Linear Threshold Units

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

- 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

What can be represented by an LTU:

- Conjunctions

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

- At least *m*-of-*n*

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

Things that cannot be represented:

- Non-trivial disjunctions:

$(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \Leftrightarrow y$

$1 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 + 1 \cdot x_4 \geq 2$ predicts

$f(\langle 0110 \rangle) = 1.$

- Exclusive-OR:

$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) \Leftrightarrow y$

A canonical representation

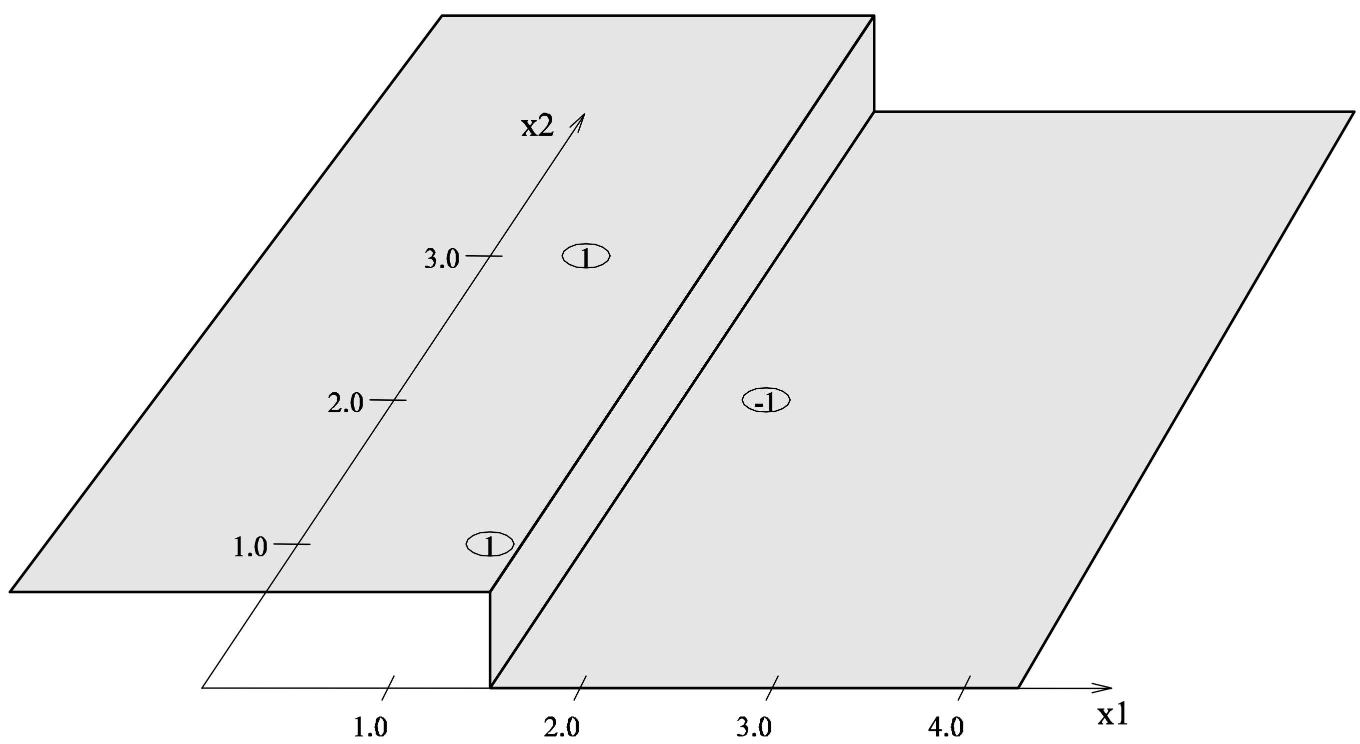
- Given a training example of the form
  $(\langle x_1, x_2, x_3, x_4 \rangle, y)$
- transform it to
  $(1, x_1, x_2, x_3, x_4 \rangle, y)$
- The parameter vector will then be
  $\mathbf{w} = \langle w_0, w_1, w_2, w_3, w_4 \rangle$.
- We will call the *unthresholded* hypothesis $u(\mathbf{x},\mathbf{w})$
  $u(\mathbf{x},\mathbf{w}) = \mathbf{w} \cdot \mathbf{x}$
- Each hypothesis can be written
  $h(\mathbf{x}) = \mathrm{sgn}(u(\mathbf{x},\mathbf{w}))$
- Our goal is to find **w**.

The LTU Hypothesis Space

- Fixed size: There are $O\left(2^{n^2}\right)$ distinct linear threshold units over *n* boolean features
- Deterministic
- Continuous parameters

Geometrical View

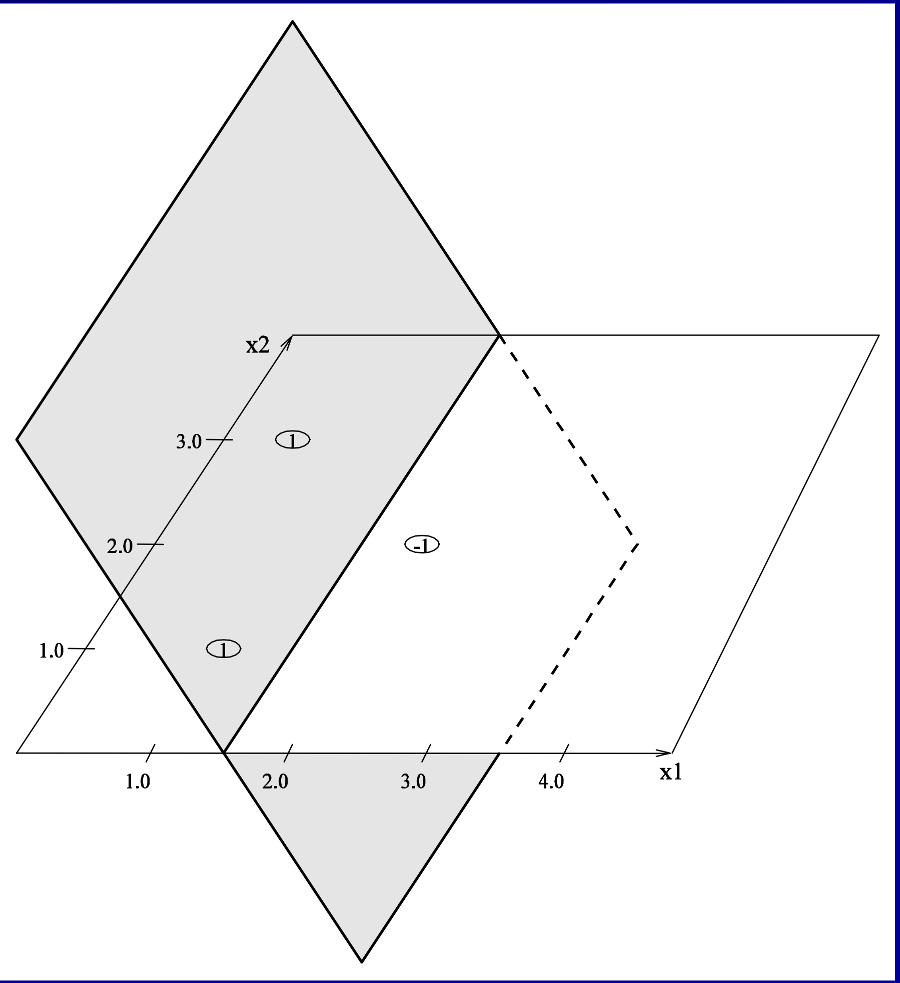
- Consider three training examples: $(\langle 1.0, 1.0 \rangle, +1)$
  $(\langle 0.5, 3.0 \rangle, +1)$
  $(\langle 2.0, 2.0 \rangle, -1)$
- We want a classifier that looks like the following:



The Unthresholded Discriminant Function is a Hyperplane

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

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



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

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

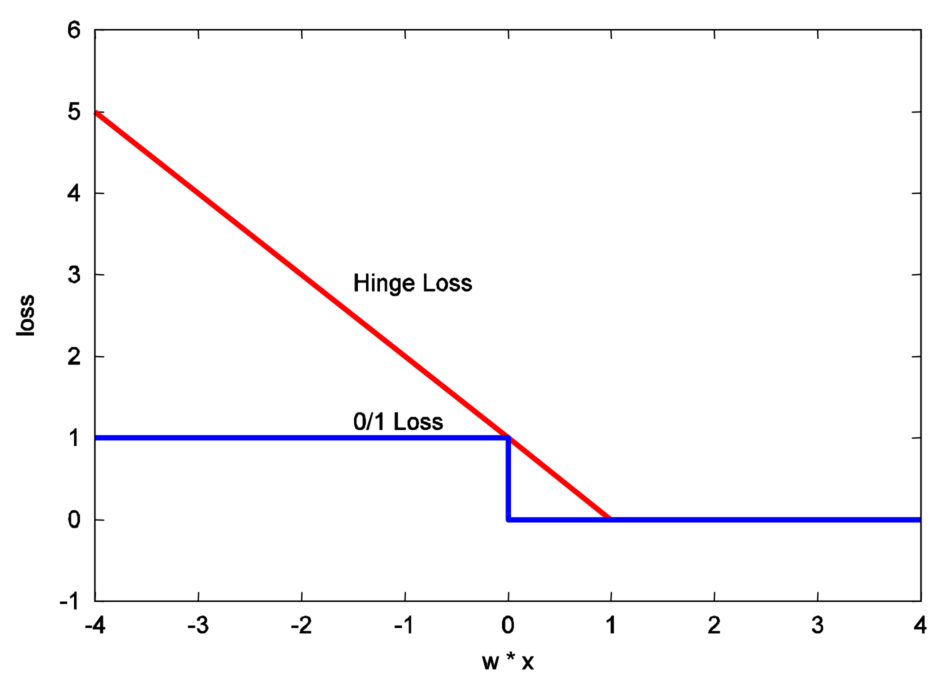
- In general, machine learning algorithms apply some optimization algorithm to find a good hypothesis. In this case, *J* is <u>piecewise constant</u>, which makes this a difficult problem

Approximating the expected loss by a smooth function

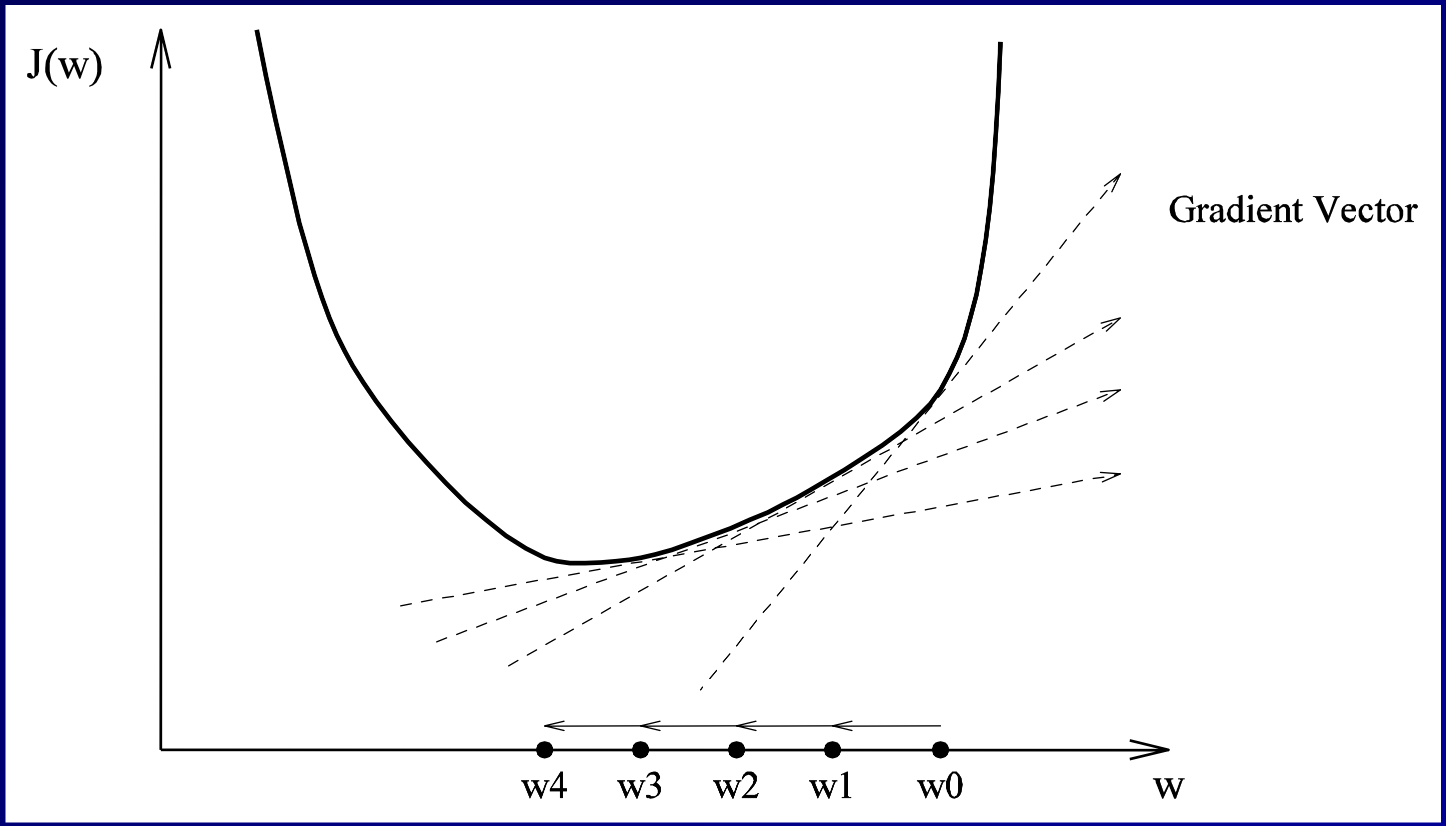
- Simplify the optimization problem by replacing the original objective function by a smooth, differentiable function. For example, consider the *hinge loss*:

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

When $y = 1$



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

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

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

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

Batch Perceptron Algorithm

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

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

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

Repeat until convergence

  For $i = 1$ **to** N **do**

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

    If $(y_i \cdot u_i < 0)$

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

Repeat forever

Accept training example i: $\langle \mathbf{x}_i, y_i \rangle$

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

If $(y_i u_i < 0)$

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

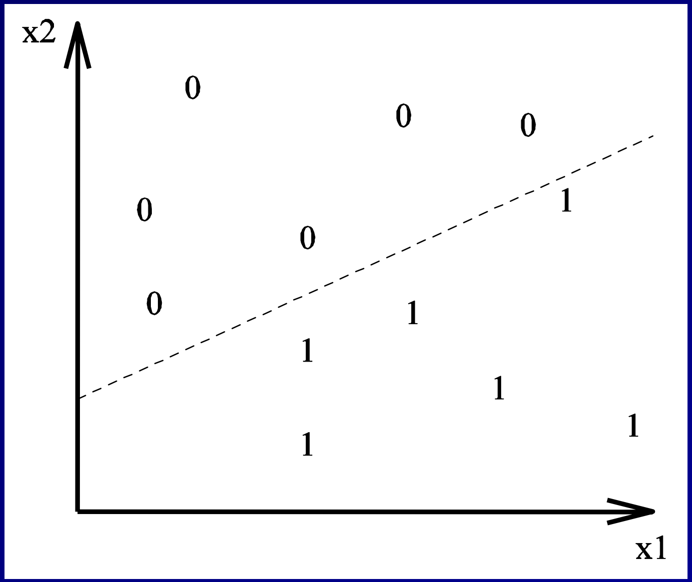
- The learning rate η must decrease to zero in order to guarantee convergence. The online case is known as the Robbins-Munro algorithm. It is guaranteed to converge under the following assumptions:

$$\begin{aligned} \lim_{t \to \infty} \eta_t &= 0 \\ \sum_{t=0}^{\infty} \eta_t &= \infty \\ \sum_{t=0}^{\infty} \eta_t^2 &< \infty \end{aligned}$$

- The learning rate is also called the <u>step size</u>. 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

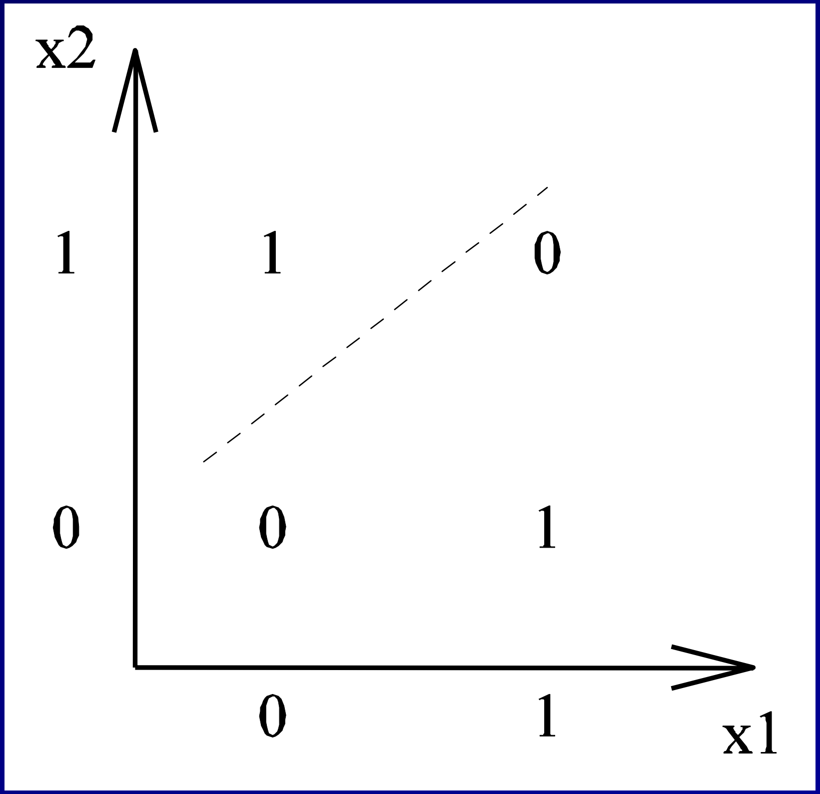
Decision Boundaries

- A classifier can be viewed as partitioning the input space or feature space 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 said to be linearly separable.

Exclusive-OR is Not Linearly Separable



Extending Perceptron to More than Two 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 examples from class $y = k$ as the positive examples and treating the examples from all other classes as negative examples. Then we classify a new data point $\mathbf{x}$ according to

$$\hat{y} = \operatorname*{argmax}_k \mathbf{w}_k \cdot \mathbf{x}.$$

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

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 adjustable parameters. Assume only two classes y = 0 and y = 1, and

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

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

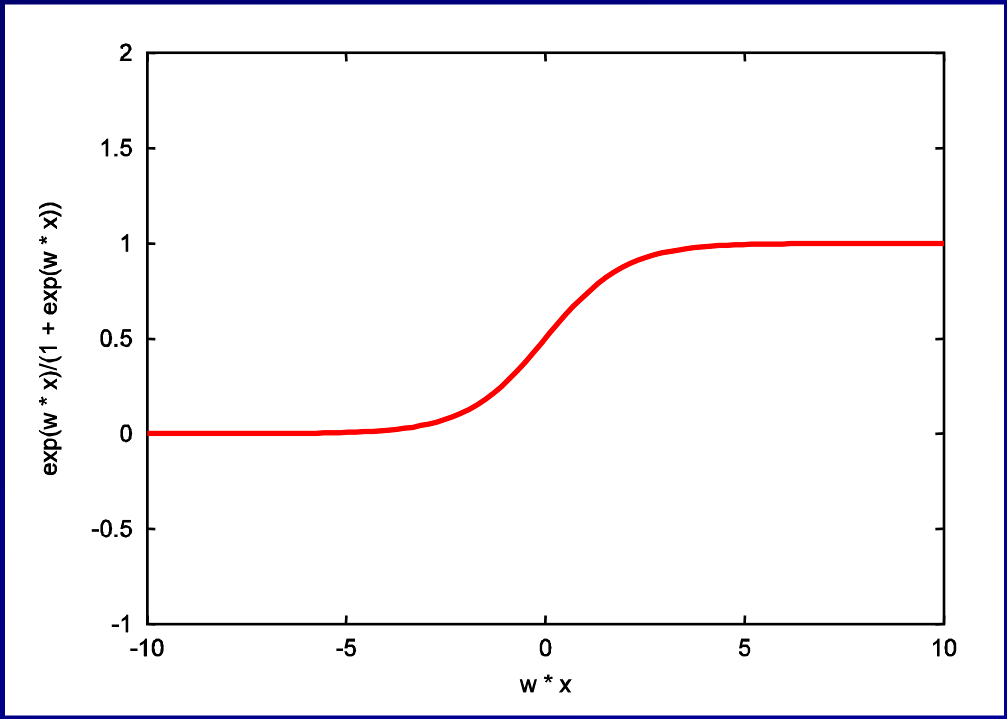
- On the homework, you will show that this is equivalent to

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

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

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 likelihood function. The log likelihood is frequently used as the objective function for learning. It is often written as $\ell(\mathbf{w})$.

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 ($\mathbf{x}_i$,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:

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

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

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial w_j} &= \sum_i \frac{\partial}{\partial w_j} \ell(y_i; \mathbf{x}_i, \mathbf{w}) \\
\frac{\partial}{\partial w_j} \ell(y_i; \mathbf{x}_i, \mathbf{w}) &= \frac{\partial}{\partial w_j} \left((1 - y_i) \log[1 - p_1(\mathbf{x}_i; \mathbf{w})] + y_1 \log p_1(\mathbf{x}_i; \mathbf{w}) \right) \\
&= (1 - y_i) \frac{1}{1 - p_1(\mathbf{x}_i; \mathbf{w})} \left(- \frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) + y_i \frac{1}{p_1(\mathbf{x}_i; \mathbf{w})} \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) \\
&= \left[\frac{y_i}{p_1(\mathbf{x}_i; \mathbf{w})} - \frac{(1 - y_i)}{1 - p_1(\mathbf{x}_i; \mathbf{w})} \right] \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) \\
&= \left[\frac{y_i (1 - p_1(\mathbf{x}_i; \mathbf{w})) - (1 - y_i) p_1(\mathbf{x}_i; \mathbf{w})}{p_1(\mathbf{x}_i; \mathbf{w})(1 - p_1(\mathbf{x}_i; \mathbf{w}))} \right] \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right) \\
&= \left[\frac{y_i - p_1(\mathbf{x}_i; \mathbf{w})}{p_1(\mathbf{x}_i; \mathbf{w})(1 - p_1(\mathbf{x}_i; \mathbf{w}))} \right] \left(\frac{\partial p_1(\mathbf{x}_i; \mathbf{w})}{\partial w_j} \right)
\end{aligned}
$$

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 $\mathbf{w} = (0, 0, 0, 0, \ldots, 0)$ 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)

Logistic Regression Implements a Linear Discriminant Function

- In the 2-class 0/1 loss function case, we should predict ŷ = 1 if

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

- 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:

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

- 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 | $\mathbf{x}$)
- 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