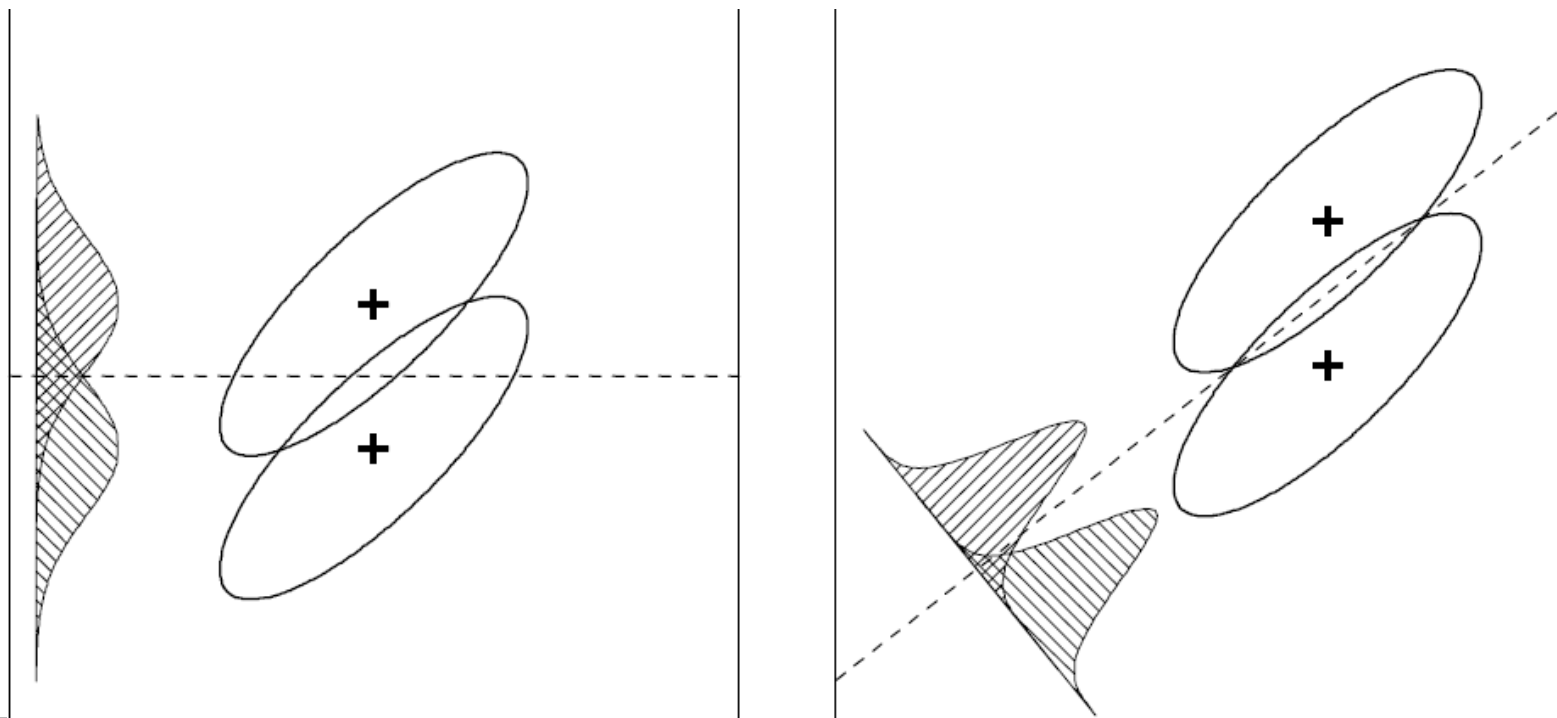
$$\log P(\mathbf{x} | y = k) \propto \log \pi_k - \frac{1}{2} D_M(x, \mu_k)^2$$

- Therefore, we can view LDA as computing  $D_M(x, \mu_0)^2$  and  $D_M(x, \mu_1)^2$  and then classifying  $\mathbf{x}$  according to which mean is closest in Mahalanobis distance corrected by  $\log \pi_k$



## View 2: Most Informative Low-Dimensional Projection

- LDA can be also viewed as finding a hyperplane of dimension  $K-1$  such that  $\mathbf{x}$  and the  $\{\mu_k\}$  are projected down into this hyperplane and then  $\mathbf{x}$  is classified to the nearest  $\mu_k$  using Euclidean distance inside this hyperplane



# Summary of LDA

- Learns the joint probability distribution  $P(\mathbf{x}, y)$
- Direct Computation. The maximum likelihood estimate of  $P(\mathbf{x}, y)$  can be computed from the data without search
- Eager. The classifier is constructed from training examples which can be discarded.
- Batch algorithm. Most implementations are batch algorithm. An online algorithm can be constructed if the matrix can be inverted incrementally.

## Which should we use?

- **Statistical Efficiency:** If the generative model is correct, then LDA gives the highest accuracy, particularly if the data set is small. If the model is correct, LDA requires 30% less data than Logistic Regression in theory
- **Computational Efficiency:** Generative models are the easiest to learn. In our example, we do not need a gradient descent method to compute a LDA
- **Robustness to changing loss functions:** Both generative and conditional methods allow the loss function to be changed at runtime without re-learning. Perceptron requires re-training the classifier if the loss function changes.

# Which should we use

- **Robustness to model assumptions:** The generative models generally have *\*strong\** model assumptions. In turn they perform poorly when these are violated. For instance if  $P(\mathbf{x}|\mathbf{y})$  is very non-Gaussian, then LDA will not work well. Logistic regression is more robust to model assumptions but perceptron is the most robust one.
- **Robustness to missing values and noise:** In many applications, some of the features may be missing or corrupted. Generative models typically provide better ways of handling this.