Linear Models

(some materials from Tom Dietterich & Andrew Ng)

Hsing-Kuo Pao (鮑興國)
National Taiwan University of Science & Technology (Taiwan Tech)

Outline

- Introduction
- Some linear models to discuss
  - Learn a classifier: The LMS algorithm
  - Learn a conditional distribution: Logistic regression
  - Learn a joint distribution: Linear discriminant analysis
- Bayesian interpretation of those models

Why Linear Models?

- Linear methods are easy to implement and analyze
- Rich research materials and techniques for linear modeling
- While initially linear methods may seem overly simple:
  - Many non-linear methods are direct generalizations of linear methods
  - A variety of techniques, such as change of variables, or augmenting our attribute set to include quadratic and cross-product terms (a basis transformation), can be applied such that linear methods (and codes) will be useful
  - E.g., linear SVM and the kernel trick

Three Main Approaches to Machine Learning

- Learn a classifier. The learning algorithm produces a classifier f.
- Learn a conditional probability distribution. The learning algorithm produces a conditional distribution $P(y \mid \mathbf{x})$.
- Learn the joint probability distribution. The learning algorithm produces the joint distribution $P(\mathbf{x}, y)$.

Examples of each of these methods for linear modeling:

- Learn a classifier: The Least Mean Square (LMS) algorithm
- Learn a conditional distribution: Logistic Regression
- Learn a joint distribution: Linear Discriminant Analysis

Linear Regression

— Linear Models

Recall Linear Regression

- Given the training set as

$$\{(\mathbf{x}^{(i)}; y_i) = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}; y_i) : i = 1, \ldots, m\}$$

  we want to find the relation h between $\mathbf{x}$ and y

- Assume $h(\mathbf{x})$ is linear

$$\begin{aligned} h(\mathbf{x}) &= w_0 + w_1 x_1 + w_2 x_2 + \cdots w_n x_n \\ &= \sum_{j=0}^{n} w_j x_j \ (\text{assume } x_0 = 1) \end{aligned}$$

  and we want to minimize the mean squared error

$$E(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - h(\mathbf{x}^{(i)}))^2$$

- We can solve this for the w_j that minimizes the error

Minimizing Mean Squared Error

- Let E be the error function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}^{(i)}))^2$$

- Taking derivatives of E and let them be zero:

$$-\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{m} x_j^{(i)} \cdot (y_i - h(\mathbf{x}^{(i)})) = 0, \quad j = 0, \ldots, n$$

that is,

$$\sum_{i=1}^{m} x_j^{(i)} \cdot y_i = \sum_{k=0}^{n} w_k \sum_{i=1}^{m} x_j^{(i)} \cdot x_k^{(i)}, \quad j = 0, \ldots, n$$

Normal Equations in Matrix Form

- Let us use matrix form to simplify the equations:

$$A = \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \vdots \\ \mathbf{x}^{(m)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

- We want to find $\mathbf{w}$, such that $A\mathbf{w}$ will be as close to $\mathbf{y}$ as possible, i.e., we want to find

$$\begin{aligned} \mathbf{w}^* = \text{argmin}_{\mathbf{w}} E(\mathbf{w}) &= \text{argmin}_{\mathbf{w}} \sum\nolimits_i (\mathbf{y} - A\mathbf{w})_i^2 \\ &= \text{argmin}_{\mathbf{w}} (\mathbf{y} - A\mathbf{w})^T (\mathbf{y} - A\mathbf{w}) \\ &= \text{argmin}_{\mathbf{w}} \ \mathbf{w}^T A^T A\mathbf{w} - 2\mathbf{w}^T A^T \mathbf{y} \end{aligned}$$

- Let gradient vector to be zero

$$\nabla E(\mathbf{w}) = 2A^T A\mathbf{w} - 2A^T \mathbf{y} = 0 \quad \Leftrightarrow \quad \mathbf{w}^* = (A^T A)^{-1} A^T y$$

Linear Regression in Practice

- Computing matrix inverse may take great efforts when the matrix is huge
- Instead, we work on minimizing

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - h(\mathbf{x}^{(i)}))^2$$

- via gradient descent with weight adjust in each iteration by

$$\Delta w_j = \sum_{i=1}^{m} x_j^{(i)} \cdot (y_i - h(\mathbf{x}^{(i)})) = 0, \quad j = 0, \ldots, n$$

- The iterative algorithms are useful when we do not know the solution in closed form!
- Other efficient iterative algorithms: Newton's method, conjugate gradient method

Linear Classification

— Linear Models

Linear Classification

- Let's start with the simplest kind of classifier, a **linear threshold unit (LTU):**

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } w_1 x_1 + \ldots + w_n x_n \geq -w_0 \\ 0 & \text{otherwise} \end{cases}$$

- We'll look at three algorithms, all of which learn linear decision boundaries:
  - Directly learn the LTU: Using Least Mean Square (LMS) algorithm
  - Learn the conditional distribution: Logistic regression
  - Learn the joint distribution: Linear discriminant analysis (LDA)
- Do we need iterative algorithms for those?
- What could be the learning rules then?

Linear Threshold Units

- Examples of things that can be expressed (assuming Boolean features)
  - conjunctions:

$$x_1 \wedge x_3 \wedge x_4 \Rightarrow 1 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 3$$

$$x_1 \wedge \neg x_3 \wedge x_4 \Rightarrow 1 \cdot x_1 + 0 \cdot x_2 + (-1) \cdot x_3 + 1 \cdot x_4 \geq 2$$

  - at-least-m-of-n

$$\text{at-least-2-of } (x_1, x_2, x_4) \Rightarrow 1 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 + 1 \cdot x_4 \geq 2$$

- Examples of things that cannot be expressed:
  - non-trivial disjunctions:

$$(x_1 \wedge x_3) + (x_3 \wedge x_4)$$

  - exclusive-or

$$(x_1 \wedge \neg x_2) + (\neg x_1 \wedge x_2)$$

Canonical Representation

- LTU in canonical representation
- Given a training example of the form $\{(x_1, x_2, x_3, x_4; y)\}$
  - transform to $\{(1, x_1, x_2, x_3, x_4; y)\}$
  - the parameter vector will then be $\mathbf{w} = (w_0, w_1, \ldots, w_4)$
- The unthresholded hypothesis is $u(\mathbf{x}, \mathbf{w}) = \mathbf{w} \cdot \mathbf{x}$
- The (thresholded) hypothesis is written

$$h(\mathbf{x}) = \text{stp}(u(\mathbf{x}, \mathbf{w})) = (\text{sgn}(u(\mathbf{x}, \mathbf{w})) + 1)/2$$

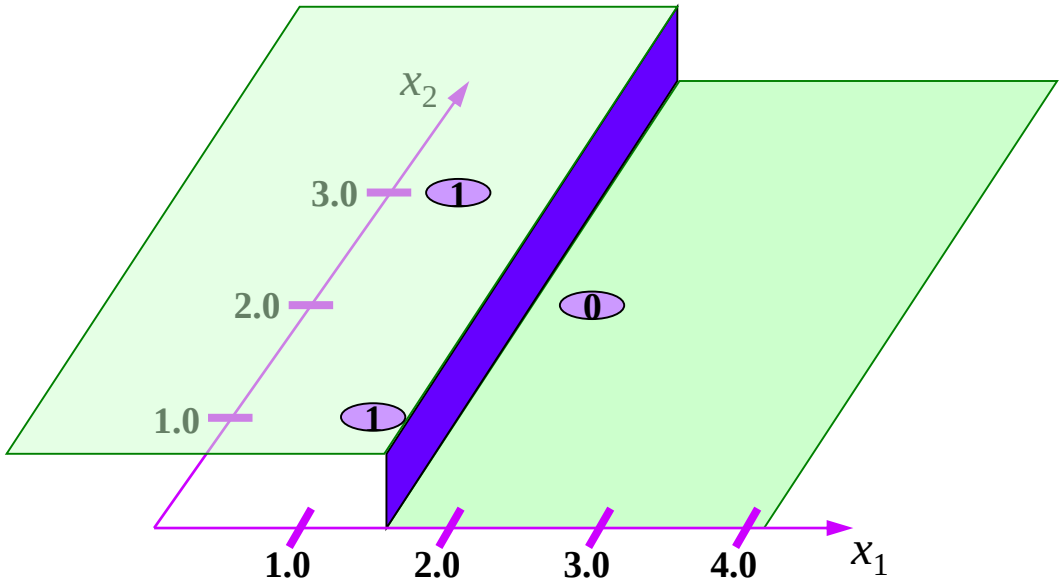
- This hypothesis space is $\{(w_0, w_1, \ldots, w_4) \in \mathbb{R}^5\}$

$$\text{sgn}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{if } u < 0 \end{cases} \qquad \text{stp}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ 0 & \text{if } u < 0 \end{cases}$$

- Our goal is to find $\mathbf{w}$, the parameters of the LTU

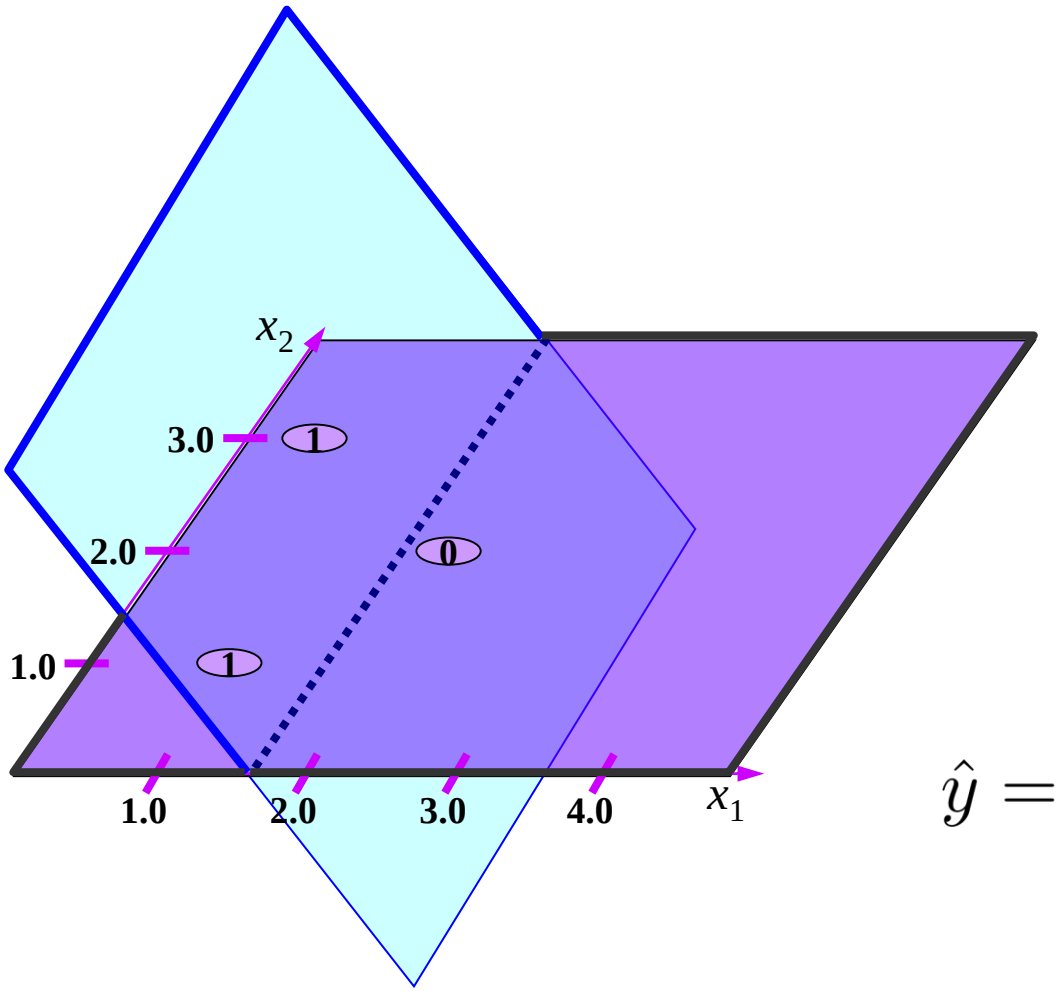
Geometrical View

- Learning a piecewise constant function
- Consider three training examples:
- $\mathcal{D} = \{(1.0, 1.0; 1), (0.5, 3.0; 1), 2.0, 2.0; 0)\}$
- We want a classifier that looks like the following:



The Unthresholded Discriminant Function is a Hyperplane

- The equation $u(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ is a plane.



$$\hat{y} = \begin{cases} 1 & \text{if } u(\mathbf{x}) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Machine Learning as Optimization

- When we learning a classifier directly, the learning problem is:
  - **Given:** A set of m training examples $\{(\mathbf{x}^{(i)}, y_i)\}, i = 1, \ldots, m$ and a loss function L
  - **Find:** The weight vector $\mathbf{w}$ that minimizes the expected loss of the training data $J(\mathbf{w})$:

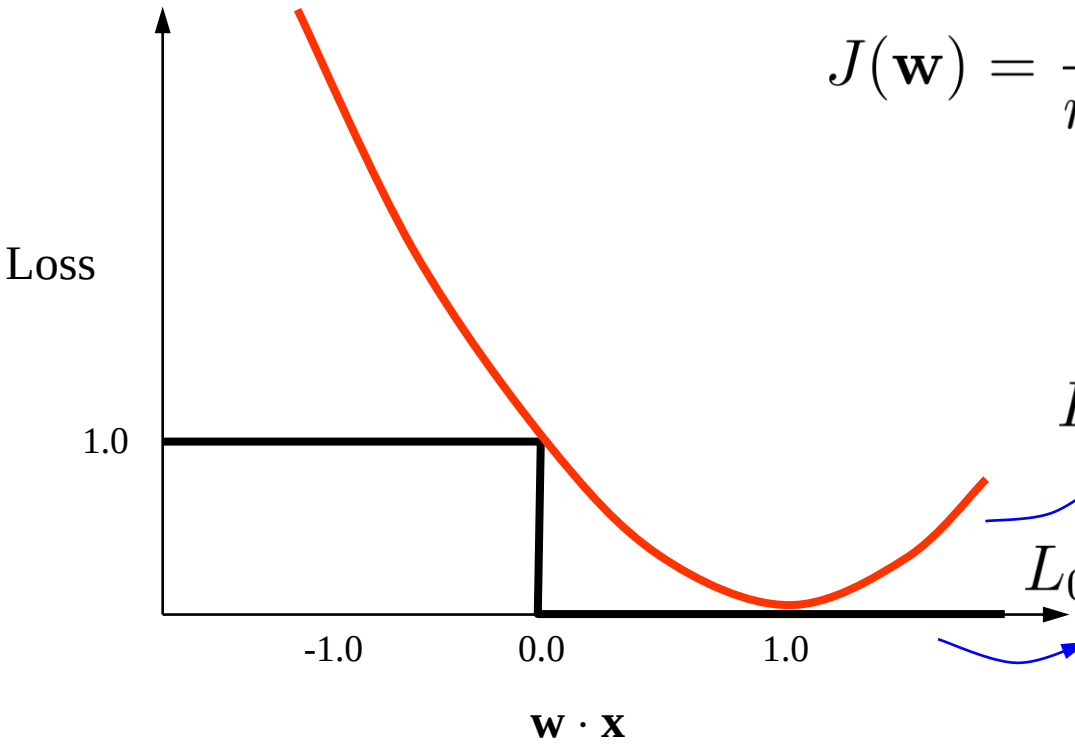
$$J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} L(\mathrm{stp}(\mathbf{w} \cdot \mathbf{x}^{(i)}), y_i)$$

- In general, machine learning algorithms apply some optimization strategy to find a good hypothesis
- What loss function is appropriate?
- What (algorithm) could be used in practice for optimization?

0/1 Loss

- In this case, J is piecewise constant which makes this a difficult problem
- Instead, replace original objective function with a smooth, differentiable function, such as MSE:

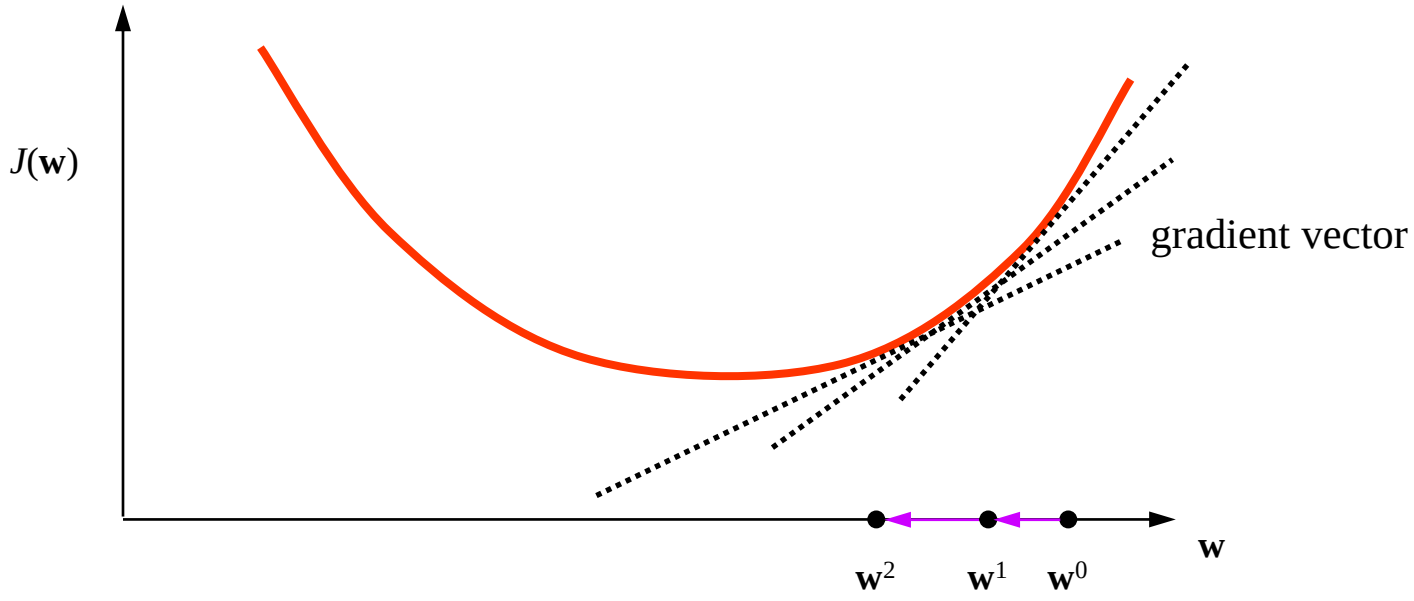
$$J(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_i - y_i)^2$$



$$L_{\text{smooth}}(\mathbf{x}, 1) = (\mathbf{w}\cdot\mathbf{x} - 1)^2$$

$$L_{01}(\mathbf{x}, 1) = (\text{stp}(\mathbf{w}\cdot\mathbf{x}) - 1)^2$$

Minimizing J by Gradient Descent



- Start with initial weight vector w_0
- Compute the gradient $\nabla J(\mathbf{w}) = \left(\frac{\partial J(\mathbf{w})}{\partial w_0}, \frac{\partial J(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_n} \right)$
- Compute $\mathbf{w} = \mathbf{w} - \alpha \nabla J(\mathbf{w})$
- Repeat until convergence

Computing the Gradient

- First, let's work out the partial derivative in the case where we have a single training example $(\mathbf{x}, y)$:

$$\begin{aligned}
\frac{\partial}{\partial w_j} J(\mathbf{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2} (\mathbf{w} \cdot \mathbf{x} - y)^2 \\
&= 2 \cdot \frac{1}{2} (\mathbf{w} \cdot \mathbf{x} - y) \frac{\partial}{\partial w_j} (\mathbf{w} \cdot \mathbf{x} - y) \\
&= (\mathbf{w} \cdot \mathbf{x} - y) \frac{\partial}{\partial w_j} \left(\sum_{k=0}^{n} w_k x_k - y \right) \\
&= (\mathbf{w} \cdot \mathbf{x} - y) x_j
\end{aligned}$$

- This gives the update rule: $w_j = w_j + \alpha (y - \mathbf{w} \cdot \mathbf{x}) x_j \quad \forall j$
- This is called the LMS update rule. Also known as the Widrow-Hoff learning rule.
- Intuitive: magnitude of the update is proportional to the error.

Batch Gradient Descent

- Given a set of m training examples

$$\{(\mathbf{x}^{(i)}, y_i)\}, i = 1, \ldots, m$$

- Repeat until convergence

$$w_j = w_j + \alpha \sum_{i=1}^{m} (y_i - \mathbf{w} \cdot \mathbf{x}^{(i)}) x_j^{(i)} \quad \forall j$$

- This method looks at every example in the entire training set on every step
- While in general gradient descent can get stuck in local minima, in this case the loss function is convex, so there is a single, global optima
- Gradient descent is guaranteed to find the optimal solution (as long as α is not too large)

Incremental Gradient Descent

(also called Stochastic Gradient Descent)

- Repeat until convergence

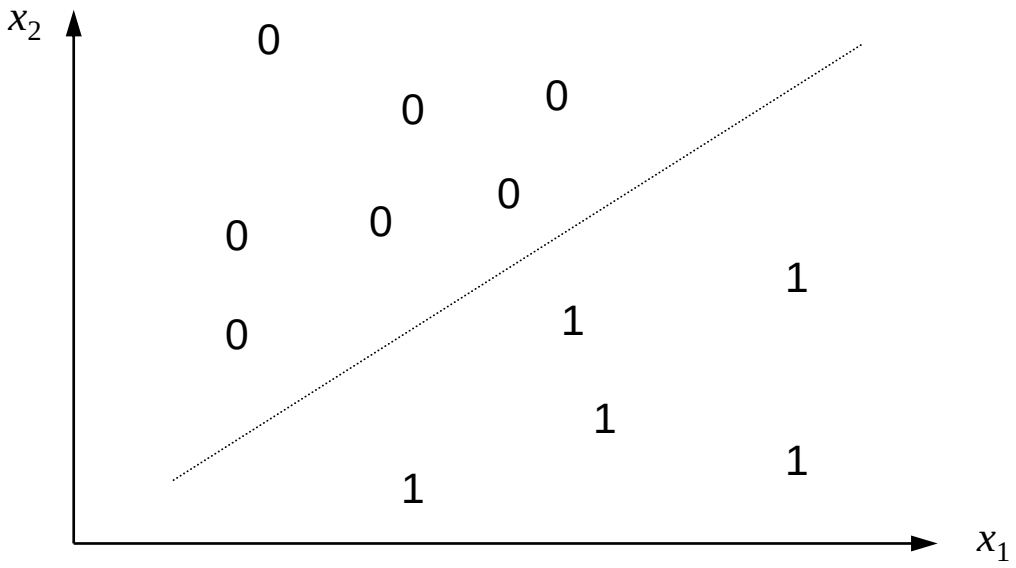
  for i = 1, …, m

$$w_j = w_j + \alpha(y_i - \mathbf{w} \cdot \mathbf{x}^{(i)})x_j^{(i)} \quad \forall j$$

- Whereas batch gradient descent has to scan through the entire training set before taking a step, incremental gradient descent begins making changes right away
- Often $\mathbf{w}$ gets close to the minimum faster than batch gradient descent
- However it may oscillate around the minimum
- One way to avoid this is, instead of using fixed α, slowly let the learning rate go to zero
- SGD is often preferred to BGD, especially when the training set is large
- There are other more sophisticated algorithms such as Newton's method and Conjugate Gradient Descent that choose the step size automatically and converge faster

Decision Boundaries

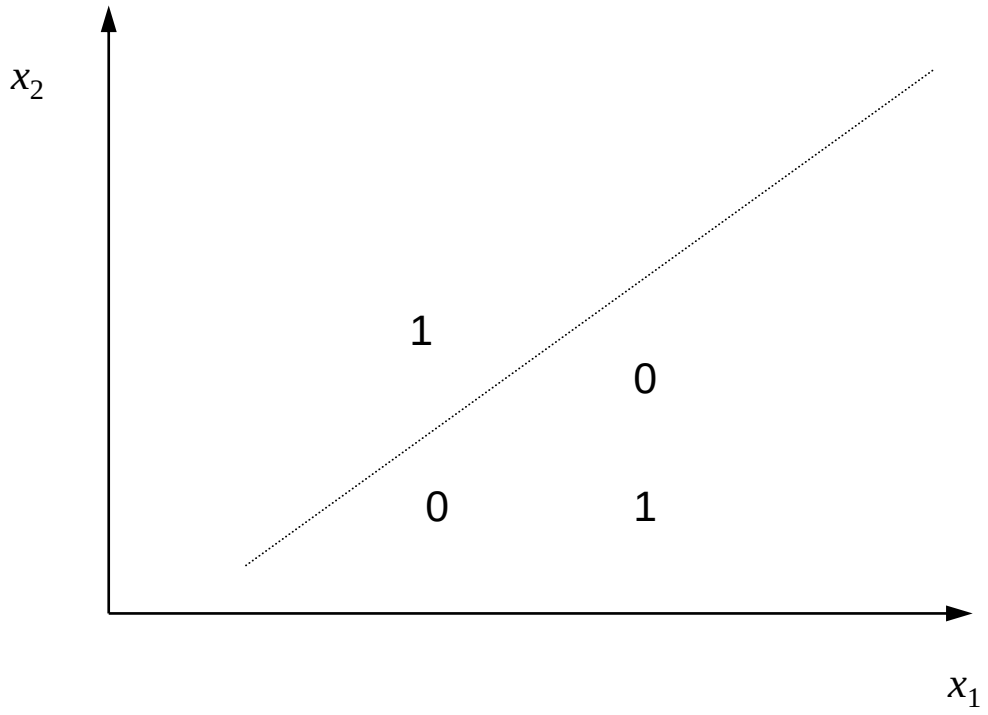
- A classifier can be viewed as partitioning the input space or feature space $\mathbf{x}$ into decision regions



- A linear threshold unit always produces a linear decision boundary. A set of points that can be separated by a linear decision boundary is linearly separable

Non-linearly separable example

- Exclusive-OR is not linearly separable



- How to deal with non-linearly separable case?

Extending LMS to more > 2 classes

- If we have $K > 2$ classes, we can learn a separate LTU for each class. Let $\mathbf{w}_k$ be the weight vector for class k. We train it by treating the examples from class $y = k$ as the positive examples, and treating examples from all the other classes as negative examples. We classify a new point $\mathbf{x}$ according to

$$\hat{y} = \text{argmax}_k \mathbf{w}_k \cdot \mathbf{x}$$

- This works for some problems, but not always. Here is an example where it doesn't work, but LDA does:

LMS for LTUs Summary

- Search Procedure
  - Directly Learns a Classifier
  - Local search: Begins with initial weight vector. Modifies it iteratively to minimize an error function. The error function is a smooth approximation to the misclassification error
- Timing
  - Eager: The classifier is constructed from the training examples. Examples can then be discarded
- Online or batch:
  - Both batch and incremental versions of gradient descent can be used

Issues in LMS for LTUs

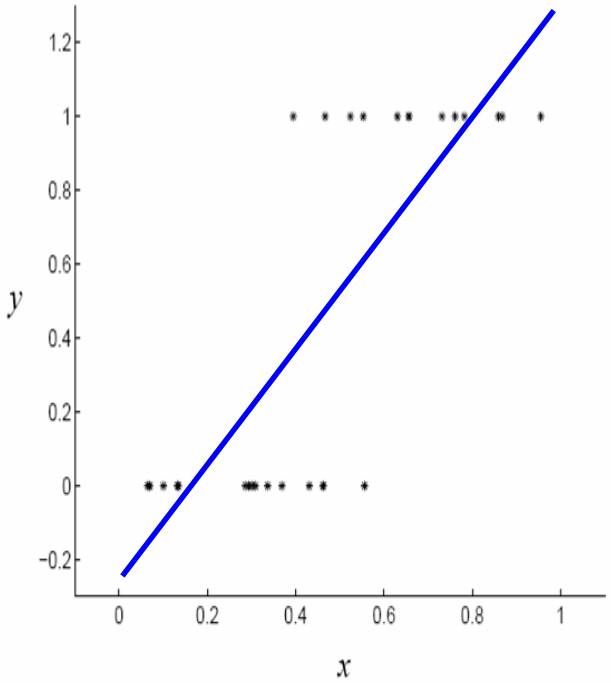
- Classification worked on unthresholded error function for differentiability in gradient computation:
  - No difference between classification and regression problems
  - Error function can't truly reflect the gap between the true label and the predicted label

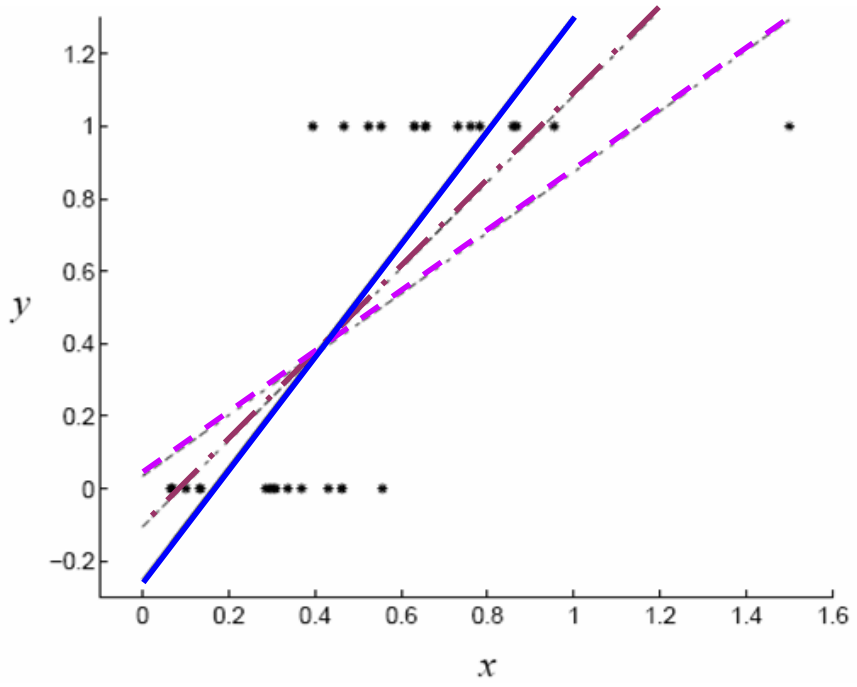
  ⇒ Changing the model with new data when no need to do so

  ⇒ Learning rule updates weights inappropriately
- Can't deal with non-linearly separate cases

Issues in LMS for LTUs (cont'd)

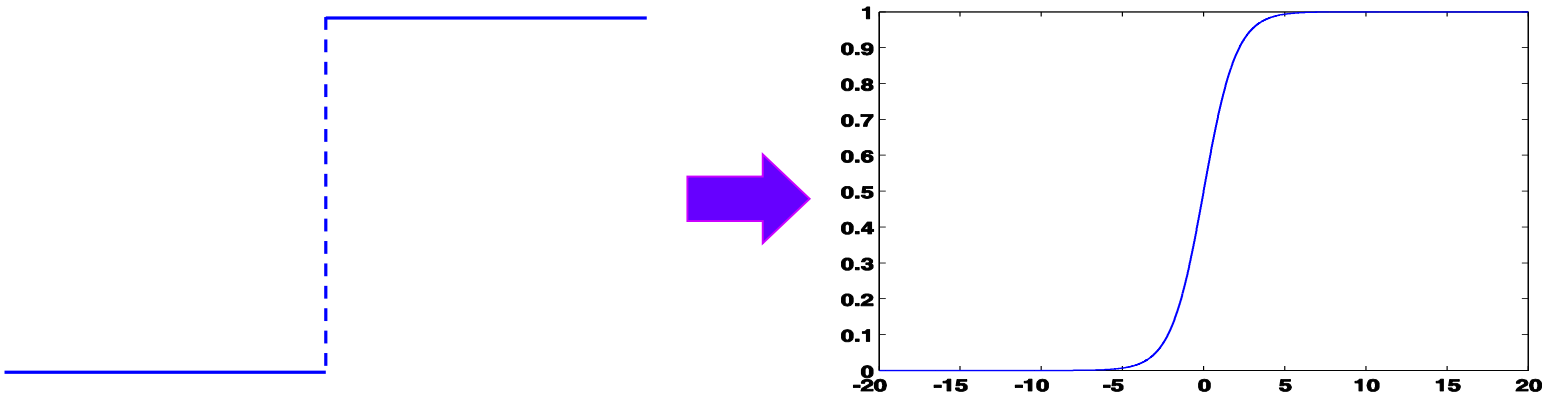
- Adding one training example can change the regression line dramatically even it is correctly classified by the original regression line!
- In the right, the solid line is the same fit as shown in the left, the dash-dot line is the fit to the data with one additional point at $(1.5, 1)$, and the dashed line is the fit to the data with five additional points at $(1.5, 1)$





A Refinement for Linear Classification

- Linear classification uses thresholded $\mathrm{stp}(u(\mathbf{x}, \mathbf{w})) = \mathrm{stp}(\mathbf{w} \cdot \mathbf{x})$, while linear regression uses unthresholded $u(\mathbf{x}, \mathbf{w}) = \mathbf{w} \cdot \mathbf{x}$
- Linear regression assumes noise Gaussian distributed which is not reasonable in the case of linear classification
- As for linear classification, we need some appropriate assumption
  - We can make a function close to a step function which will give the similar performance and can be differentiable at the same time



Logistic Regression

— Linear Models

Logistic Regression

- In logistic regression, we learn the conditional distribution

$$P(y \mid \mathbf{x})$$

- Let $h_y(\mathbf{x}; \mathbf{w})$ be our estimate of $P(y \mid \mathbf{x})$, where $\mathbf{w}$ is a vector of adjustable parameters.
- Assume there are two classes, $y = 0$ and $y = 1$ and

$$\begin{aligned} h(\mathbf{x}) &= h_1(\mathbf{x}; \mathbf{w}) &= \frac{\exp(\mathbf{w} \cdot \mathbf{x})}{1 + \exp(\mathbf{w} \cdot \mathbf{x})} \\ 1 - h(\mathbf{x}) &= h_0(\mathbf{x}; \mathbf{w}) &= 1 - p_1(\mathbf{x}; \mathbf{w}) \end{aligned}$$

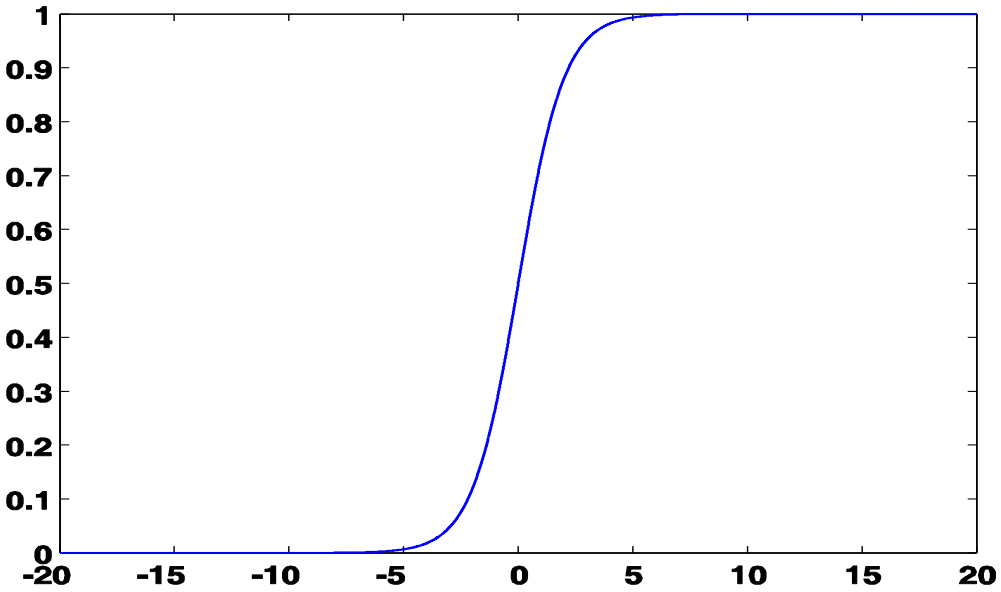
- This is equivalent to

$$\log \frac{h_1(\mathbf{x}; \mathbf{w})}{h_0(\mathbf{x}; \mathbf{w})} = \mathbf{w} \cdot \mathbf{x}$$

in other words, the log odds of class 1 is a linear function of $\mathbf{x}$

Why the exp function?

- One reason: transforms a linear function in the range (-∞,+∞) to be positive and sum to 1 so that it can represent a probability



$$\frac{\exp(\mathbf{w}\cdot\mathbf{x})}{1+\exp(\mathbf{w}\cdot\mathbf{x})}$$

Constructing a Learning Algorithm

- Fitting a conditional probability distribution, no longer seeking to minimize the expected loss on the training data
- Instead, finding the probability distribution h that is most likely, given the data
- Let D be the training sample. Our goal is to find h to maximize $P(h_\theta \mid D)$

$$
\begin{aligned}
\text{argmax}_{h_\theta} P(h_\theta \mid D) &= \text{argmax}_{h_\theta} (P(D \mid h_\theta) P(h_\theta)) / P(D) \\
&= \text{argmax}_{h_\theta} P(D \mid h_\theta) P(h_\theta) \\
&= \text{argmax}_{h_\theta} P(D \mid h_\theta) \\
&= \text{argmax}_{h_\theta} \log P(D \mid h_\theta)
\end{aligned}
$$

Constructing a Learning Algorithm (cont'd)

- The **likelihood function** views $P(D \mid h_\theta)$ as a function of the parameters in the model. In this case, our parameters are the weights, $\mathbf{w}$

$$L(\mathbf{w}; S) = P(D \mid h_\theta)$$

- The log likelihood is a commonly used objective function for learning algorithms, denoted by $\ell(\mathbf{w}; S)$

- The $\mathbf{w}$ is the **maximum likelihood estimator**

Computing the Likelihood

- In this framework, we assume that each training example $(\mathbf{x}^{(i)}, y_i)$ is drawn from the same underlying (but unknown) probability distribution $P(\mathbf{x}, y)$. This means that the log likelihood of D is the sum of the log likelihoods of the individual training examples:

$$\begin{aligned}
\log P(D \mid h) &= \log \prod_i P((\mathbf{x}^{(i)}, y_i) \mid h) \\
&= \sum_i \log P((\mathbf{x}^{(i)}, y_i) \mid h) \\
\operatorname{argmax}_h \log P(D \mid h) &= \operatorname{argmax}_h \sum_i \log P((\mathbf{x}^{(i)}, y_i) \mid h) \\
&= \operatorname{argmax}_h \sum_i \log P(y_i \mid \mathbf{x}^{(i)}, h) P(\mathbf{x}^{(i)} \mid h) \\
&= \operatorname{argmax}_h \sum_i \log P(y_i \mid \mathbf{x}^{(i)}, h) P(\mathbf{x}^{(i)}) \\
&= \operatorname{argmax}_h \sum_i \log P(y_i \mid \mathbf{x}^{(i)}, h)
\end{aligned}$$

- Hence, the log likelihood of D is the sum of the log conditional likelihood of the individual data points

Log Likelihood for Conditional Probability Estimators

- We can express the log likelihood in a compact form called the **cross-entropy**
- Taking an example $(\mathbf{x}^{(i)}, y_i)$

  if $y_i = 0$, the log likelihood is $\log(1 - h(\mathbf{x}))$
  if $y_i = 1$, the log likelihood is $\log h(\mathbf{x})$

- These two are mutually exclusive, so we can combine them to get:

$$\ell(\mathbf{w}; \mathbf{x}, y) = \log P(y_i \mid \mathbf{x}^{(i)}, \mathbf{w}) = (1 - y_i) \log(1 - h(\mathbf{x})) + y_i \log h(\mathbf{x})$$

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

$$J(\mathbf{w}) = \ell(\mathbf{w}; \mathbf{x}_i, y)$$

Fitting Logistic Regression by Gradient Ascent

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial w_j} &= \sum_i \frac{\partial}{\partial w_j} \ell(\mathbf{w}; y_i, \mathbf{x}^{(i)}) \\
\frac{\partial}{\partial w_j} \ell(\mathbf{w}; y_i, \mathbf{x}^{(i)}) &= \frac{\partial}{\partial w_j} ((1 - y_i) \log(1 - h(\mathbf{x}^{(i)})) + y_i \log h(\mathbf{x}^{(i)})) \\
&= (1 - y_i) \frac{1}{1 - h(\mathbf{x}^{(i)})} \left(-\frac{\partial h(\mathbf{x}^{(i)})}{\partial w_j} \right) + y_i \frac{1}{h(\mathbf{x}^{(i)})} \left(\frac{\partial h(\mathbf{x}^{(i)})}{\partial w_j} \right) \\
&= \left[\frac{y_i}{h(\mathbf{x}^{(i)})} - \frac{1 - y_i}{1 - h(\mathbf{x}^{(i)})} \right] \left(\frac{\partial h(\mathbf{x}^{(i)})}{\partial w_j} \right) \\
&= \left[\frac{y_i (1 - h(\mathbf{x}^{(i)})) - (1 - y_i) h(\mathbf{x}^{(i)})}{h(\mathbf{x}^{(i)})(1 - h(\mathbf{x}^{(i)}))} \right] \left(\frac{\partial h(\mathbf{x}^{(i)})}{\partial w_j} \right) \\
&= \left[\frac{y_i - h(\mathbf{x}^{(i)})}{h(\mathbf{x}^{(i)})(1 - h(\mathbf{x}^{(i)}))} \right] \left(\frac{\partial h(\mathbf{x}_i)}{\partial w_j} \right)
\end{aligned}
$$

Gradient Computation

- Another way of writing the logistic regression function is:

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

- So we get:

$$\begin{aligned}
\frac{\partial h(\mathbf{x}_i)}{\partial w_j} &= \frac{1}{(1 + e^{-\mathbf{w} \cdot \mathbf{x}_i})^2} \frac{\partial}{\partial w_j} (1 + e^{-\mathbf{w} \cdot \mathbf{x}_i}) \\
&= \frac{1}{(1 + e^{-\mathbf{w} \cdot \mathbf{x}_i})^2} e^{-\mathbf{w} \cdot \mathbf{x}_i} \frac{\partial}{\partial w_j} (-\mathbf{w} \cdot \mathbf{x}_i) \\
&= \frac{1}{(1 + e^{-\mathbf{w} \cdot \mathbf{x}_i})^2} e^{-\mathbf{w} \cdot \mathbf{x}_i} (-x_{ij}) \\
&= h(\mathbf{x}_i)(1 - h(\mathbf{x}_i)) x_{ij}
\end{aligned}$$

Gradient Computation (cont'd)

- The gradient of the log-likelihood for a single point is:

$$\begin{aligned}
\frac{\partial}{\partial w_j}\ell(\mathbf{w};\mathbf{x}^{(i)},y_i) &= \left[\frac{y_i - h(\mathbf{x}^{(i)})}{h(\mathbf{x}^{(i)})(1-h(\mathbf{x}^{(i)}))}\right]\left(\frac{\partial h(\mathbf{x}^{(i)})}{\partial w_j}\right) \\
&= \left[\frac{y_i - h(\mathbf{x}^{(i)})}{h(\mathbf{x}^{(i)})(1-h(\mathbf{x}^{(i)}))}\right] h(\mathbf{x}^{(i)})(1-h(\mathbf{x}^{(i)}))x_j^{(i)} \\
&= (y_i - h(\mathbf{x}^{(i)}))x_j^{(i)}
\end{aligned}$$

- The overall gradient is:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_i (y_i - h(\mathbf{x}^{(i)}))x_j^{(i)}$$

Batch Gradient Ascent

- Given a set of training examples

$$\{(\mathbf{x}^{(i)}; y_i) : i = 1, \ldots, m\}$$

- Repeat until convergence

$$w_j = w_j + \alpha \sum_{i=1}^{m} \left(y_i - \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}^{(i)}}} \right) x_j^{(i)} \quad \forall j$$

Logistic Regression for $K > 2$

- To handle $K > 2$ classes, we make one class the "reference" class. Suppose it is class K. Then we represent each of the other classes as a logistic function of the odds of class k versus class K:

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

- The conditional probability for class $k \neq K$ is

$$P(y=k \mid \mathbf{x}) = \frac{e^{\mathbf{w}_k \cdot \mathbf{x}}}{1 + \sum_{j=1}^{K-1} e^{\mathbf{w}_j \cdot \mathbf{x}}}$$

and for class $k = K$: $P(y=K \mid \mathbf{x}) = \dfrac{1}{1 + \sum_{j=1}^{K-1} e^{\mathbf{w}_j \cdot \mathbf{x}}}$

Summary of Logistic Regression

- Learns the Conditional Probability Distribution $P(y \mid \mathbf{x})$
- Local Search. Begins with initial weight vector. Modifies it iteratively to maximize an objective function. The objective function is the **log likelihood** of the data – so the algorithm seeks the probability distribution $P(y \mid \mathbf{x})$ that is most likely given the data.
- Online or Batch

A Summary of Learning Rules so far

- Perceptron learning rule

$$w_j = w_j + \alpha(y_i - s(\mathbf{w} \cdot \mathbf{x}^{(i)}))x_j^{(i)} \quad \forall j$$

- Gradient learning rule (aka LMS update rule, the Widrow-Hoff learning rule)

$$w_j = w_j + \alpha(y_i - \mathbf{w} \cdot \mathbf{x}^{(i)})x_j^{(i)} \quad \forall j$$

- Logistic regression learning rule

$$w_j = w_j + \alpha(y_i - \sigma(\mathbf{w} \cdot \mathbf{x}^{(i)}))x_j^{(i)} \quad \forall j$$

- Gradient learning rule with sigmoid function

$$w_j = w_j + \alpha\sigma(1-\sigma)(y_i - \sigma(\mathbf{w} \cdot \mathbf{x}^{(i)}))x_j^{(i)} \quad \forall j$$

s: step function, σ: sigmoid function

- All above has a batch mode and incremental mode when put into practice

A Summary of Learning Rules so far (cont'd)

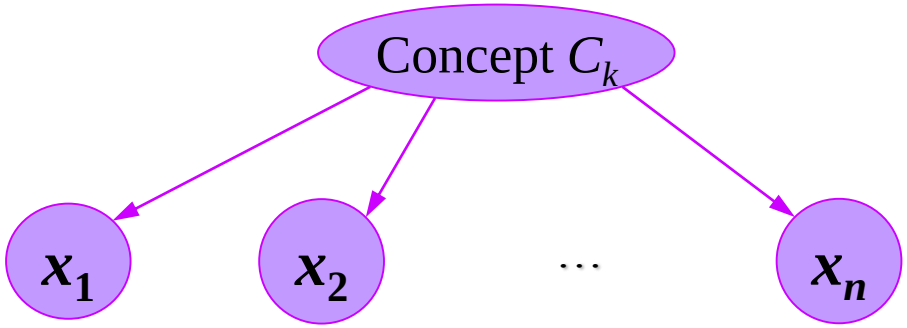
- Perceptron learning rule:
  - No theoretical support
  - Loop forever for non-linearly separable case
- Gradient learning rule (aka LMS update rule, the Widrow-Hoff learning rule)
  - Inappropriate error estimation and not suitable for classification problems
- Logistic regression learning rule
  - Learning (posterior) probability instead of a classifier function
- Gradient learning rule with sigmoid function
  - Potentially useful for non-linearly separable case when putting multiple layers together!

Linear Discriminant Analysis

— Linear Models

Gaussian Naïve Bayes classifier

- The same assumption, but for a set of continuous attributes
  - $x_1, x_2, \ldots, x_n$ are independent given C_k
    i.e., $P(x_1, \ldots, x_n \mid C_k) = \prod_j P(x_j \mid C_k)$
  - Each $P(x_j \mid C_k)$ is governed by a Gaussian distribution $\mathcal{N}(\mu_{jk}, \sigma_j)$
  - The standard deviation σ_j vary from attribute to attribute, but not depending on C_k
- The Naïve Bayes classifier gives the form actually equivalent to a special case of logistic regression



Gaussian Naïve Bayes Classifier & Logistic Regression

- Assuming only binary classification $(y = 0, 1)$ for simplicity
- $P(y = 1) = \pi, P(y = 0) = 1 - \pi$

$$
\begin{aligned}
P(y=1 \mid \mathbf{x}) &= \frac{P(y=1)P(\mathbf{x} \mid y=1)}{P(y=1)P(\mathbf{x} \mid y=1) + P(y=0)P(\mathbf{x} \mid y=0)} \\
&= \frac{1}{1 + \frac{P(y=0)P(\mathbf{x} \mid y=0)}{P(y=1)P(\mathbf{x} \mid y=1)}} \\
&= \frac{1}{1 + \exp\left[\ln \frac{P(y=0)P(\mathbf{x} \mid y=0)}{P(y=1)P(\mathbf{x} \mid y=1)}\right]} \\
&= \frac{1}{1 + \exp\left[\ln \frac{P(y=0)}{P(y=1)} + \sum_j \ln \frac{P(x_j \mid y=0)}{P(x_j \mid y=1)}\right]} \\
&= \frac{1}{1 + \exp\left[\ln \frac{1-\pi}{\pi} + \sum_j \ln \frac{P(x_j \mid y=0)}{P(x_j \mid y=1)}\right]}
\end{aligned}
$$

Bayes rule

Naïve Bayes

Gaussian Naïve Bayes Classifier & Logistic Regression (cont'd)

Gaussian assumption

$$
\begin{aligned}
\sum_j \ln \frac{P(x_j \mid y=0)}{P(x_j \mid y=1)} &= \sum_j \ln \frac{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(x_j-\mu_{j0})^2}{2\sigma_j^2}\right)}{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(x_j-\mu_{j1})^2}{2\sigma_j^2}\right)} \\
&= \sum_j \ln \exp\left(\frac{(x_j-\mu_{j1})^2-(x_j-\mu_{j0})^2}{2\sigma_j^2}\right) \\
&= \sum_j \frac{(x_j-\mu_{j1})^2-(x_j-\mu_{j0})^2}{2\sigma_j^2} \\
&= \sum_j \frac{(x_j^2-2x_j\mu_{j1}+\mu_{j1}^2)-(x_j^2-2x_j\mu_{j0}+\mu_{j0}^2)}{2\sigma_j^2} \\
&= \sum_j \frac{2x_j(\mu_{j0}-\mu_{j1})+\mu_{j1}^2-\mu_{j0}^2}{2\sigma_j^2} \\
&= \sum_j \left(\frac{\mu_{j0}-\mu_{j1}}{\sigma_j^2}x_j + \frac{\mu_{j1}^2-\mu_{j0}^2}{2\sigma_j^2}\right)
\end{aligned}
$$

Gaussian Naïve Bayes Classifier & Logistic Regression (cont'd)

$$
\begin{aligned}
P(y=1\,|\,\mathbf{x}) &= \frac{1}{1+\exp\left[\ln\frac{1-\pi}{\pi}+\sum_j\left(\frac{\mu_{j0}-\mu_{j1}}{\sigma_j^2}x_j+\frac{\mu_{j1}^2-\mu_{j0}^2}{2\sigma_j^2}\right)\right]} \\
&= \frac{1}{1+\exp\left(w_0+\sum_{j=1}^n w_j x_j\right)} \\
\text{where}\quad w_j &= \frac{\mu_{j0}-\mu_{j1}}{\sigma_j^2} \\
\text{and}\quad w_0 &= \ln\frac{1-\pi}{\pi}+\sum_j\frac{\mu_{j1}^2-\mu_{j0}^2}{2\sigma_j^2} \\
\text{Also, we have}\quad P(y=0\,|\,x) &= \frac{\exp\left(w_0+\sum_{j=1}^n w_j x_j\right)}{1+\exp\left(w_0+\sum_{j=1}^n w_j x_j\right)}
\end{aligned}
$$

- That is, Gaussian naïve Bayes has the form of logistic regression, with a special assignment of w_j

Linear Discriminant Analysis

- Gaussian Naïve Bayes is a special case of a Bayesian approach called Linear Discriminant Analysis (LDA)
- In LDA, we learn the class-conditioned distribution

$$P(\mathbf{x} \mid y)$$

- We assume $P(\mathbf{x} \mid y)$ is distributed according to a multivariate normal distribution and $P(y)$ is Bernoulli distributed
- Discriminant analysis does not need to be linear in general, even assuming the class-conditioned densities are Gaussian distributed

Putting it all together in LDA

- Also called Gaussian Discriminant Analysis
- Here
  - $y \sim \text{Bernoulli}(p)$
  - $\mathbf{x} \mid y = 0 \sim \mathcal{N}(\mu_0, \Sigma)$
  - $\mathbf{x} \mid y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$

- writing this out, we get:

$$p(y;\pi) = \pi^y (1-\pi)^{1-y}$$

$$p(\mathbf{x} \mid y = 0) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right]$$

$$p(\mathbf{x} \mid y = 1) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right]$$

Computing Posterior Probability

$$
\begin{aligned}
p(y=1|\mathbf{x},\theta) &= \frac{p(\mathbf{x},y=1|\theta)}{p(\mathbf{x},y=1|\theta)+p(\mathbf{x},y=0|\theta)} \\
&= \frac{\pi \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1)\right\}}{\pi \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1)\right\}+(1-\pi)\exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_0)^T\Sigma^{-1}(\mathbf{x}-\mu_0)\right\}} \\
&= \frac{1}{1+\exp\left\{-\log\frac{\pi}{1-\pi}+\frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1)-\frac{1}{2}(\mathbf{x}-\mu_0)^T\Sigma^{-1}(\mathbf{x}-\mu_0)\right\}} \\
&= \frac{1}{1+\exp\left\{-(\mu_1-\mu_0)^T\Sigma^{-1}\mathbf{x}+\frac{1}{2}(\mu_1-\mu_0)^T\Sigma^{-1}(\mu_1+\mu_0)-\log\frac{\pi}{1-\pi}\right\}} \\
&= \frac{1}{1+\exp\{-\beta^T\mathbf{x}-\gamma\}}
\end{aligned}
$$

- Again, it has the form of logistic regression!

More Comments

- When the covariance matrix is the same in two classes, the quadratic term $\mathbf{x}^T\Sigma^{-1}\mathbf{x}$ term in the numerator and denominator of the posterior probability are cancelled

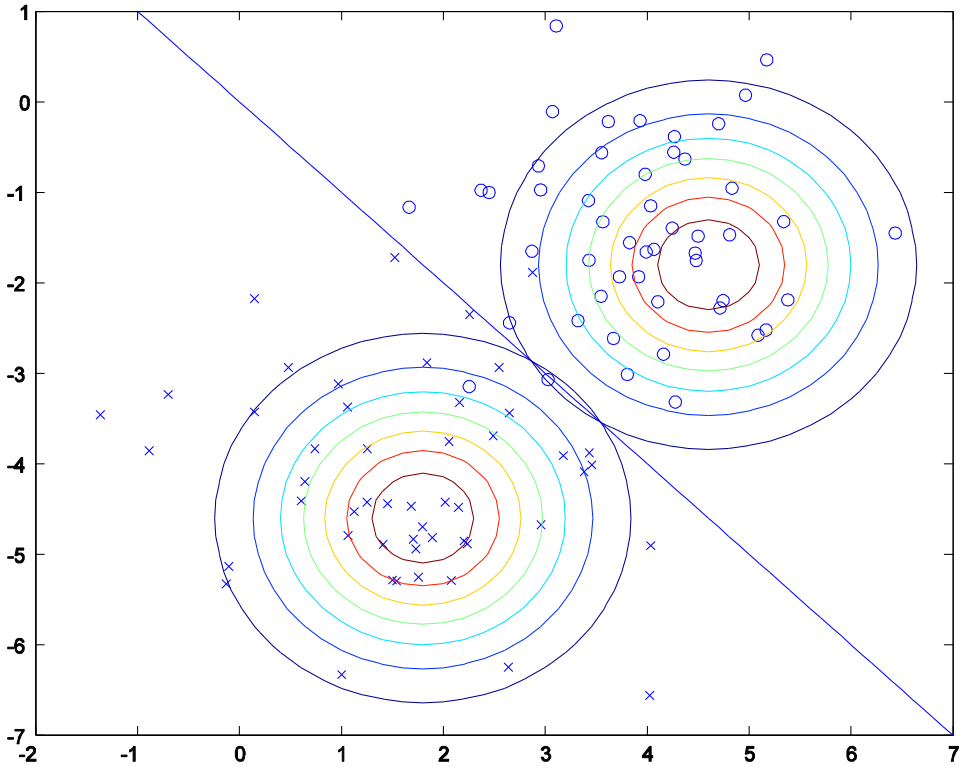
$\Rightarrow$ The decision boundary is linear, so called linear classifier

$\Rightarrow$ When $\Sigma = I$, β is equal to $\mu_1 - \mu_0$, and the contours of equal posterior probability are lines orthogonal to the difference vector between the means of two classes

- When the covariance matrix are different for two classes, we still obtain a logistic form for the posterior probability, but the argument to the logistic function is now quadratic in $\mathbf{x}$

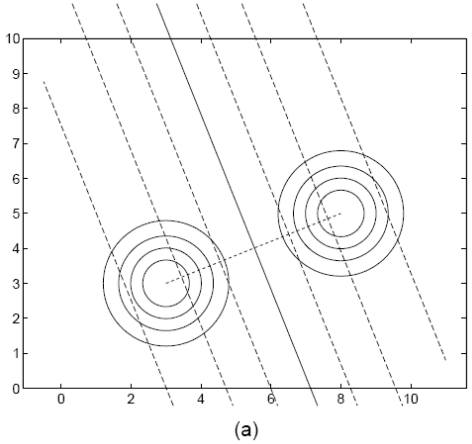
Example of Parametric Methods

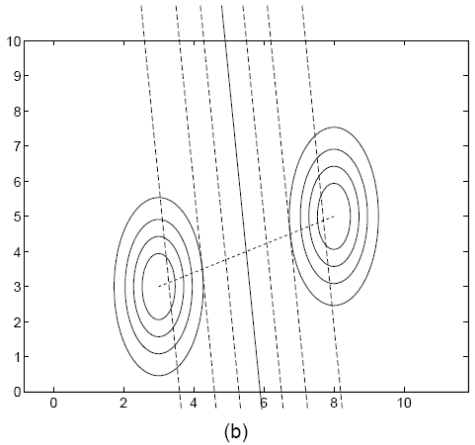
- The decision boundary is at $P(y = 1 \mid \mathbf{x}) = 0.5$
- We have the identical covariance matrix and with all zeros in the off-diagonal terms in the covariance matrix



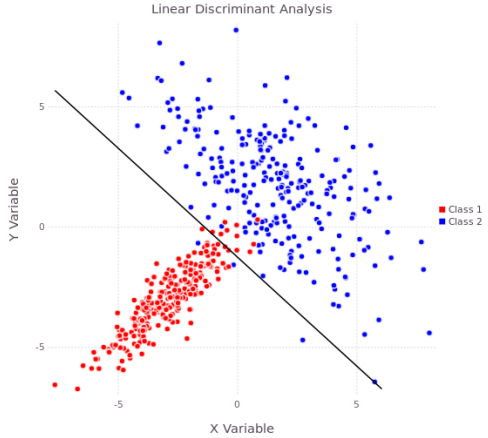
Different Kinds of Covariance Matrices

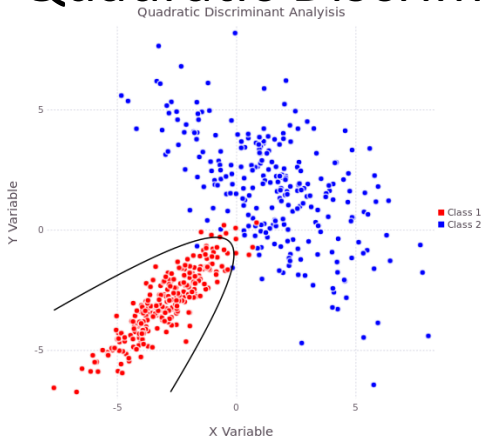
- The decision boundary remains to be linear as long as we have the same covariance matrices for all classes





- Trying Linear Discriminant Analysis or Quadratic Discriminant Analysis





Comparing LMS, LR, LDA

- There is a big debate about the relative merits of
  - direct classifiers (like LMS) versus
  - conditional models (like LR) versus
  - generative models (like LDA)

Issues

- **Statistical efficiency:** if the LDA (generative model) is correct, then it usually gives better accuracy, especially for small training sets.
- **Computational efficiency:** generative models typically are the easiest to compute. In LDA, we estimated the parameters directly, no need for gradient ascent
- **Robustness to changing loss function:** Both LDA (generative model) and logistic regression (conditional model) allow the loss function to change without re-estimating the model. This is not true for LMS
- **Robustness to model assumptions:** The LDA (generative model) usually performs poorly when the assumptions are violated.
- **Robustness to missing values and noise:** In many applications, some of the features may be missing or corrupted for some training examples. Generative models provide better ways of handling this than non-generative models.

LDA vs. LR

- What is the relationship?
  - In LDA, it turns out the $P(y \mid \mathbf{x})$ can be expressed as a logistic function where the weights are some function of π, μ_1, μ_2, and Σ!
  - But, the converse is NOT true. If $P(y \mid \mathbf{x})$ is a logistic function, that does not imply $P(\mathbf{x} \mid y)$ is MVG
- LDA makes stronger modeling assumptions than LR
  - When these modeling assumptions are correct, LDA will perform better
    - GDA is asymptotically efficient: in the limit of very large training sets, there is no algorithm that is strictly better than LDA
  - However, when these assumptions are incorrect, LR is more robust
    - makes weaker assumptions and is more robust to deviations from the modeling assumptions
    - if the data is non-Gaussian, then in the limit, logistic regression will outperform LDA
    - For this reason, LR is a more commonly used algorithm

Generative Modeling vs. Discriminative Modeling

— Linear Models

Generative vs. Discriminative Models

- Consider a classification problem where we want to learn to distinguish elephants $(y = 1)$ from giraffes $(y = 0)$, based on some features of an animal.
- LTU and logistic regression try to find a straight line — a decision boundary — that separates the elephants and the giraffes. Then, to classify a new animal as an elephant or giraffe, it checks which side of the decision boundary the animal lands, and makes the appropriate prediction.
- Another approach, e.g. Naïve Bayes is to first, look at the elephants and build a model of what elephants look like. Next, looking at the giraffes, build a separate model of what giraffes look like. Then, to classify a new animal, we match it against the elephant model and match it against the giraffe model, and see which it is most like.

Generative vs. Discriminative Models (cont'd)

- Algorithms that learn $P(y \mid \mathbf{x})$ directly (aka logistic regression) or algorithms that try to learn mappings directly from the space of inputs $\mathbf{x}$ to the labels $\{0, 1\}$ (such as LTU) are called discriminative models
- Instead, we can try to model $P(\mathbf{x} \mid y)$ and $P(y)$. this approach is called the generative model approach, because we can think of as $P(\mathbf{x}, y)$ a model of how the data is generated.
  - For example, if y indicates whether an example is a giraffe (0) or an elephant (1), then $P(\mathbf{x} \mid y = 0)$ models the distribution of giraffes' features and $P(\mathbf{x} \mid y = 1)$ models the distribution of elephants' features.

Generative vs. Discriminative Models (cont'd)

- For generative models, after learning $P(y)$ (the class priors) and $P(\mathbf{x} \mid y)$, we use Bayes' rule to derive the posterior distribution of y given $\mathbf{x}$:

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

  We can compute $P(\mathbf{x})$ from $P(y)$ and $P(\mathbf{x} \mid y)$

- To make a prediction:

$$\text{argmax}_y P(y \mid \mathbf{x}) = \text{argmax}_y \frac{P(\mathbf{x} \mid y)P(y)}{P(\mathbf{x})}$$

- As for discriminative models, we are calculating $P(y \mid \mathbf{x})$ to make a prediction, therefore we don't need to compute $P(\mathbf{x})$

Generative vs. Discriminative Models (cont'd)

(Rubinstein 97)

	Generative	Discriminative
Example	Naïve Bayes	Logistic Regression
Objective	Learning the full distribution	Learning the border
Model Assumptions	Class densities: $P(\mathbf{x} \mid y = k)$ e.g. Gaussian in Gaussian Naïve Bayes	Discriminant functions $h(\mathbf{x})$
Parameter Estimation	"Easy" – One single sweep	"Hard" – iterative optimization
Advantages	More efficient if model correct, borrows strength from $P(\mathbf{x})$	More flexible, robust because fewer assumptions
Disadvantages	Bias if model is incorrect	May also be biased. Ignores information in $P(\mathbf{x})$