Machine Learning (CS 567) Lecture 4

Fall 2008

Time: T-Th 5:00pm - 6:20pm
Location: GFS 118

Instructor: Sofus A. Macskassy (macskass@usc.edu)
Office: SAL 216
Office hours: by appointment

Teaching assistant: **Cheol Han (cheolhan@usc.edu)**
Office: SAL 229
Office hours: M 2-3, W 11-12

Class web page:
http://www-scf.usc.edu/~csci567/index.html

Administrative: Home work

- Due at start of class on due date.
- 25% off if after class but before 8am next morning
- 50% off if received next day after 8am
- 100% off if not received next day
- If you have a valid excuse, let me know ASAP with proper documentation. No exceptions.

- Grading compliant policy: if students have problems at assignment grading, feel free to talk to the instructor/TA for it. However, even if students only request re-grading one of the answers, all this assignment will be checked and graded again. (take the risk of possible losing total points.)

Administrative: Final Project

- Groups of 2-4 people
- Deliverables:
  - Research paper
    - Write paper as though submitting to a conference or workshop.
  - Presentation at the last two classes
- Work should cover:
  - Good evaluation technique, compare at least 2 classifiers, findings should be clearly stated and validated, related work should be cited.
- Project pre-proposals due on September 23
  - You can interact with me before hand
  - 1-2 paragraphs outlining the problem and what you will do
- Project proposals will due on October 9
  - 1-2 pages detailing what you will do, who are in the group, the data, the machine learning methods you will look at, etc

Project Idea: Applied Learning

Take an interesting dataset

Compare several learning approaches for prediction

- decision trees
- ANNs
- instance-based methods
- SVMs
- boosting

Project Idea: Improvements to Learning Methods

There are many suggestions on how to improve various learning methods, both in books and in papers.

Identify some suggestions and test them empirically.

Project Idea: Comparison of Learning Methods

Identify some learners.

Run thorough comparison tests and determine the reasons for their different performances.

Text Classification

Easy to get lots of text: web, TREC data, email (e.g., enron)

Predict topic, authorship, sentiment, style, affect, attitude.

Reinforcement Learning

Can be challenging to do well: generate data or direct control

- Optimize gas well production
- Object tracking
- "Tag" grid world
- Rover control
- animal behavior experiments

Compare approaches (direct policy search, value function learning, model-based, …)

Active Learning

Take a classic dataset.

Explore the tradeoff between size of training set and generalization.

Devise schemes for choosing items to "pay" for labels to maximize accuracy with minimum cost.

Project Methodology

Lots of good ideas for algorithms and domains.

The hard question is: "How will you evaluate it?"

Ultimately, you need to present more than one algorithm (and perhaps more than one problem) and you'll need some way of saying what worked better.

What's the gold standard?

Lecture 4 Outline

- Linear Threshold Units
  - Perceptron
  - Logistic Regression

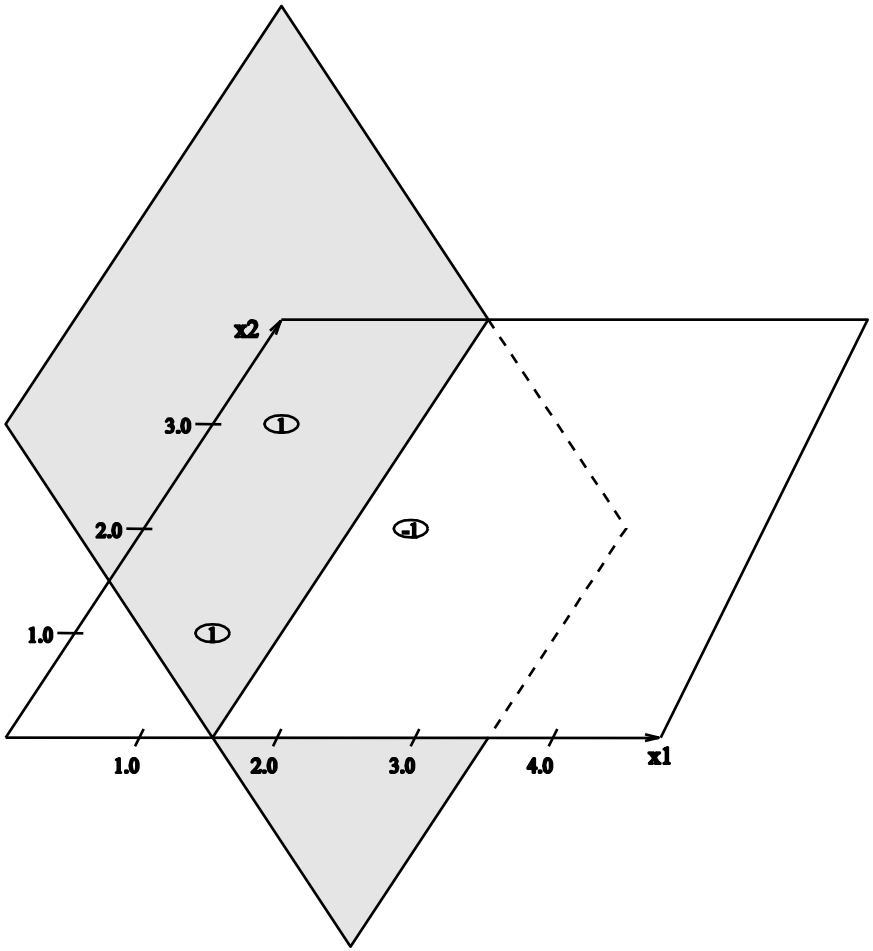
The Unthresholded Discriminant Function is a Hyperplane

- The equation

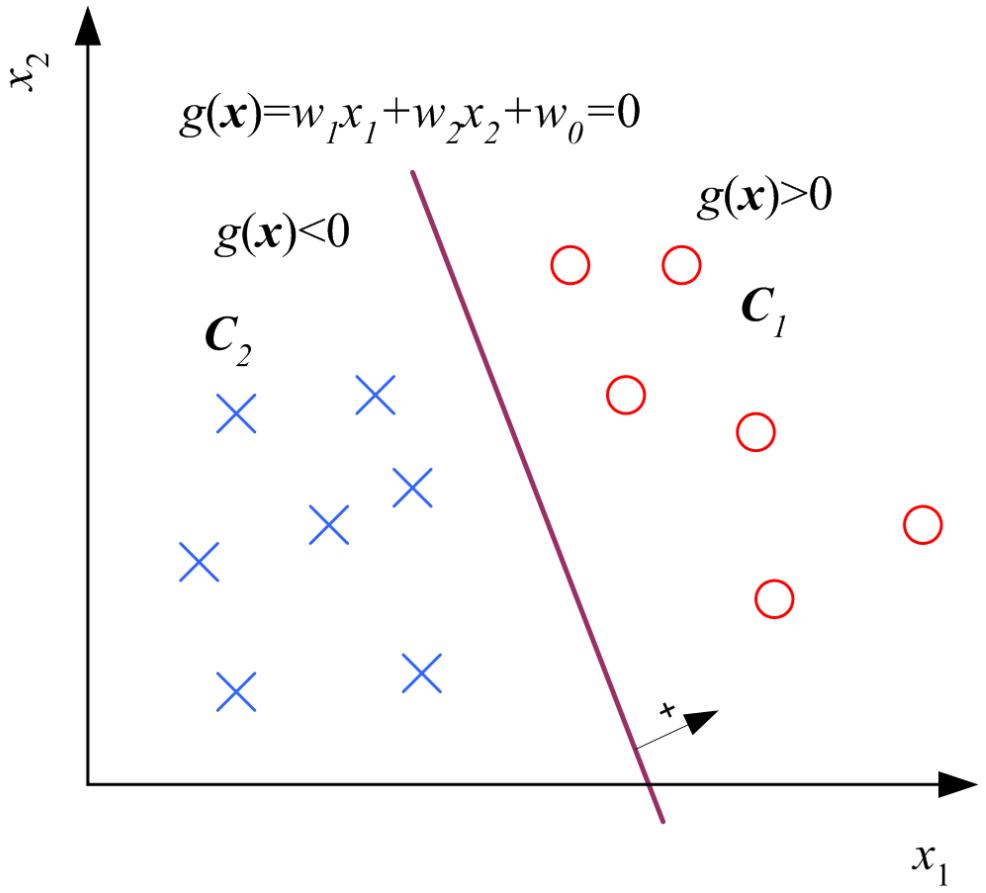
  $g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$

  is a plane

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



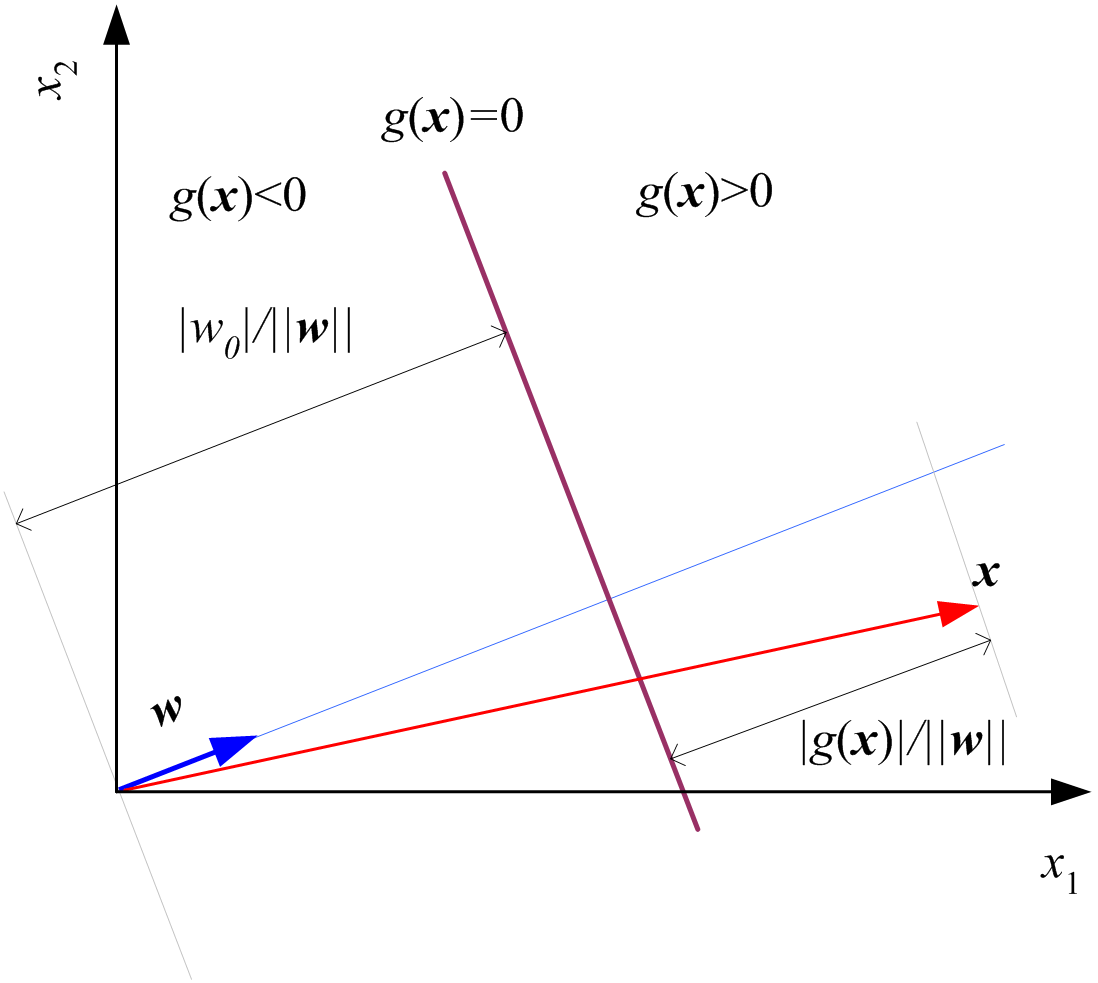
Alternatively…



$$
\begin{aligned}
g(\mathbf{x}) &= g_1(\mathbf{x}) - g_2(\mathbf{x}) \\
&= \left(\mathbf{w}_1^T\mathbf{x} + w_{10}\right) - \left(\mathbf{w}_2^T\mathbf{x} + w_{20}\right) \\
&= \left(\mathbf{w}_1 - \mathbf{w}_2\right)^T\mathbf{x} + \left(w_{10} - w_{20}\right) \\
&= \mathbf{w}^T\mathbf{x} + w_0
\end{aligned}
$$

$$
\text{choose} \quad \begin{cases} C_1 & \text{if } g(\boldsymbol{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}
$$

Geometry

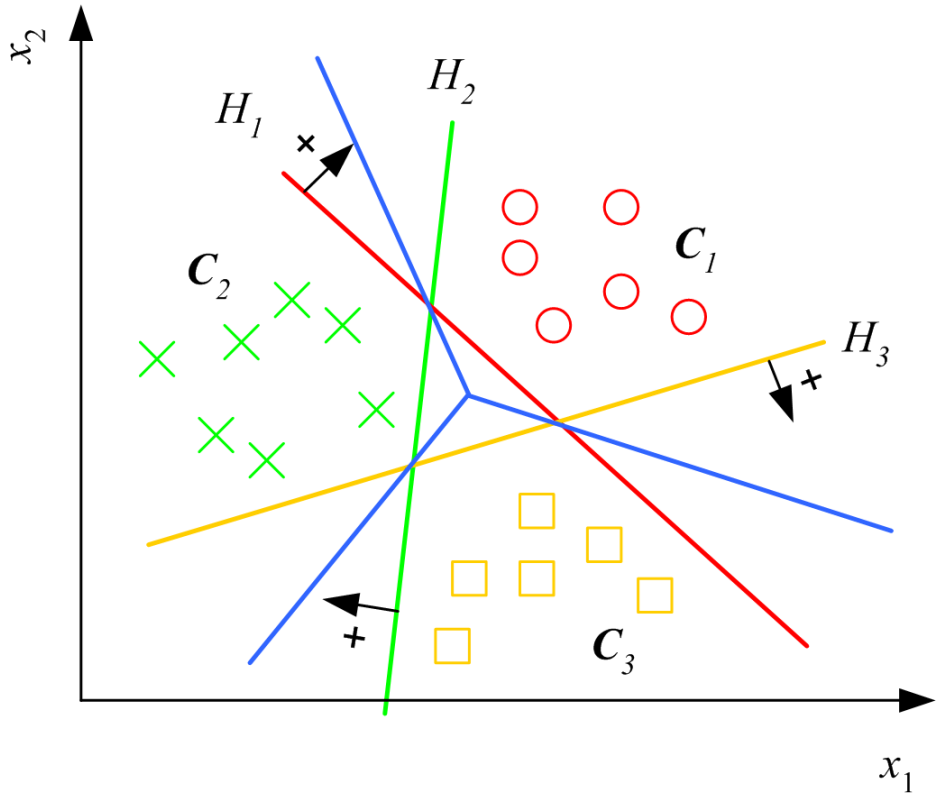

x_2
$g(\boldsymbol{x})=0$
$g(\boldsymbol{x})<0$
$g(\boldsymbol{x})>0$
$|w_0|/||\boldsymbol{w}||$
$\boldsymbol{x}$
$\boldsymbol{w}$
$|g(\boldsymbol{x})|/||\boldsymbol{w}||$
x_1

Multiple Classes

$$g_i(\boldsymbol{x} \mid \boldsymbol{w}_i, w_{i0}) = \boldsymbol{w}_i^T \boldsymbol{x} + w_{i0}$$

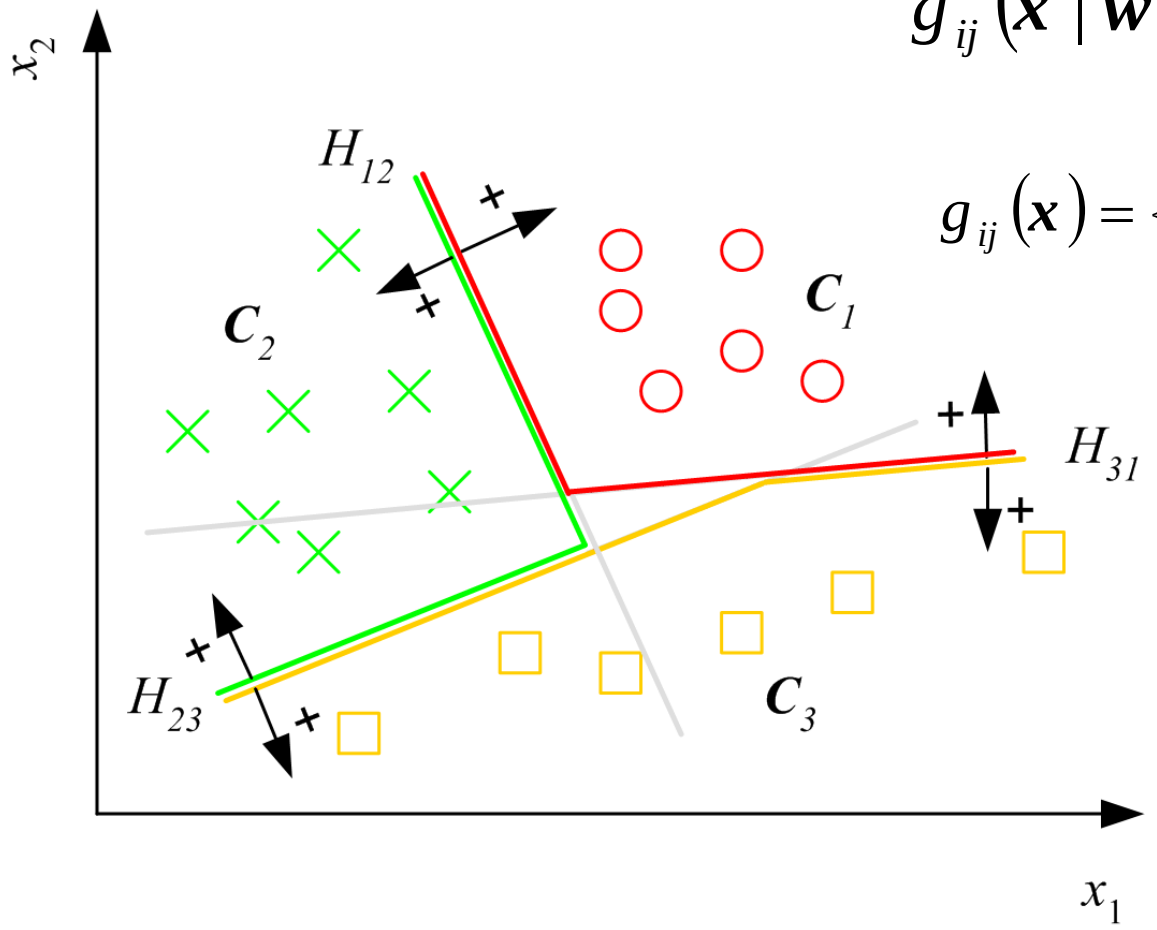
Choose C_i if

$$g_i(\mathbf{x}) = \max_{j=1}^{K} g_j(\mathbf{x})$$



Classes are linearly separable

Pairwise Separation



$$g_{ij}\left(\boldsymbol{x} \mid \boldsymbol{w}_{ij}, w_{ij0}\right) = \boldsymbol{w}_{ij}^{T}\boldsymbol{x} + w_{ij0}$$

$$g_{ij}(\boldsymbol{x}) = \begin{cases} > 0 & \text{if } \boldsymbol{x} \in C_i \\ \leq 0 & \text{if } \boldsymbol{x} \in C_j \\ \text{don't care} & \text{otherwise} \end{cases}$$

choose C_i if

$$\forall j \neq i, g_{ij}(\boldsymbol{x}) > 0$$

Machine Learning and Optimization

- When learning a classifier, the natural way to formulate the learning problem is the following:
  - Given:
    - A set of *N* training examples
      $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$
    - A loss function *L*
  - Find:
    - The weight vector **w** that minimizes the expected loss on the training data

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L(\text{sgn}(\mathbf{w} \cdot \mathbf{x}_i), y_i).$$

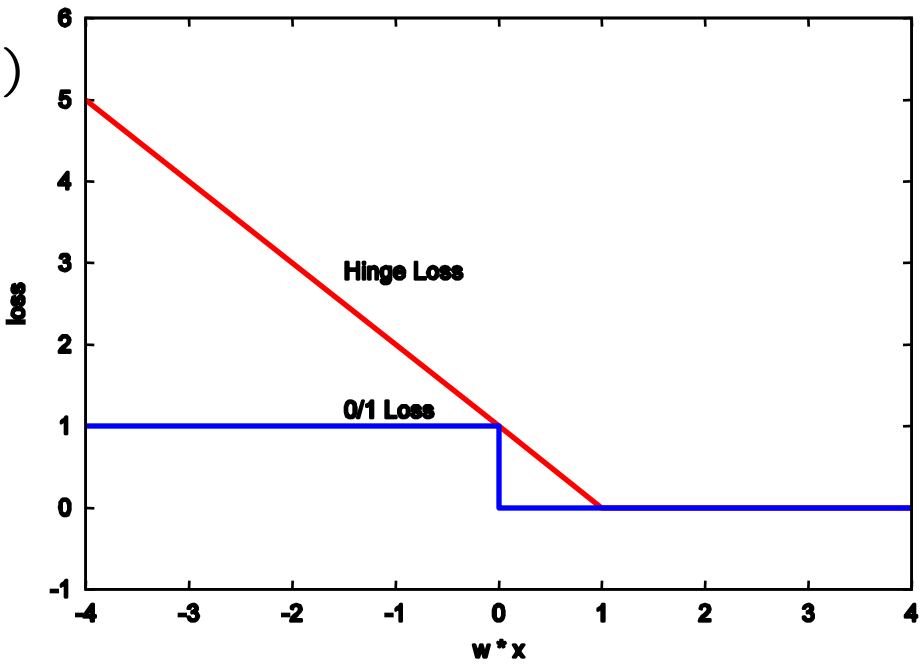
- In general, machine learning algorithms apply some optimization algorithm to find a good hypothesis. In this case, *J* is piecewise constant, which makes this a difficult problem

Approximating the expected loss by a smooth function

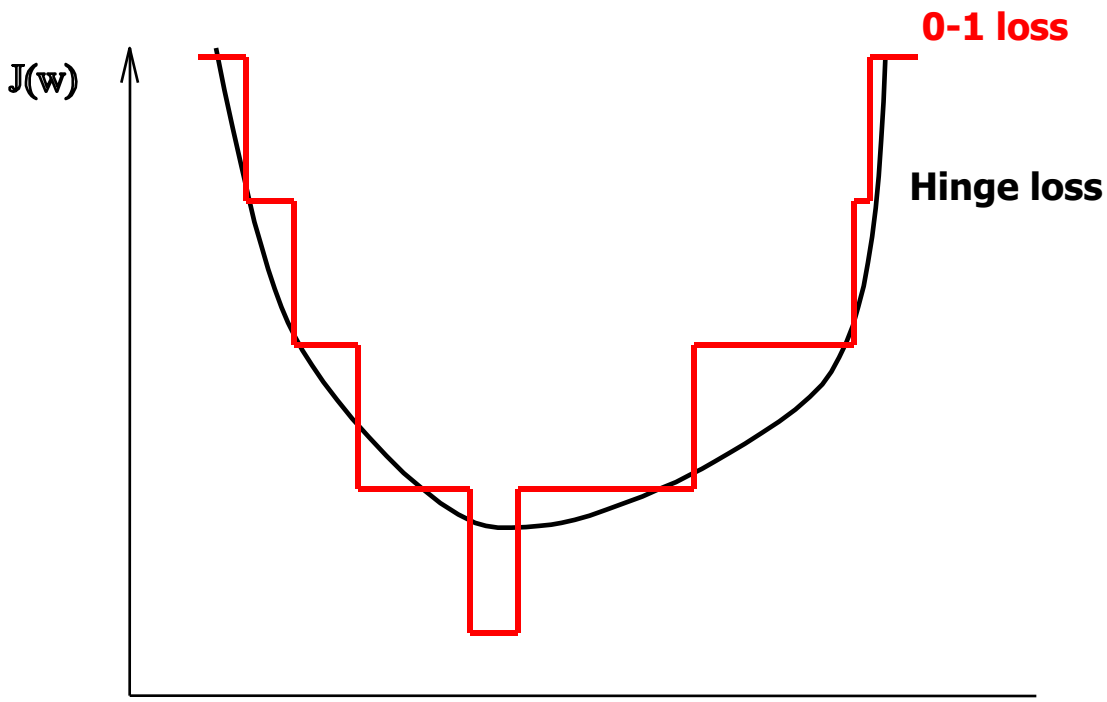
- Simplify the optimization problem by replacing the original objective function by a smooth, differentiable function. For example, consider the *hinge loss*:

$$\tilde{J}(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} max(0, 1 - y_i \mathbf{w} \cdot \mathbf{x}_i)$$

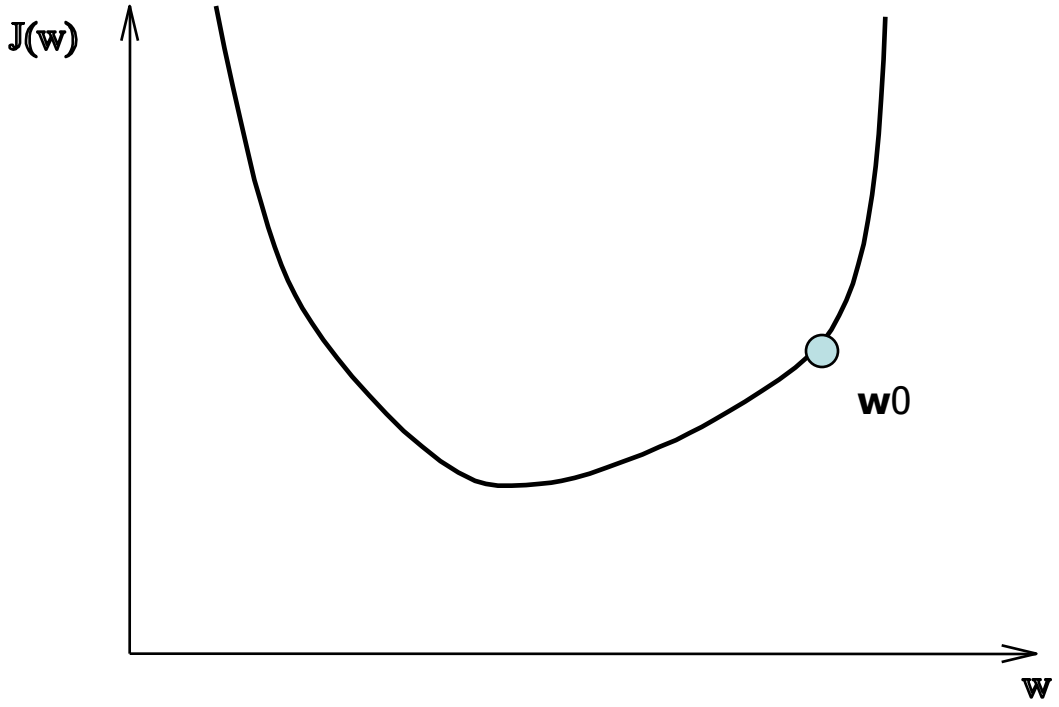
When $y = 1$



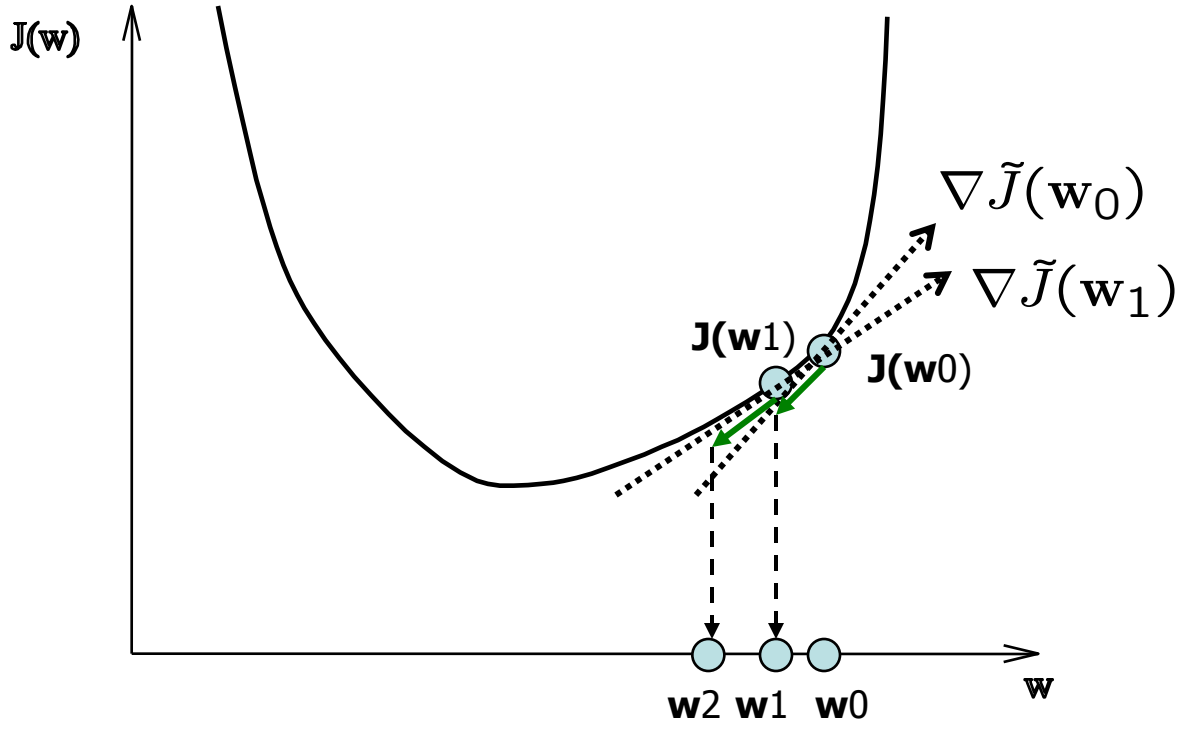
Optimizing… what?



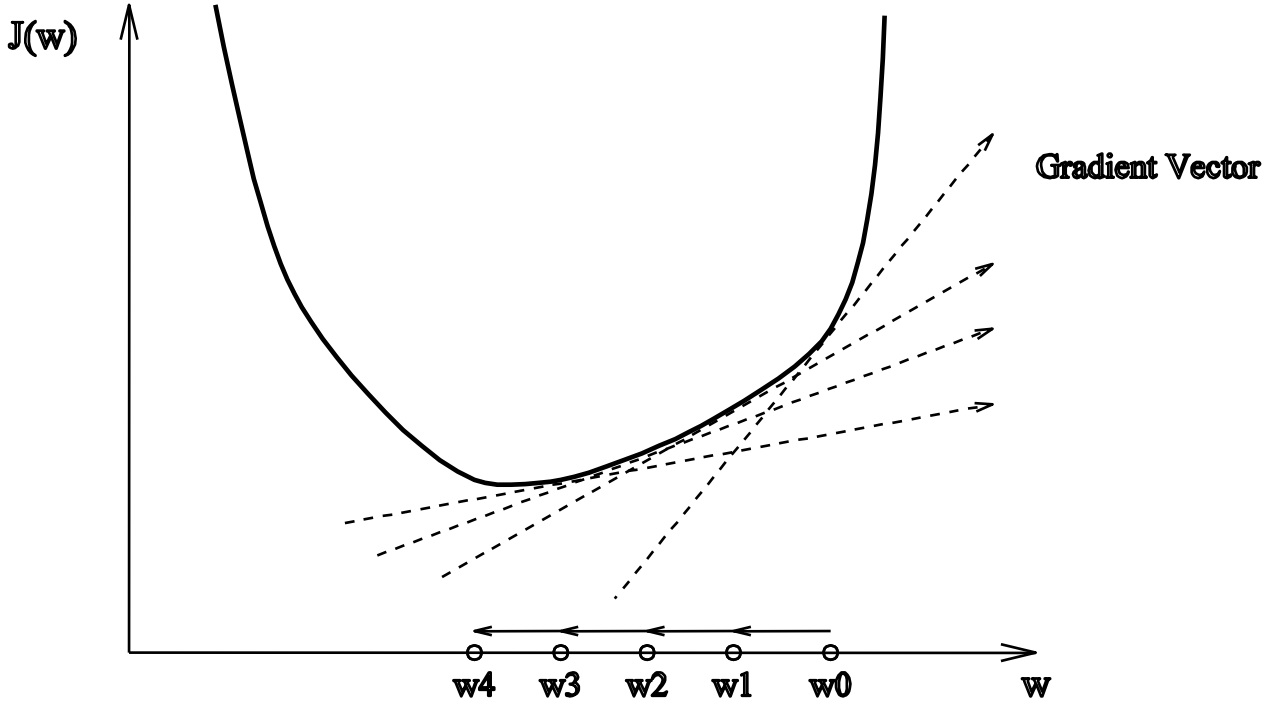
Optimizing… what?



Optimizing… what?



Minimizing $\tilde{J}$ by Gradient Descent Search



- Start with weight vector $\mathbf{w}_0$
- Compute gradient $\nabla\tilde{J}(\mathbf{w}_0) = \left(\frac{\partial\tilde{J}(\mathbf{w}_0)}{\partial w_0}, \frac{\partial\tilde{J}(\mathbf{w}_0)}{\partial w_1}, \ldots, \frac{\partial\tilde{J}(\mathbf{w}_0)}{\partial w_n}\right)$
- Compute $\mathbf{w}_1 = \mathbf{w}_0 - \eta\nabla\tilde{J}(\mathbf{w}_0)$
  where η is a "step size" parameter
- Repeat until convergence

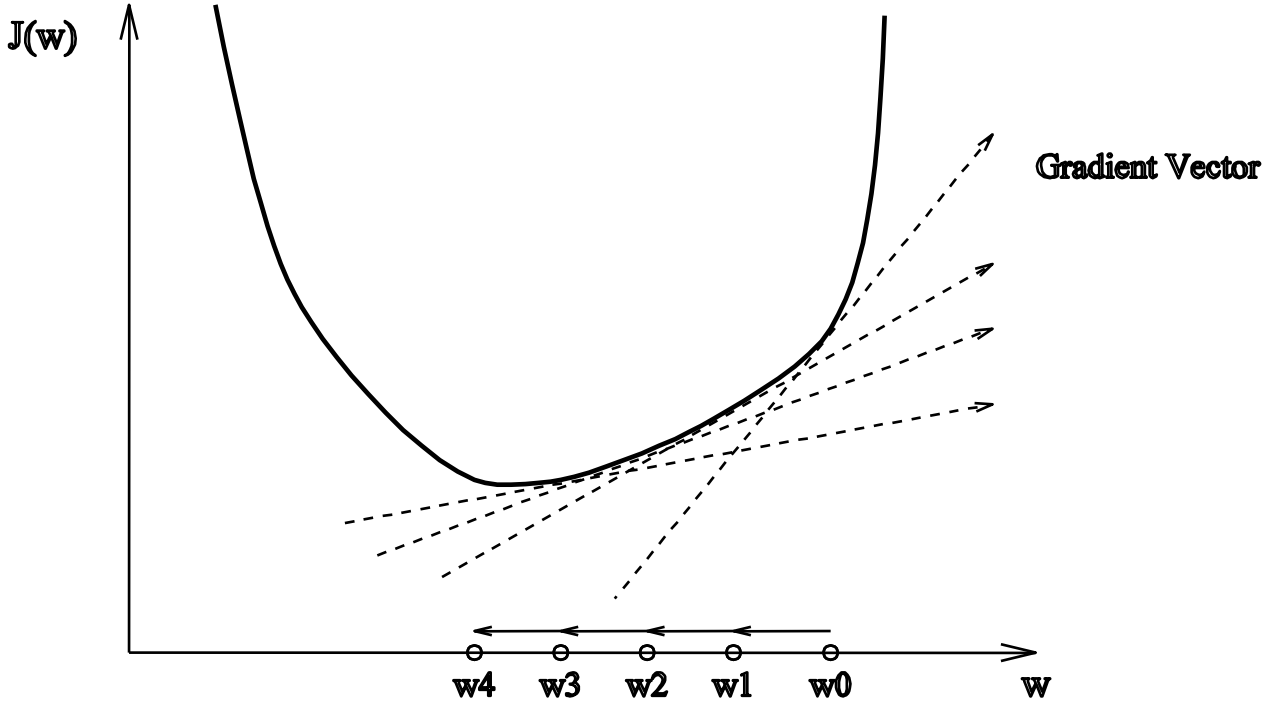
Computing the Gradient

$$\text{Let } \tilde{J}_i(\mathbf{w}) = \max(0, -y_i \mathbf{w} \cdot \mathbf{x}_i)$$

$$\begin{aligned}
\frac{\partial \tilde{J}(\mathbf{w})}{\partial w_k} &= \frac{\partial}{\partial w_k}\left(\frac{1}{N}\sum_{i=1}^{N} \tilde{J}_i(\mathbf{w})\right) \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\partial}{\partial w_k}\tilde{J}_i(\mathbf{w})\right)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \tilde{J}_i(\mathbf{w})}{\partial w_k} &= \frac{\partial}{\partial w_k}\max\left(0, -y_i \sum_j w_j x_{ij}\right) \\
&= \begin{cases} 0 & \text{if } y_i \sum_j w_j x_{ij} > 0 \\ -y_i x_{ik} & \text{otherwise} \end{cases}
\end{aligned}$$

Perceptron: Gradient Descent Search



- Start with weight vector $\mathbf{w}_0$
- Compute gradient

$$\nabla \tilde{J}(\mathbf{w}_0) = \left(\frac{\partial \tilde{J}(\mathbf{w}_0)}{\partial w_0}, \frac{\partial \tilde{J}(\mathbf{w}_0)}{\partial w_1}, \ldots, \frac{\partial \tilde{J}(\mathbf{w}_0)}{\partial w_n} \right)$$

- Compute $\mathbf{w}_1 = \mathbf{w}_0 - \eta \nabla \tilde{J}(\mathbf{w}_0)$ where η is a "step size" parameter
- Repeat until convergence

$$\frac{\partial \tilde{J}_i(\mathbf{w})}{\partial w_k} = \begin{cases} 0 & \text{if } y_i \sum_j w_j x_{ij} > 0 \\ -y_i x_{ik} & \text{otherwise} \end{cases}$$

Batch Perceptron Algorithm

Given: training examples $(\mathbf{x}_i, y_i)$, $i = 1 \ldots N$

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

Repeat until convergence

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

For $i = 1$ to N do

$u_i = \mathbf{w} \cdot \mathbf{x}_i$

if $(y_i \cdot u_i \leq 0)$

For $j = 1$ to n do

$g_j = g_j - y_i \cdot x_{ij}$

$\mathbf{g} = \mathbf{g}/N$

$\mathbf{w} = \mathbf{w} - \eta\mathbf{g}$

Simplest case: $\eta = 1$, don't normalize g: "Fixed Increment Perceptron"

Online Perceptron Algorithm

Let $\mathbf{w}$ = (0,0, …, 0) be the initial weight vector
Repeat forever
    Let $\mathbf{g}$ = (0,0, …, 0) be the gradient vector
    Accept training example i: $<\mathbf{x}_i, y_i>$
    $u_i = \mathbf{w} \cdot \mathbf{x}_i$
    if ($y_i \cdot u_i \leq 0$)
        For j = 1 to n do
            $g_j = y_i \cdot x_{ij}$
        $\mathbf{w} = \mathbf{w} + \eta\mathbf{g}$

This is called stochastic gradient descent because the overall gradient is approximated by the gradient from each individual example

Learning Rates and Convergence

- The learning rate η must decrease to zero in order to guarantee convergence. The online case is known as the Robbins-Munro algorithm. It is guaranteed to converge under the following assumptions:

$$\lim_{t \to \infty} \eta_t = 0$$

$$\sum_{t=0}^{\infty} \eta_t = \infty$$

$$\sum_{t=0}^{\infty} \eta_t^2 < \infty$$

- The learning rate is also called the step size. Some algorithms (e.g., Newton's method, conjugate gradient) choose the stepsize automatically and converge faster
- There is only one "basin" for linear threshold units, so a local minimum is the global minimum. Choosing a good starting point can make the algorithm converge faster

Summary of Perceptron algorithm for LTUs

- Directly Learns a Classifier
- Local Search
  - Begins with an initial weight vector. Modifies it iterative to minimize an error function. The error function is loosely related to the goal of minimizing the number of classification errors
- Eager
  - The classifier is constructed from the training examples
  - The training examples can then be discarded
- Online or Batch
  - Both variants of the algorithm can be used

Linear Threshold Units

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

- We assume that each feature x_j and each weight w_j is a real number (we will relax this later)
- We will study three different algorithms for learning linear threshold units:
  - Perceptron: classifier
  - Logistic Regression: conditional distribution
  - Linear Discriminant Analysis: joint distribution

Logistic Regression

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

$$p_1(\mathbf{x}; \mathbf{w}) = \frac{\exp \mathbf{w} \cdot \mathbf{x}}{1 + \exp \mathbf{w} \cdot \mathbf{x}}.$$

$$p_0(\mathbf{x}; \mathbf{w}) = 1 - p_1(\mathbf{x}; \mathbf{w}).$$

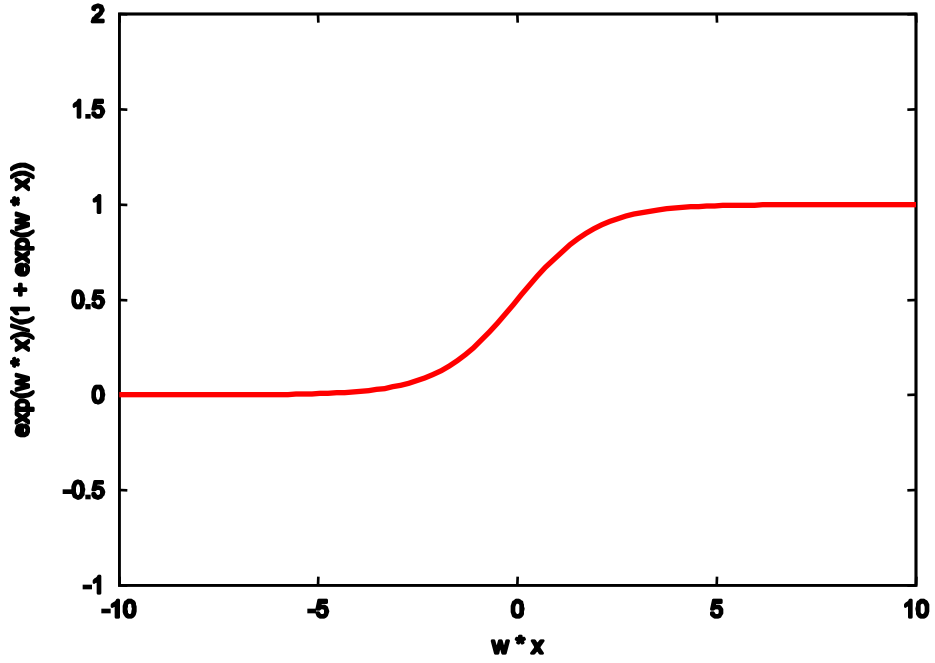
- This is equivalent to

$$\log \frac{p_1(\mathbf{x}; \mathbf{w})}{p_0(\mathbf{x}; \mathbf{w})} = \mathbf{w} \cdot \mathbf{x}.$$

- In other words, the log odds of class 1 is a linear function of **x**.

Why the exp function?

- One reason: A linear function has a range from $[-\infty, \infty]$ and we need to force it to be positive and sum to 1 in order to be a probability:



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 seek to find the probability 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):

$$
\begin{aligned}
\operatorname*{argmax}_h P(h|S) &= \operatorname*{argmax}_h \frac{P(S|h)P(h)}{P(S)} && \text{by Bayes' Rule} \\
&= \operatorname*{argmax}_h P(S|h)P(h) && \text{because } P(S) \text{ doesn't depend on } h \\
&= \operatorname*{argmax}_h P(S|h) && \text{if we assume } P(h) = \text{uniform} \\
&= \operatorname*{argmax}_h \log P(S|h) && \text{because log is monotonic}
\end{aligned}
$$

The distribution P(S|*h*) is called the <u>likelihood function</u>. The log likelihood is frequently used as the objective function for learning. It is often written as $\ell(h)$.

The *h* that maximizes the likelihood on the training data is called the <u>maximum likelihood estimator (MLE)</u>

Computing the Likelihood

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

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

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

Computing the Likelihood (2)

- Recall that *any* joint distribution P(a,b) can be factored as P(a|b) P(b). Hence, we can write

$$\begin{aligned}\operatorname*{argmax}_h \log P(S|h) &= \operatorname*{argmax}_h \sum_i \log P(\mathbf{x}_i, y_i|h) \\ &= \operatorname*{argmax}_h \sum_i \log P(y_i|\mathbf{x}_i, h)P(\mathbf{x}_i|h)\end{aligned}$$

- In our case, P(**x** | *h*) = P(**x**), because it does not depend on *h*, so

$$\begin{aligned}\operatorname*{argmax}_h \log P(S|h) &= \operatorname*{argmax}_h \sum_i \log P(y_i|\mathbf{x}_i, h)P(\mathbf{x}_i|h) \\ &= \operatorname*{argmax}_h \sum_i \log P(y_i|\mathbf{x}_i, h)\end{aligned}$$

Log Likelihood for Conditional Probability Estimators

- We can express the log likelihood in a compact form known as the cross entropy.
- 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 $\log [p_1(\mathbf{x}; \mathbf{w})]$
- These cases are mutually exclusive, so we can combine them to obtain:

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

- The goal of our learning algorithm will be to find **w** to maximize

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

Fitting Logistic Regression by Gradient Ascent

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial w_j} &= \sum_i \frac{\partial}{\partial w_j} \ell(y_i; \mathbf{x}_i, \mathbf{w}) \\
\frac{\partial}{\partial w_j} \ell(y_i; \mathbf{x}_i, \mathbf{w}) &= \frac{\partial}{\partial w_j} \left((1 - y_i) \log[1 - p_1(\mathbf{x}_i; \mathbf{w})] + y_i \log p_1(\mathbf{x}_i; \mathbf{w}) \right) \\
&= (1 - y_i) \frac{1}{1 - p_1(\mathbf{x}_i; \mathbf{w})} \left(-\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) + y_i \frac{1}{p_1(\mathbf{x}_i; \mathbf{w})} \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) \\
&= \left[\frac{y_i}{p_1(\mathbf{x}_i; \mathbf{w})} - \frac{(1 - y_i)}{1 - p_1(\mathbf{x}_i; \mathbf{w})} \right] \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) \\
&= \left[\frac{y_i (1 - p_1(\mathbf{x}_i; \mathbf{w})) - (1 - y_i) p_1(\mathbf{x}_i; \mathbf{w})}{p_1(\mathbf{x}_i; \mathbf{w})(1 - p_1(\mathbf{x}_i; \mathbf{w}))} \right] \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) \\
&= \left[\frac{y_i - p_1(\mathbf{x}_i; \mathbf{w})}{p_1(\mathbf{x}_i; \mathbf{w})(1 - p_1(\mathbf{x}_i; \mathbf{w}))} \right] \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right)
\end{aligned}
$$

Gradient Computation (continued)

- Note that p_1 can also be written as

$$p_1(\mathbf{x}_i; \mathbf{w}) = \frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])}.$$

- From this, we obtain:

$$\begin{aligned}
\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} &= -\frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])^2} \frac{\partial}{\partial w_j}(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i]) \\
&= -\frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])^2} \exp[-\mathbf{w} \cdot \mathbf{x}_i] \frac{\partial}{\partial w_j}(-\mathbf{w} \cdot \mathbf{x}_i) \\
&= -\frac{1}{(1 + \exp[-\mathbf{w} \cdot \mathbf{x}_i])^2} \exp[-\mathbf{w} \cdot \mathbf{x}_i](-x_{ij}) \\
&= p_1(\mathbf{x}_i; \mathbf{w})(1 - p_1(\mathbf{x}_i; \mathbf{w}))x_{ij}
\end{aligned}$$

Completing the Gradient Computation

- The gradient of the log likelihood of a single point is therefore

$$\begin{aligned}
\frac{\partial}{\partial w_j}\ell(y_i;\mathbf{x}_i,\mathbf{w}) &= \left[\frac{y_i - p_1(\mathbf{x}_i;\mathbf{w})}{p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))}\right]\left(\frac{\partial p_1(\mathbf{x}_i;\mathbf{w})}{\partial w_j}\right) \\
&= \left[\frac{y_i - p_1(\mathbf{x}_i;\mathbf{w})}{p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))}\right] p_1(\mathbf{x}_i;\mathbf{w})(1-p_1(\mathbf{x}_i;\mathbf{w}))x_{ij} \\
&= (y_i - p_1(\mathbf{x}_i;\mathbf{w}))x_{ij}
\end{aligned}$$

- The overall gradient is

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

Batch Gradient Ascent for Logistic Regression

Given: training examples $(\mathbf{x}_i, y_i)$, $i = 1 \ldots N$

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

Repeat until convergence

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

  For $i = 1$ to N do

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

    $\text{error}_i = y_i - p_i$

    For $j = 1$ to n do

      $g_j = g_j - \text{error}_i \cdot x_{ij}$

  $\mathbf{w} = \mathbf{w} + \eta\mathbf{g}$ (step in direction of increasing gradient)

- An online gradient ascent algorithm can be constructed, of course
- Most statistical packages use a second-order (Newton-Raphson) algorithm for faster convergence. Each iteration of the second-order method can be viewed as a weighted least squares computation, so the algorithm is known as Iteratively-Reweighted Least Squares (IRLS)

Batch Gradient Ascent for Logistic Regression

Given: training examples $(\mathbf{x}_i, y_i)$, $i = 1 \ldots N$

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

Repeat until convergence

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

For $i = 1$ to N do

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

$\text{error}_i = y_i - p_i$

For $j = 1$ to n do

$g_j = g_j - \text{error}_i \cdot x_{ij}$

$\mathbf{w} = \mathbf{w} + \eta \mathbf{g}$ (step in direction of increasing gradient)

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

- An online gradient ascent algorithm can be constructed, of course
- Most statistical packages use a second-order (Newton-Raphson) algorithm for faster convergence. Each iteration of the second-order method can be viewed as a weighted least squares computation, so the algorithm is known as Iteratively-Reweighted Least Squares (IRLS)

Logistic Regression Implements a Linear Discriminant Function

- In the 2-class 0/1 loss function case, we should predict $\hat{y} = 1$ if

$$
\begin{aligned}
E_{y|\mathbf{x}}[L(0,y)] &> E_{y|\mathbf{x}}[L(1,y)] \\
\sum_y P(y|\mathbf{x})L(0,y) &> \sum_y P(y|\mathbf{x})L(1,y) \\
P(y=0|\mathbf{x})L(0,0) + P(y=1|\mathbf{x})L(0,1) &> P(y=0|\mathbf{x})L(1,0) + P(y=1|\mathbf{x})L(1,1) \\
P(y=1|\mathbf{x}) &> P(y=0|\mathbf{x}) \\
\frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} &> 1 \qquad \text{if } P(y=0|X) \neq 0 \\
\log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})} &> 0 \\
\mathbf{w} \cdot \mathbf{x} &> 0
\end{aligned}
$$

- A similar derivation can be done for arbitrary L(0,1) and L(1,0).

Extending Logistic Regression to K > 2 classes

- 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:

$$\log \frac{P(y=1|\mathbf{x})}{P(y=K|\mathbf{x})} = \mathbf{w}_1 \cdot \mathbf{x}$$
$$\log \frac{P(y=2|\mathbf{x})}{P(y=K|\mathbf{x})} = \mathbf{w}_2 \cdot \mathbf{x}$$
$$\vdots$$
$$\log \frac{P(y=K-1|\mathbf{x})}{P(y=K|\mathbf{x})} = \mathbf{w}_{K-1} \cdot \mathbf{x}$$

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

Logistic Regression for K > 2 (continued)

- The conditional probability for class k ≠ K can be computed as

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

- For class K, the conditional probability is

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

Summary of Logistic Regression

- Learns conditional probability distribution P(y | **x**)
- Local Search
  - begins with initial weight vector. Modifies it iteratively to maximize the log likelihood of the data
- Eager
  - the classifier is constructed from the training examples, which can then be discarded
- Online or Batch
  - both online and batch variants of the algorithm exist