COMS W4701: Artificial Intelligence

Lecture 24: Linear Regression and Classification

Tony Dear, Ph.D.
Department of Computer Science
School of Engineering and Applied Sciences

Today

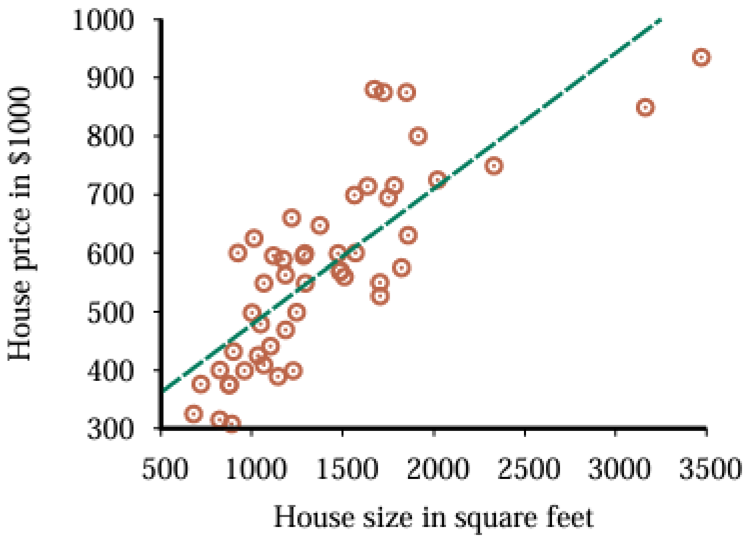
- Linear regression

- Linear classification
- Perceptron learning rule

- Logistic regression
- Gradient descent

Linear Models

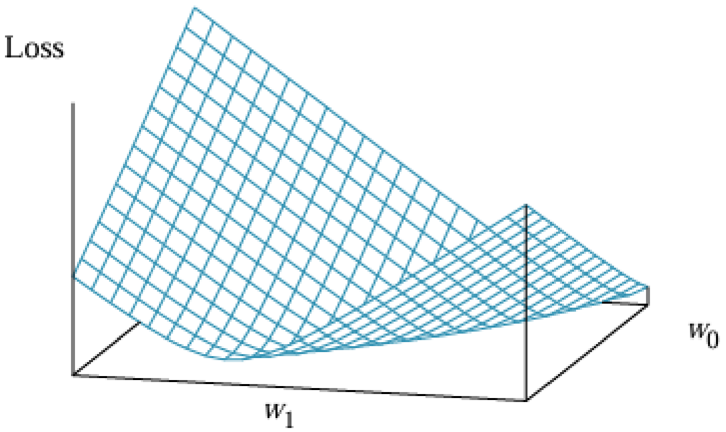
- Consider a supervised learning task in which input data are *feature vectors* $\mathbf{x}_i = (\mathbf{x}_{i1}, \dots, \mathbf{x}_{ip})$ and outputs are *scalars* y_i



- A **linear model** is described by a **weight vector w**:

$$f_{\mathbf{w}}(\mathbf{x}_i) = w_0 + \sum_{j=1}^{p} w_j x_{ij} = \mathbf{w} \cdot (1, \mathbf{x}_i)$$

- Data sets are generally noisy, and it is generally impossible to recover a hypothesis $\hat{f}$ s.t. $\hat{f}(\mathbf{x}_i) = y_i$ for all i



- **Linear regression** tries to find a hypothesis so as to *minimize* the **loss** on the data set

Loss Functions

- A **loss function** $L(y, \hat{y})$ measures the closeness between a predicted value $\hat{y}$ and the actual value y
  - Absolute ($\boldsymbol{L_1}$) loss: $L(y, \hat{y}) = |y - \hat{y}|$
  - Squared ($\boldsymbol{L_2}$) loss: $L(y, \hat{y}) = (y - \hat{y})^2$
  - **0-1** loss: $L(y, \hat{y}) = 1$ if $y \neq \hat{y}$, and 0 otherwise

- The *empirical loss* can be computed over a data set can be computed by taking the sum or the mean of one of the functions above
- L_1 and L_2 losses are useful for real values, 0-1 loss for categorical values

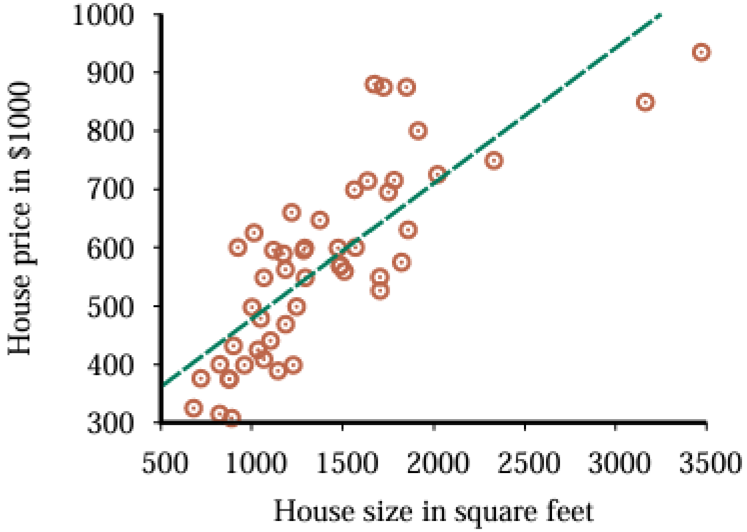
Linear Regression

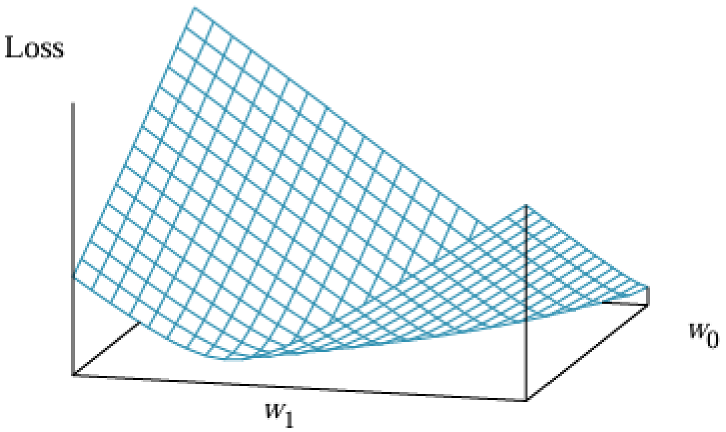
- Given a data set $((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_3))$, we can stack the feature vectors into a $n \times p$ **data matrix** $\mathbf{X}$ and the output values y_i into a n-length **output vector** $\mathbf{y}$

- Given a p-length **weight vector** $\mathbf{w}$, we can compute the *predictions* of all $\mathbf{x}_i$ as $\hat{\mathbf{y}} = \hat{f}_{\mathbf{w}}(\mathbf{X}) = \mathbf{X}\mathbf{w}$

- Linear regression minimizes the *total squared-error loss* $\sum_i (y_i - \hat{y}_i)^2$, and has a closed-form solution!

$$\mathbf{w}^* = \min_{\mathbf{w}} \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$





Linear Classification

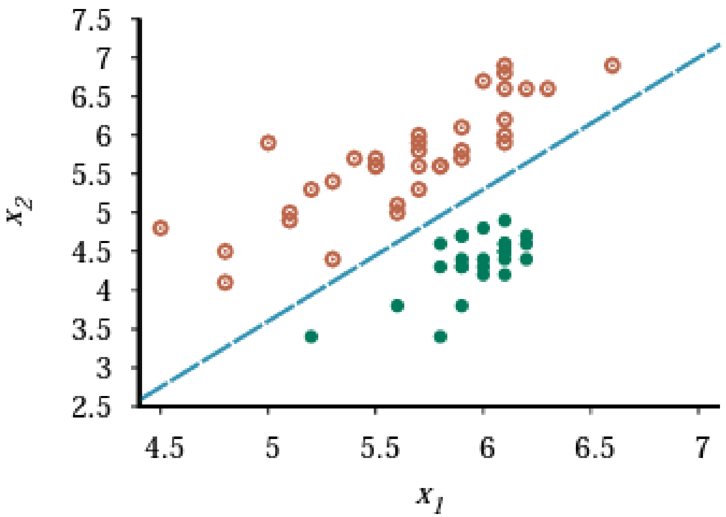
- Now consider a **classification** problem, in which output values y are categorical, e.g. $\{0,1\}$

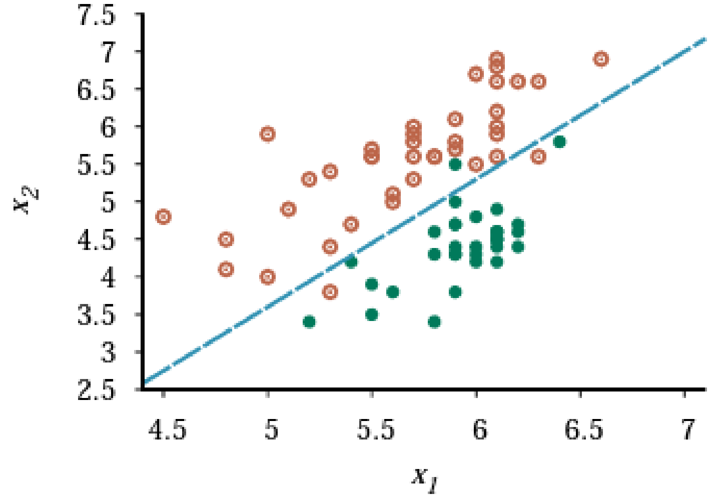
- One way of classifying data is to *separate* them in feature space, e.g. using linear functions

- For binary classification, we can define an **activation function** that composes with the linear function:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1, \text{if } f_{\mathbf{w}}(\mathbf{x}) \geq 0 \\ 0, \text{if } f_{\mathbf{w}}(\mathbf{x}) < 0 \end{cases}$$

- The hyperplane $h_{\mathbf{w}}(\mathbf{x}) = 0$ forms the **decision boundary**



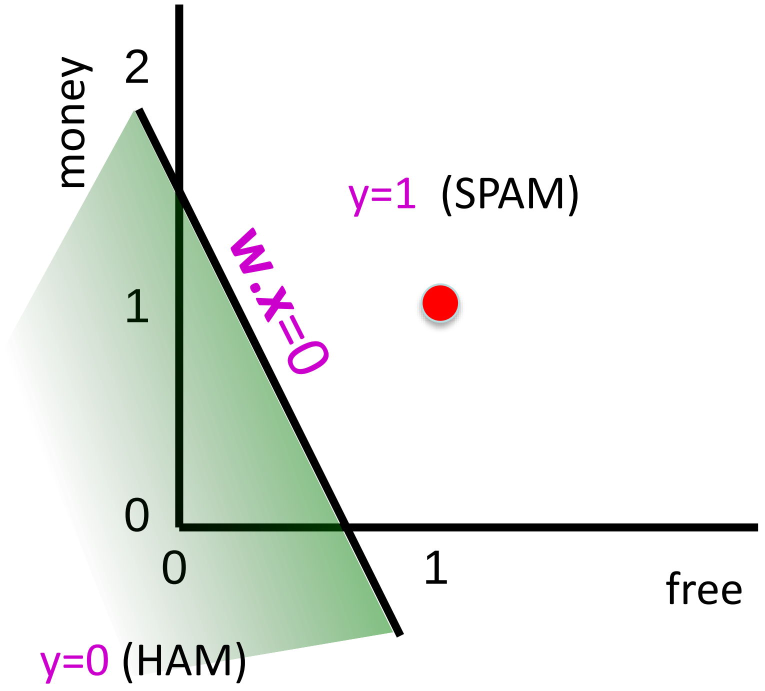


Example: Linear Classifier

- Given an email input, $\mathbf{x} = (Contains?(\text{"free"}), Contains?(\text{"money"}))$
- Want to predict either $y = 1$ (spam) or $y = 0$ (ham)

- Suppose our model (weight vector) is $\mathbf{w} = (-3,4,2)$
- Separating hyperplane is $f_w(\mathbf{x}) = -3 + 4x_1 + 2x_2 = 0$

- "Free money": $h_\mathbf{w}(\mathbf{x}) = h_\mathbf{w}((1,1)) = 1 \rightarrow$ **spam**
- "Free food": $h_\mathbf{w}(\mathbf{x}) = h_\mathbf{w}((1,0)) = 1 \rightarrow$ **spam**
- "No money": $h_\mathbf{w}(\mathbf{x}) = h_\mathbf{w}((0,1)) = 0 \rightarrow$ **ham**
- "how are you": $h_\mathbf{w}(\mathbf{x}) = h_\mathbf{w}((0,0)) = 0 \rightarrow$ **ham**



Perceptron Learning Rule

- No closed-form solution! Activation function makes the problem nonlinear
- *Iterative* method to learn $\mathbf{w}$: Classify each training data instance, and update $\mathbf{w}$ if prediction is incorrect

Initialize weights $\mathbf{w}$ (e.g., all 0) and learning rate α

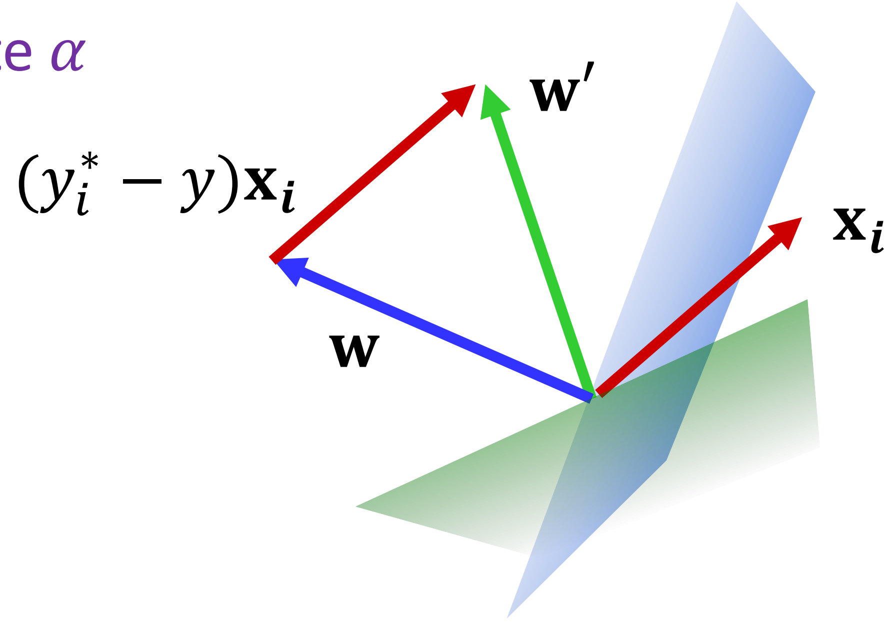
While not converged:

    For each training instance $\mathbf{x}_i$:

        Predict class y using current weights $\mathbf{w}$

        If incorrect ($y \neq y_i^*$), update weights:

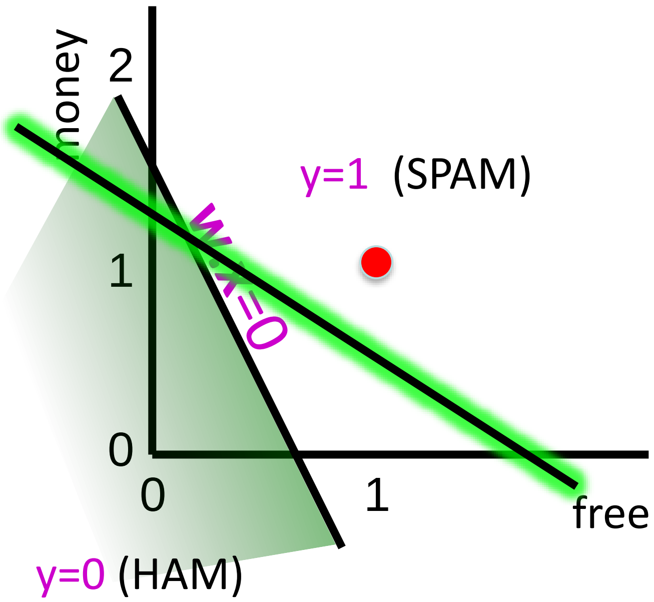
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i^* - y)(1, \mathbf{x}_i)$$



Example: Perceptron Learning Rule

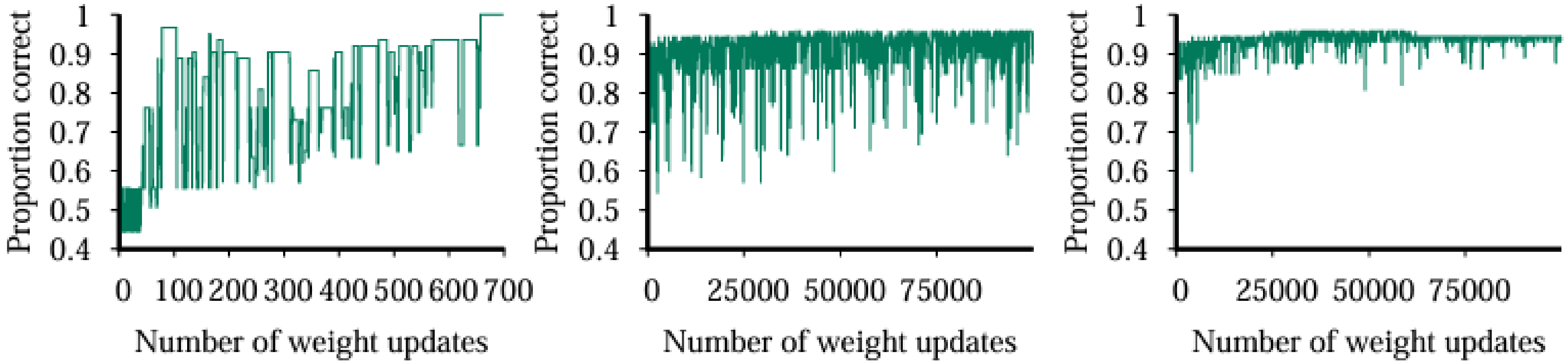
- Suppose we have the following training data:
- $\big((0,1), spam\big), \big((1,0), ham\big), \big((0,0), ham\big)$
- We currently have $\mathbf{w} = (-3,4,2)$ and we use $\alpha = 1$

- $h_{\mathbf{w}}(\mathbf{x}_1) = 0 \rightarrow$ **ham** $\rightarrow \mathbf{w} = \mathbf{w} + (1,0,1) = (-2,4,3)$
- $h_{\mathbf{w}}(\mathbf{x}_2) = 1 \rightarrow$ **spam** $\rightarrow \mathbf{w} = \mathbf{w} - (1,1,0) = (-3,3,3)$
- $h_{\mathbf{w}}(\mathbf{x}_3) = 0 \rightarrow$ **ham**; correct, no change
- $h_{\mathbf{w}}(\mathbf{x}_1) = 1 \rightarrow$ **spam**; correct, no change
- $h_{\mathbf{w}}(\mathbf{x}_2) = 1 \rightarrow$ **spam**$\rightarrow \mathbf{w} = \mathbf{w} - (1,1,0) = (-4,2,3)$
- … until all data classified correctly



Perceptron Convergence

- For linearly separable data, the perceptron is *guaranteed* to find a linear separator (though convergence may not be monotonic)
- For nonseparable data, perceptron will converge to a minimum-error solution if α is decayed as $O\left(\frac{1}{t}\right)$



Separable data | Nonseparable data | Nonseparable data with decaying α

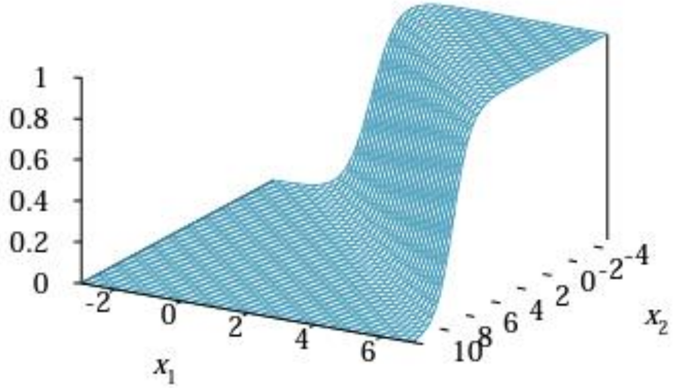
Logistic Model

- Hard threshold causes training to be discontinuous and unstable
- Classifier is not informed by the distance between data and boundary
- It is possible for the perceptron to learn *barely separating* boundaries

- Idea: Replace the *hard* threshold with a *soft* threshold
- Use a **logistic** (sigmoid) function for h:

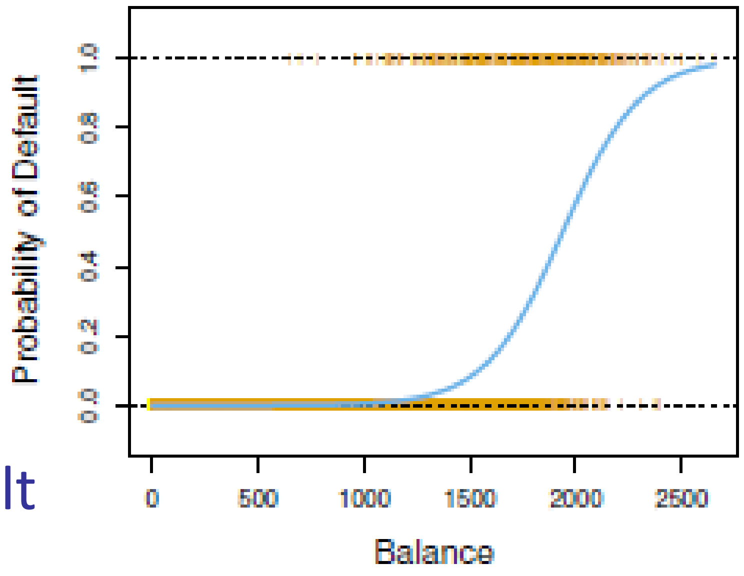
$$h_{\mathbf{w}}(\mathbf{x}) = \sigma(f_{\mathbf{w}}(\mathbf{x})) = \frac{1}{1 + \exp(-f_{\mathbf{w}}(\mathbf{x}))}$$

- Outputs $[0,1]$ can be interpreted as probabilities!
- Large values of $f_{\mathbf{w}}(\mathbf{x})$ push $h_{\mathbf{w}}(\mathbf{x})$ closer to 1
- On the decision boundary $f_{\mathbf{w}}(\mathbf{x}) = 0$, we have $h_{\mathbf{w}}(\mathbf{x}) = 0.5$

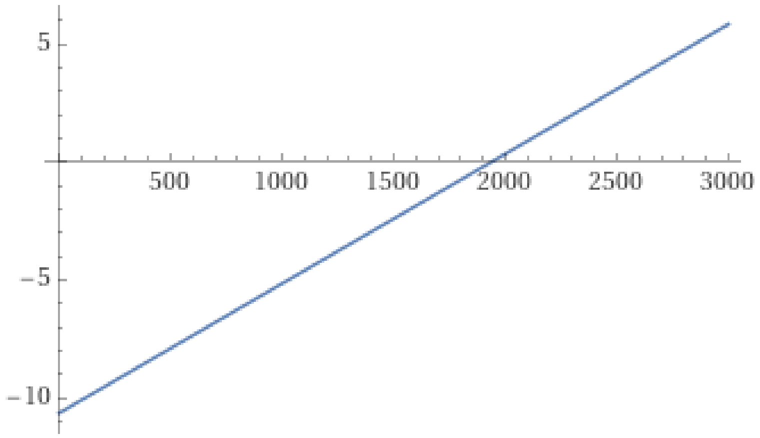


Example: Logistic Model

- Weights: $\mathbf{w} = (-10.65, .0055)$
- Logistic model: $h_{\mathbf{w}}(x) = \frac{1}{1+\exp(-(-10.65+.0055x))}$
- We can also say $\Pr(y = 1|x) = h_{\mathbf{w}}(x)$



- $\Pr(\text{default}|\text{balance} = 1000) = 0.00576$ -> predict no default
- $\Pr(\text{default}|\text{balance} = 2000) = 0.586$ -> predict default



- Note that $\log \frac{\Pr(Y=1|x)}{\Pr(Y=0|x)} = \log \frac{h_{\mathbf{w}}(x)}{1-h_{\mathbf{w}}(x)} = w_0 + wx$
- The **log odds** is exactly the linear function $f_{\mathbf{w}}(x)$

Logistic Regression

- Like the hard threshold, no closed form solution for learning weights
- Unlike the hard threshold, there is a well-defined iterative approach based on *maximizing* the *likelihood* of our data

- For $(\mathbf{x}_i, y_i)$, maximize the probability $\Pr(y_i|\mathbf{x}_i) = h_{\mathbf{w}}(\mathbf{x}_i)$ if $y_i = 1$, and alternatively $1 - h_{\mathbf{w}}(\mathbf{x}_i)$ if $y_i = 0$
- In other words, maximize the likelihood

$$\Pr(y_i|\mathbf{x}_i) = h_{\mathbf{w}}(\mathbf{x}_i)^{y_i}(1 - h_{\mathbf{w}}(\mathbf{x}_i))^{1-y_i}$$

Log Likelihood

- We will write the *negative log likelihood* to obtain easier derivatives:

$$-\log h_{\mathbf{w}}(\mathbf{x}_i)^{y_i}\big(1-h_{\mathbf{w}}(\mathbf{x}_i)\big)^{1-y_i} = -\big(y_i \log h_{\mathbf{w}}(\mathbf{x}_i) + (1-y_i)\log\big(1-h_{\mathbf{w}}(\mathbf{x}_i)\big)\big)$$

- We will *minimize* the average negative log likelihood over all data:

$$L(\mathbf{w}) = -\frac{1}{n}\sum_{i=1}^{n}\big(y_i \log h_{\mathbf{w}}(\mathbf{x}_i) + (1-y_i)\log\big(1-h_{\mathbf{w}}(\mathbf{x}_i)\big)\big)$$

- We will need the partial derivatives of L wrt each of its weights w_i:

$$\frac{\partial L}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n}\big(h_{\mathbf{w}}(\mathbf{x}_i) - y_i\big) \qquad \frac{\partial L}{\partial w_i} = \frac{1}{n}\sum_{i=1}^{n}\big(h_{\mathbf{w}}(\mathbf{x}_i) - y_i\big)\,\mathbf{x}_i$$

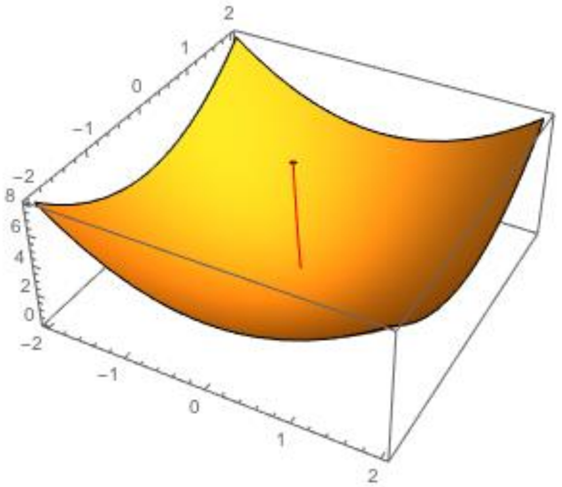
Gradient Descent

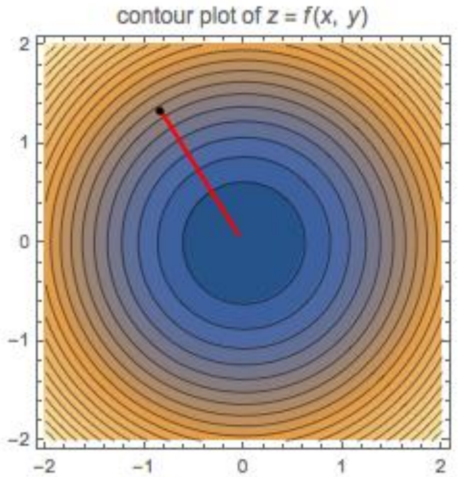
- We are *searching* for a weight vector $\mathbf{w}$ that minimizes L
- The **gradient** of L is a *vector* of partial derivatives wrt all w_j

$$\frac{\partial L}{\partial \mathbf{w}} = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \dots, \frac{\partial L}{\partial w_p}\right)$$

- Indicates magnitude and *direction* of largest increase in L
- The value of L *decreases* fastest along direction of $-\frac{\partial L}{\partial \mathbf{w}}$

- **Gradient descent:** Initialize the configuration $\mathbf{w}$, and repeatedly update it as $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}}$ until convergence





Stochastic Gradient Descent

- $\frac{\partial L}{\partial \mathbf{w}}$ is computed by summing classification "errors" over all data:

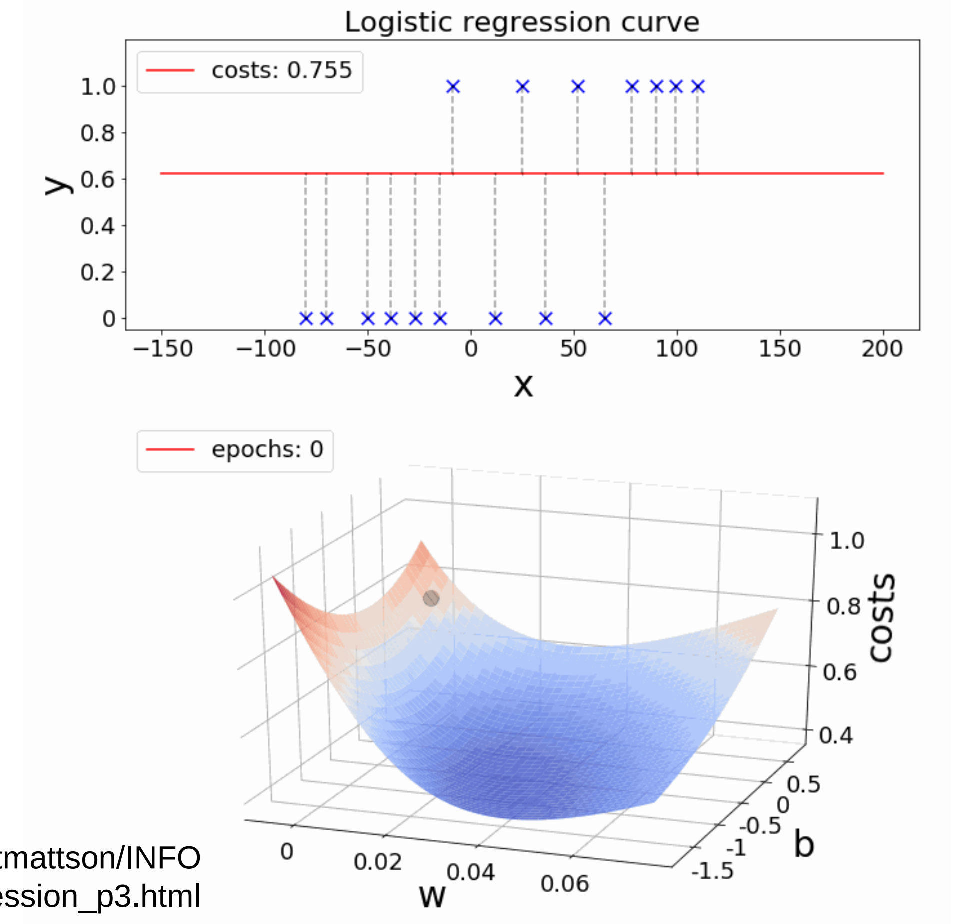
$$\frac{\partial L}{\partial \mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)\begin{bmatrix}1\\ \mathbf{x}_i\end{bmatrix}$$

- **Batch gradient descent** thus computes a single weight vector update per pass through the data set (**epoch**), opposite behavior of perceptron (update for each *data instance*)

- **Stochastic gradient descent** *approximates* $\frac{\partial L}{\partial \mathbf{w}}$ by selecting or sampling a **minibatch** of size m from the data set to compute gradient and update weights
- One epoch will thus see $\frac{n}{m}$ weight updates, allowing for faster convergence

SGD for Logistic Regression

- Given data set $\mathbf{d} = \big((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\big)$, minibatch size m, learning rate α



- Initialize weights $\mathbf{w}$, e.g. randomly
- **Until $\mathbf{w}$ has converged:**
  - Sample minibatch MB from $\mathbf{d}$
  - $grad = \mathbf{0}$
  - **For** $(\mathbf{x}, y) \in MB$:
    - $grad = grad - \big(y - h_{\mathbf{w}}(\mathbf{x})\big)\begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix}$
  - $\mathbf{w} = \mathbf{w} - \frac{\alpha}{m} \cdot grad$

https://facultystaff.richmond.edu/~tmattson/INFO303/logisticregression/logisticregression_p3.html

Multinomial Logistic Regression

- If we have K classes, then we can use the **softmax** function
- Learn a weight vector $\mathbf{w}_k$ for each class k, which can be used to compute the probability of each class given an input $\mathbf{x}$

$$\Pr(Y = k|\mathbf{x}) = \frac{\exp\left(\mathbf{w}_k \cdot (1, \mathbf{x})\right)}{\sum_{i=1}^{K} \exp\left(\mathbf{w}_i \cdot (1, \mathbf{x})\right)}$$

- Classifier can output the most likely class or the distribution itself

- The points where $\Pr(Y = k_1|\mathbf{x}) = \Pr(Y = k_2|\mathbf{x})$ form a linear decision boundary between the two classes k_1 and k_2

Summary

- Linear models compute linear combinations of feature vectors
- Closed-form solution for weight vector in the regression problem

- Linear classifiers find a decision boundary in feature space
- Can apply an activation function to the linear function output
- The perceptron learning rule can be used for hard thresholds

- Logistic regression applies a *soft* threshold to the weighted feature
- Can be solved by gradient descent of likelihood function