Machine Learning

Topic: Linear Discriminants

Bryan Pardo, EECS 349 Machine Learning, 2015

Thanks to Mark Cartwright for his contributions to these slides
Thanks to Alpaydin, Bishop, and Duda/Hart/Stork for images and ideas

Discrimination Learning Task

There is a set of possible examples $X = \{\mathbf{x_1}, ...\mathbf{x_n}\}$

Each example is a **vector** of k **real valued attributes**

$$\mathbf{x}_i = < x_{i1}, \ldots, x_{ik} >$$

A target function maps *X* onto some **categorical variable** *Y*

$$f : X \rightarrow Y$$

The DATA is a set of tuples <example, response value>

$$\{< \mathbf{x}_1, y_1 >, \ldots < \mathbf{x_n}, y_n >\}$$

Find a hypothesis ***h*** such that...

$$\forall \mathbf{x}, h(\mathbf{x}) \approx f(\mathbf{x})$$

Reminder about notation

- $\mathbf{x}$ is a vector of attributes $<x_1, x_2, \ldots x_k>$
- $\mathbf{w}$ is a vector of weights $<w_1, w_2, \ldots w_k>$
- Given this…

$$g(x) = w_0 + w_1 x_1 + w_2 x_2 \ldots + w_k x_k$$

- We can notate it with linear algebra as

$$g(x) = w_0 + \mathbf{w}^{\mathbf{T}}\mathbf{x}$$

It is more convenient if…

- $g(x) = w_0 + \mathbf{w}^\mathbf{T}\mathbf{x}$ is ALMOST what we want, but that pesky offset w_0 is not in the linear algebra part yet.

- If we define $\mathbf{w}$ to include w_0 and $\mathbf{x}$ to include an x_0 that is always 1, now…

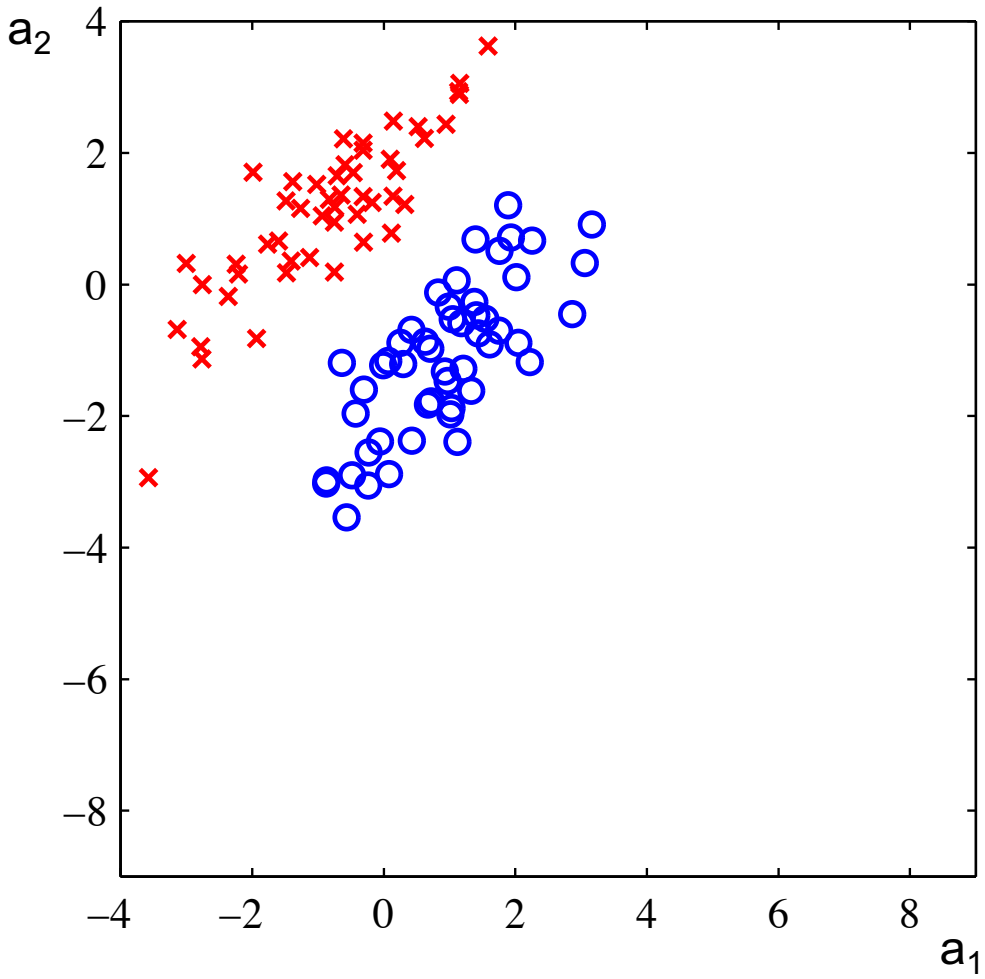
  $\mathbf{x}$ is a vector of attributes $<1, x_1, x_2, \ldots x_k>$

  $\mathbf{w}$ is a vector of weights $<w_0, w_1, w_2, \ldots w_k>$

- This lets us notate things as…

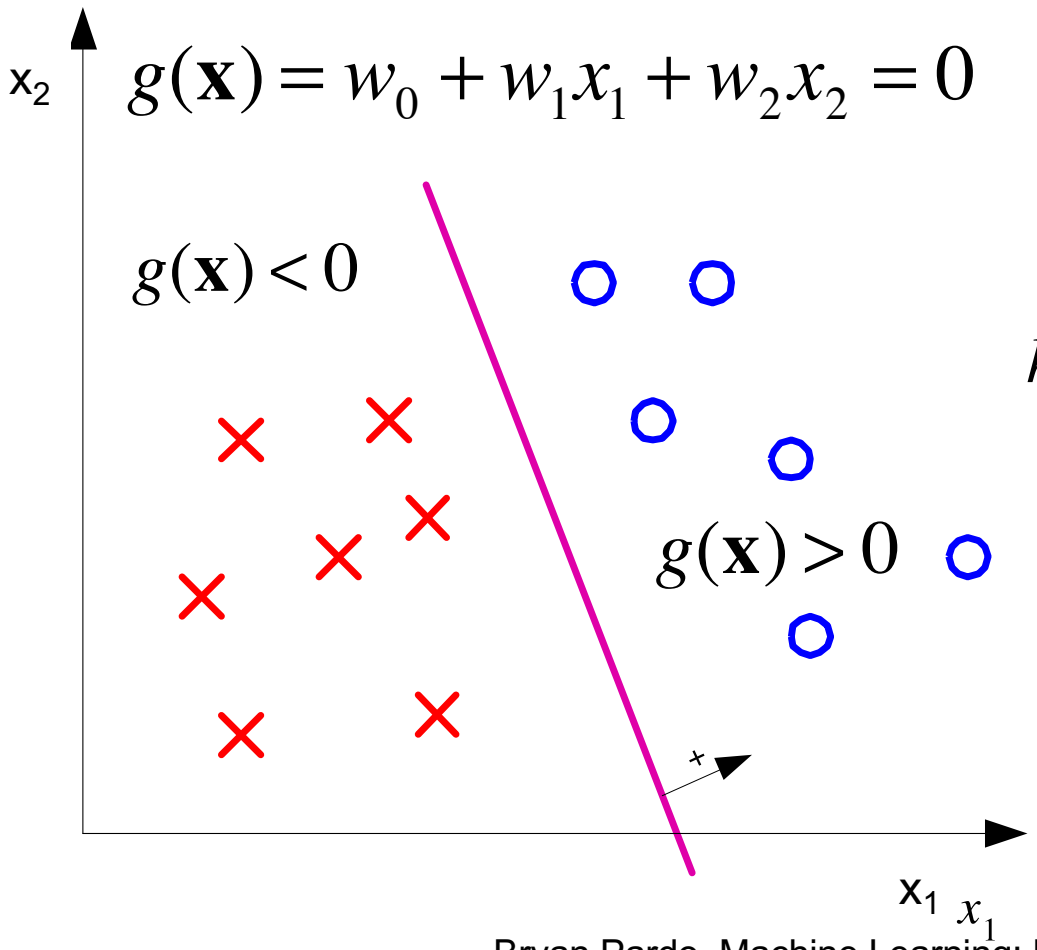
$$g(x) = \mathbf{w}^\mathbf{T}\mathbf{x}$$

Visually: Where to draw the line?



Two-Class Classification

$g(\mathbf{x}) = 0$ defines a decision boundary that splits the space in two

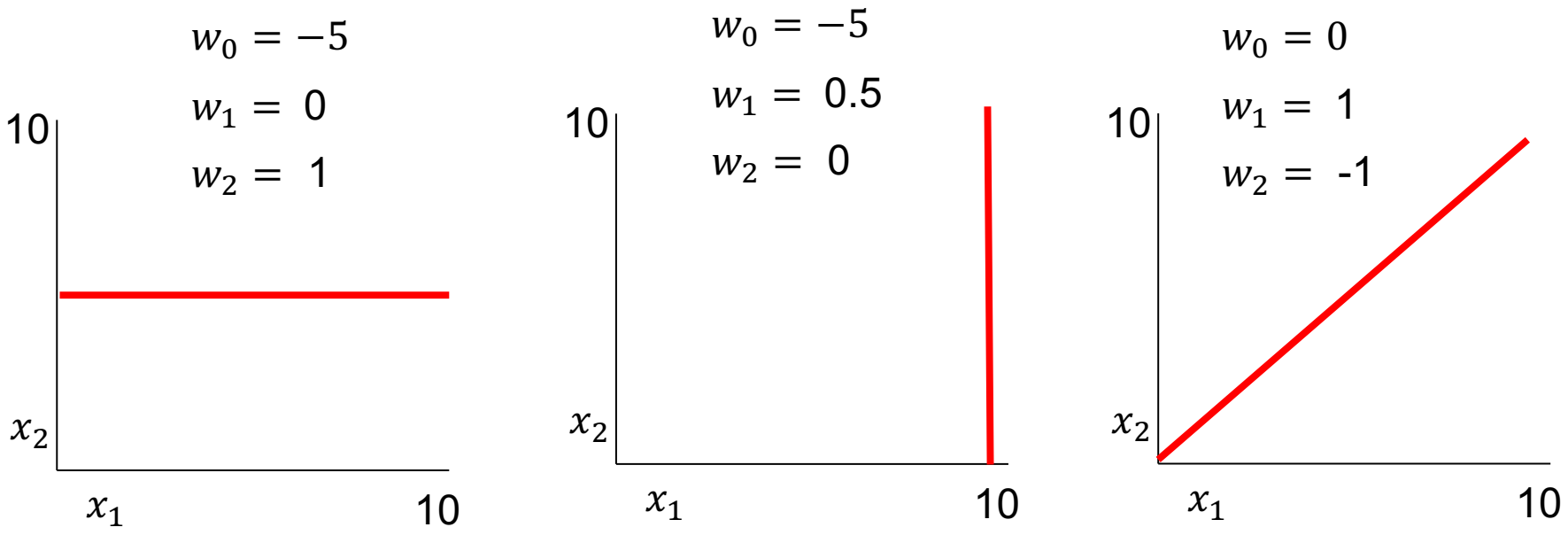


If a line exists that does this without error, the classes are *linearly separable*

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

Example 2-D decision boundaries

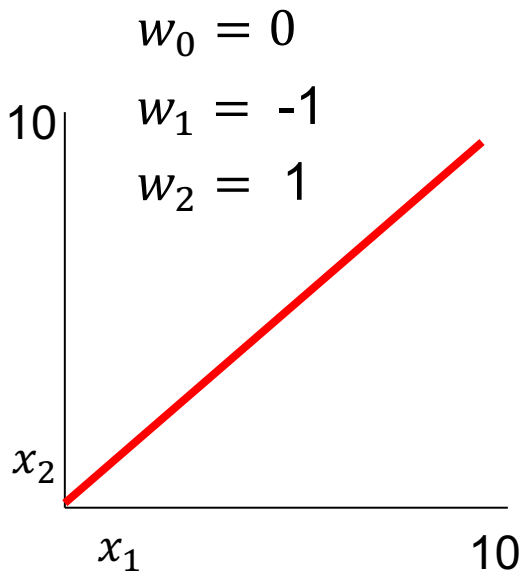
$$0 = g(x) = w_0 + w_1 x_1 + w_2 x_2 = \mathbf{w}^T \mathbf{x}$$

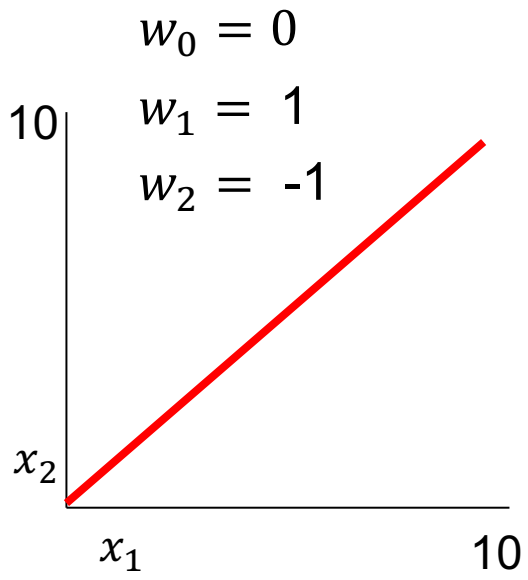


What's the difference?

$$0 = g(x) = w_0 + w_1 x_1 + w_2 x_2 = \mathbf{w}^T \mathbf{x}$$

What's the difference between these two?





Loss/Objective function

- To train a model (e.g. learn the weights of a useful line) we define a measure of the ”goodness” of that model. (e.g. the number of misclassified points).

- We make that measure a function of the parameters of the model (and the data).

- This is called a loss function, or an objective function.

- We want to minimize the loss (or maximize the objective) by picking good model parameters.

Classification via regression

- Linear regression's loss function is the the squared distance from a data point to the line, summed over all data points.

- The line that minimizes this function can be calculated by applying a simple formula.

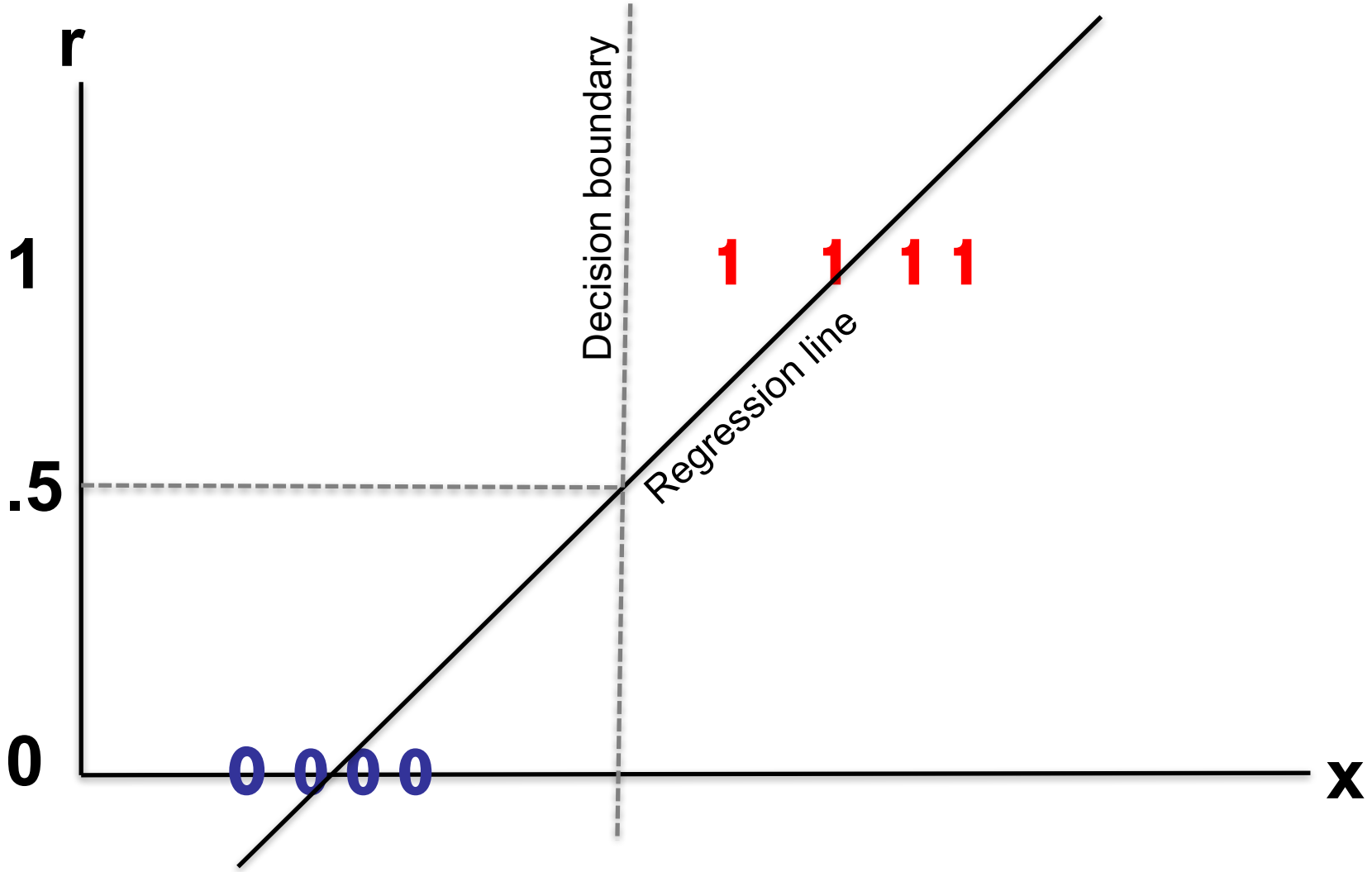
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

- Can we find a decision boundary in one step, by just repurposing the math we used for finding a regression line?

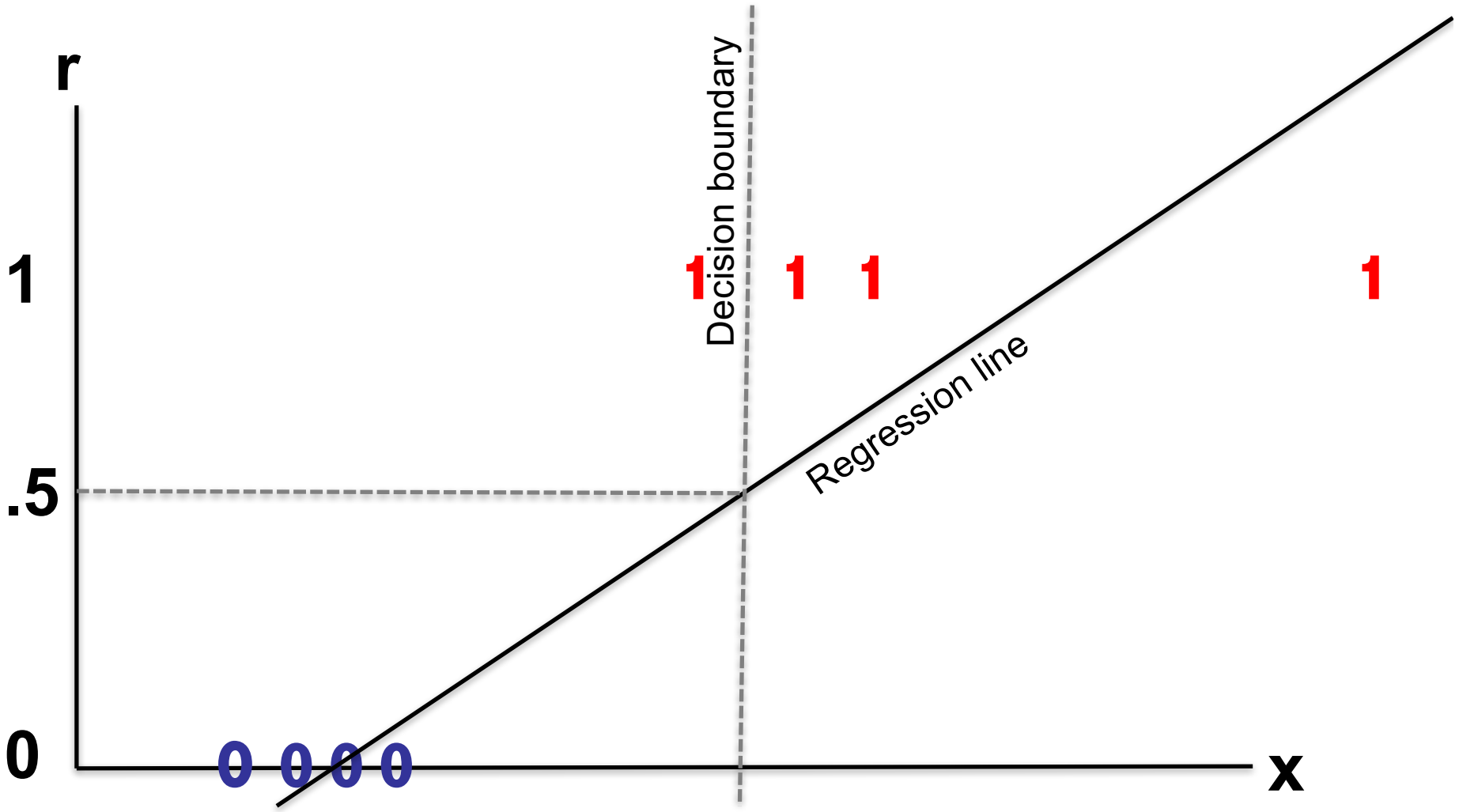
Classification via regression

- Label each class by a number
- Call that number the response variable
- Analytically derive a regression line
- Round the regression output to the nearest label number

An example



What happens now?



Classification via regression take-away

- Closed form solution: just hand me the data and I can apply that simple formula for getting the regression line.
- Very sensitive to outliers
- What's the natural mapping from categories to the real numbers?
- Not used in practice (too finicky)

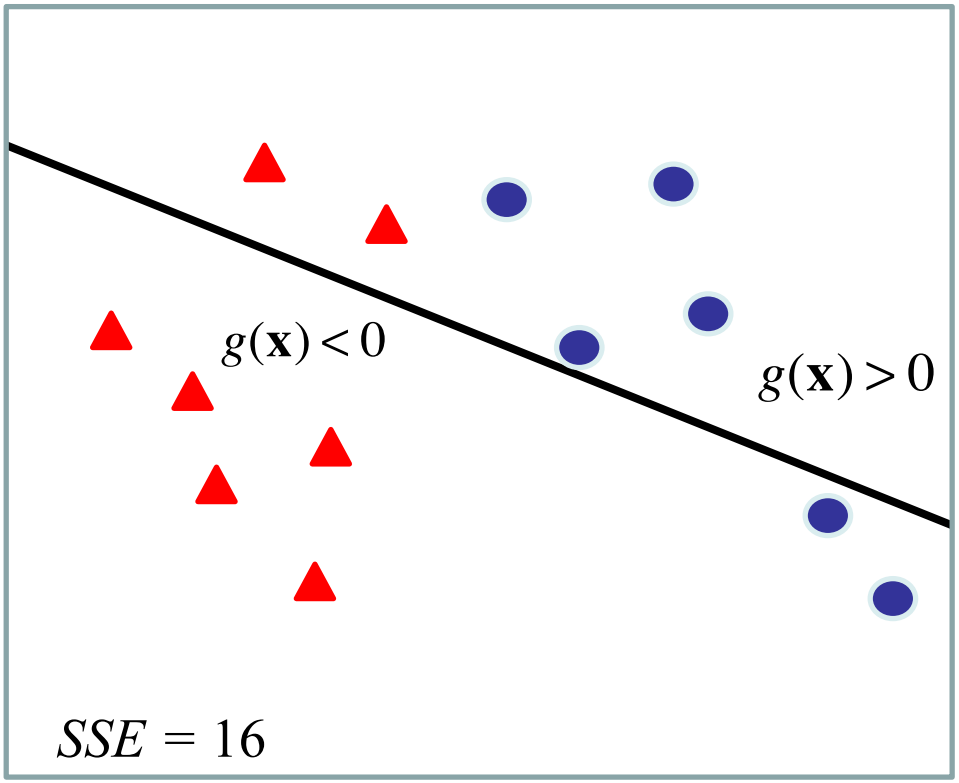
What can we do instead?

- Let's define an objective (aka "loss") function that directly measures the thing we want to get right

- Then let's try and find the line that minimizes the loss.

- How about basing our loss function on the number of misclassifications?

sum of squared errors (SSE)

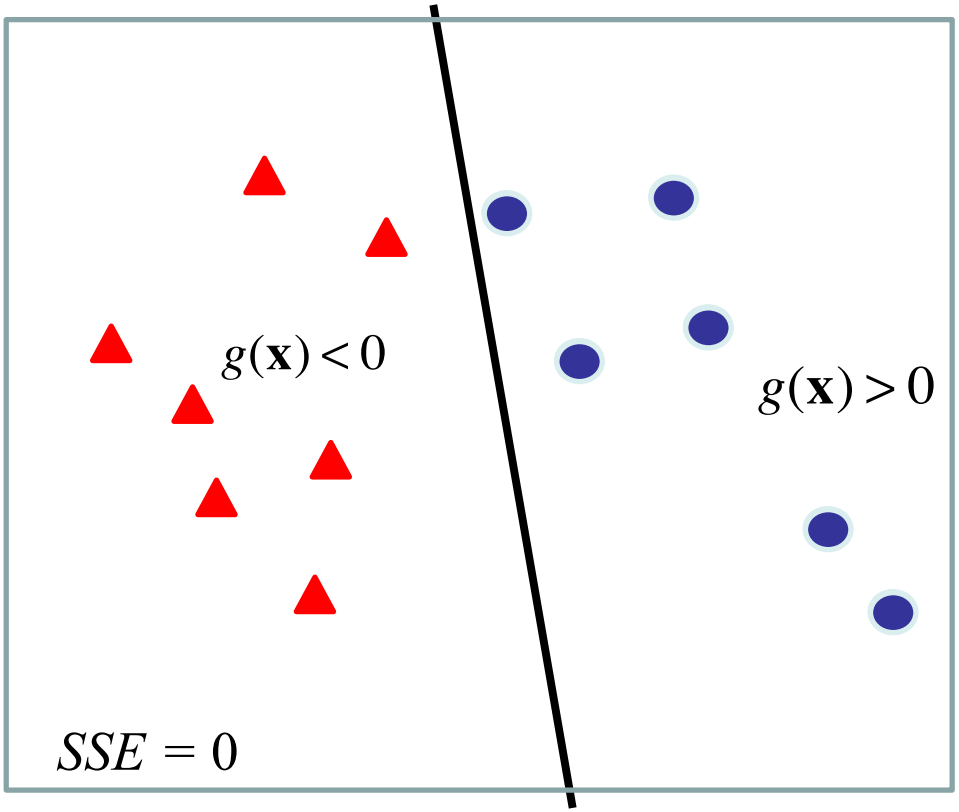


$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$SSE = \sum_i^n (y_i - h(\mathbf{x}_i))^2$$

sum of squared errors (SSE)



$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$SSE = \sum_i^n (y_i - h(\mathbf{x}_i))^2$$

No closed form solution!

- For many objective functions, (e.g. the one on the previous slide) we can't do a proof to find a formula to to get the best model parameters, like we could with regression.

- This means we have to try various guesses for what the weights should be and try them out.

- Let's look at the perceptron approach.

Let's learn a decision boundary

- We'll do 2-class classification
- We'll learn a linear decision boundary

$$0 = g(x) = \mathbf{w}^{\mathbf{T}}\mathbf{x}$$

- Things on each side of 0 get their class labels according to the sign of what g(x) outputs.

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

- We will use the Perceptron algorithm.

Defining our goal

D is our data, consisting of training examples $< \mathbf{x}, y >$. Remember y is the true label (drawn from {1,-1} and x is the thing being labeled.

Our goal : make $(\mathbf{w}^T \mathbf{x}) y > 0$ for all $< \mathbf{x}, y > \in D$

Think about why this is the goal.

An example.

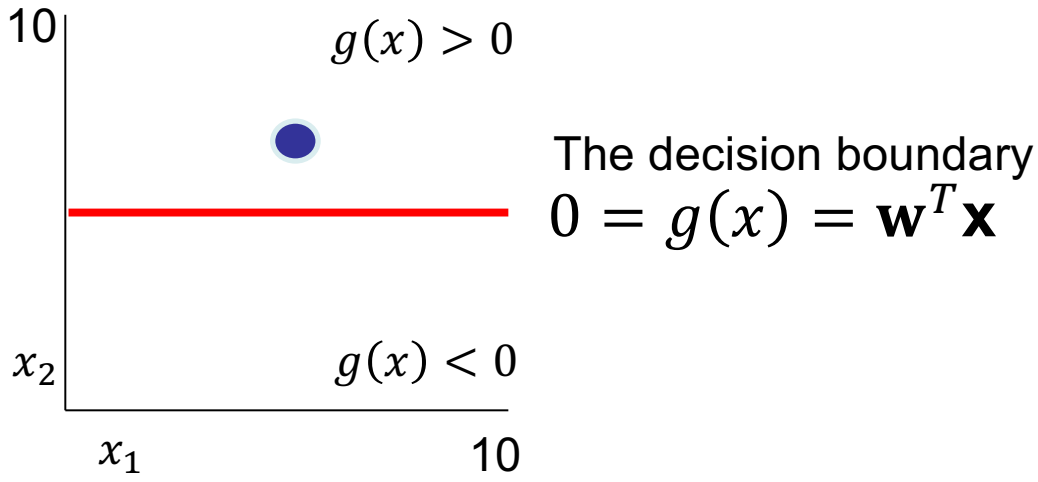
A training data point

$\mathbf{x} = [x_0, x_1, x_2] = [1,5,7]$

y = 1

Our current weights

$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$



The decision boundary

$0 = g(x) = \mathbf{w}^T\mathbf{x}$

$$(\mathbf{w}^T\mathbf{x})y = [-5,0,1]^T[1,5,7]1 = 2$$

Therefore, the line doesn't need to move to correctly classify the blue circle point.

An example.

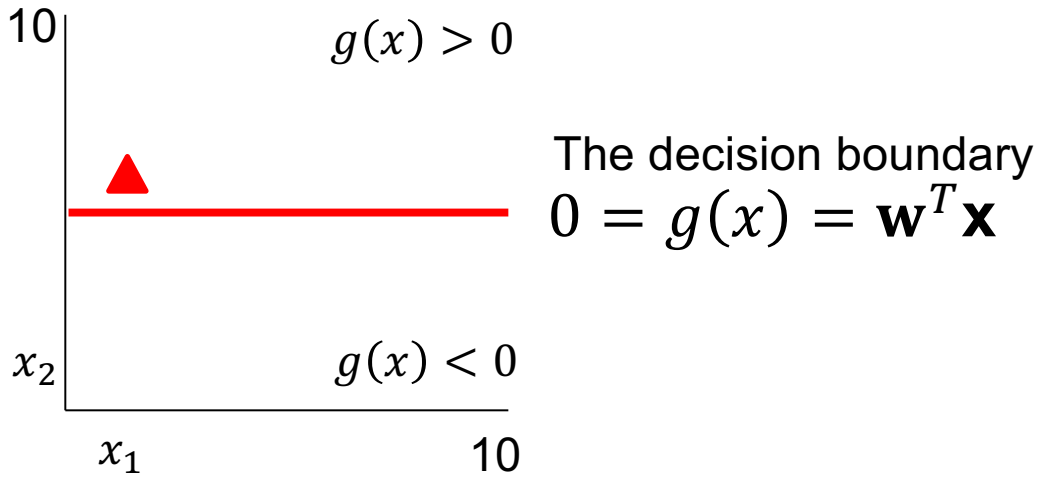
Another training data point

$\mathbf{x} = [x_0, x_1, x_2] = [1,2,6]$

y = -1

Our current weights

$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$



$(\mathbf{w}^T\mathbf{x})y = [-5,0,1]^T[1,2,6](-1) = (-5+6)(-1) = -1$

Therefore, the line DOES need to move to correctly classify the red triangle point.

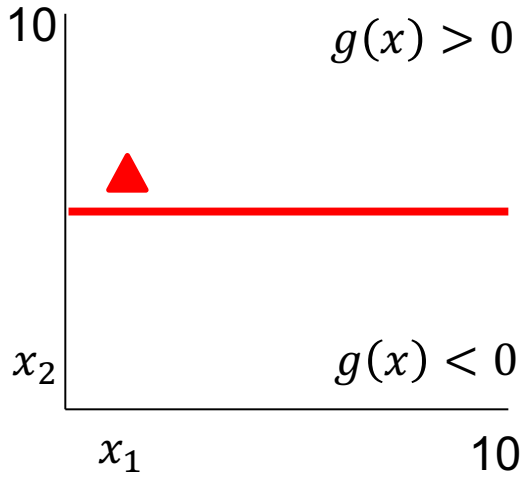
Moving the line

Our misclassified point

$\mathbf{x} = [x_0, x_1, x_2] = [1,2,6]$
y = -1

Our current weights

$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$



The decision boundary

$0 = g(x) = \mathbf{w}^T\mathbf{x}$

Let's update the line by doing $\mathbf{w} = \mathbf{w} + \mathbf{x}(y)$.

Moving the line

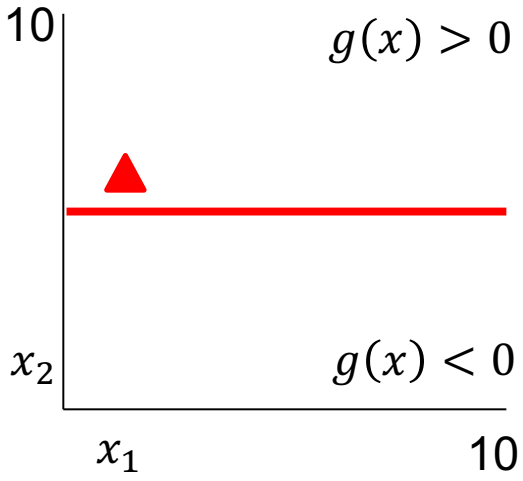
Our misclassified point

$\mathbf{x} = [x_0, x_1, x_2] = [1,2,6]$

y = -1

Our current weights

$\mathbf{w} = [w_0, w_1, w_2] = [-5, 0, 1]$



The decision boundary

$0 = g(x) = \mathbf{w}^T\mathbf{x}$

Let's update the line by doing $\mathbf{w} = \mathbf{w} + \mathbf{x}(y)$.

$$\mathbf{w} = \mathbf{w} + \mathbf{x}(y) = [-5,0,1] + [1,2,6](-1)$$
$$= [-6,-2,-5]$$

Now what ?

- What does the decision boundary look like when **w**$= [-6, -2, -5]$? Does it misclassify the blue dot now?

- What if we update it the same way, each time we find a misclassified point?

- Could this approach be used to find a good separation line for a lot of data?

Perceptron Algorithm

The decision boundary

$$0 = g(x) = \mathbf{w}^{\mathbf{T}}\mathbf{x}$$

The classification function

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$m = |D| = size\ of\ data\ set$$

The weight update algorithm

$\mathbf{w} = \ some\ random\ setting$

Do

$k = (k+1) \bmod (m)$

if $h(\mathbf{x}_k) ! = y_k$

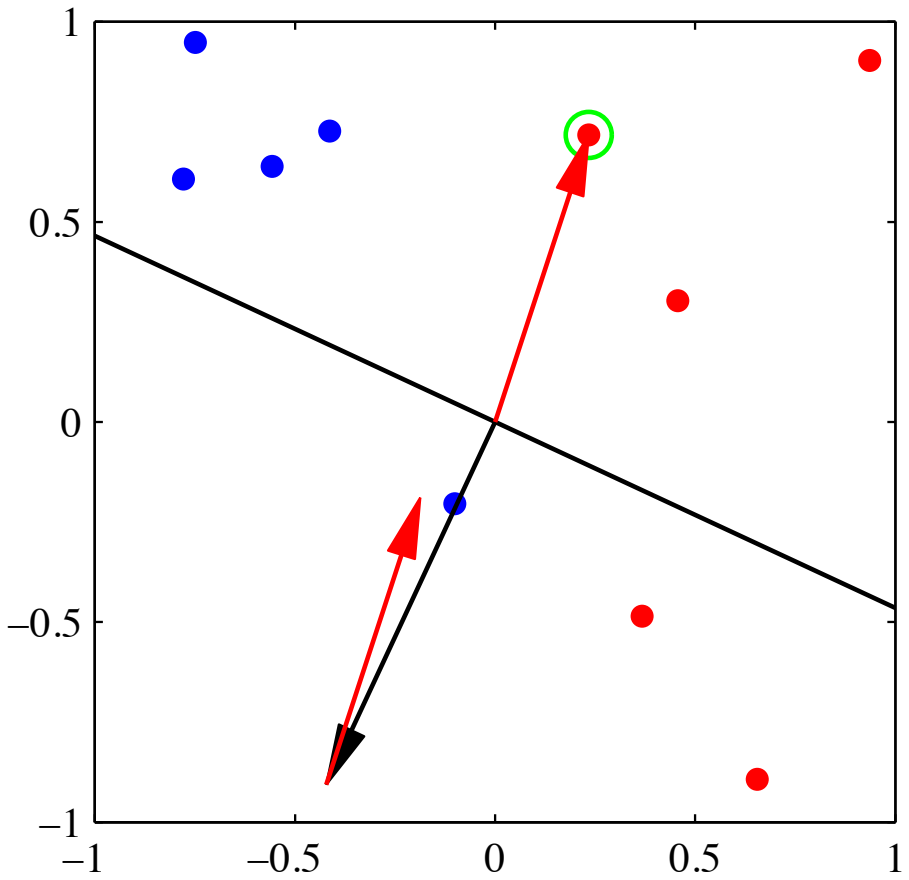
$\mathbf{w} = \mathbf{w} + \mathbf{x}y$

Until $\forall k,\ g(\mathbf{x}_k) = y_k$

Warning: Only gauranteed to terminate if classes are linearly separable!

Perceptron Algorithm

- Example:

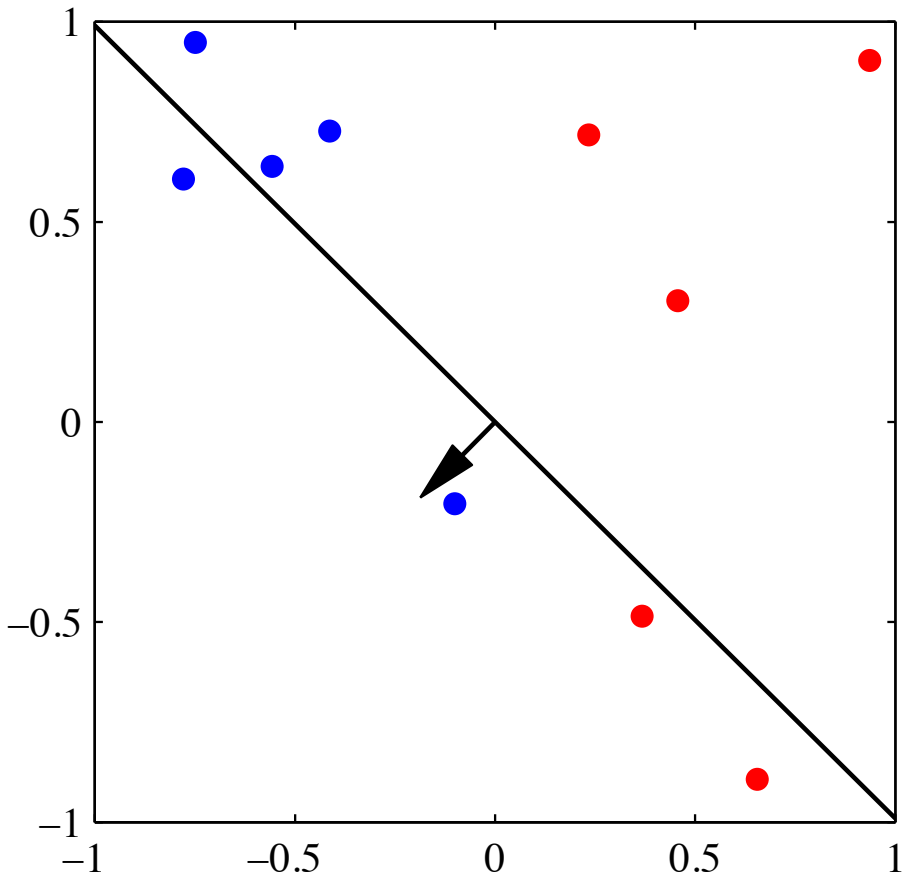


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

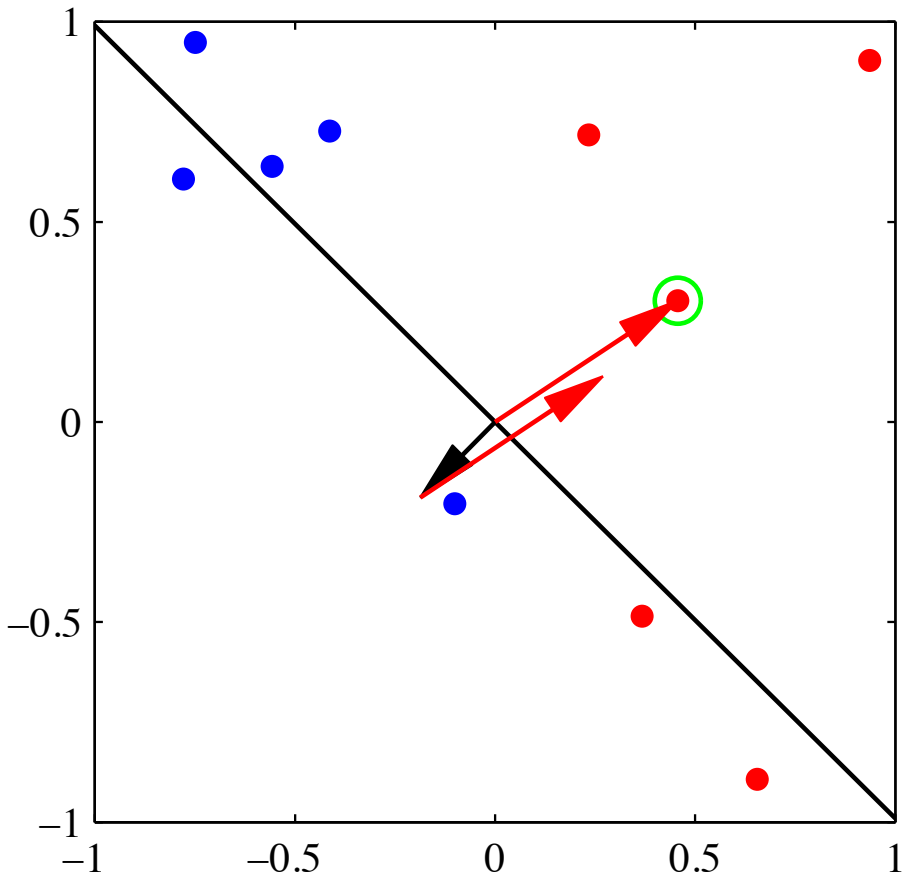


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

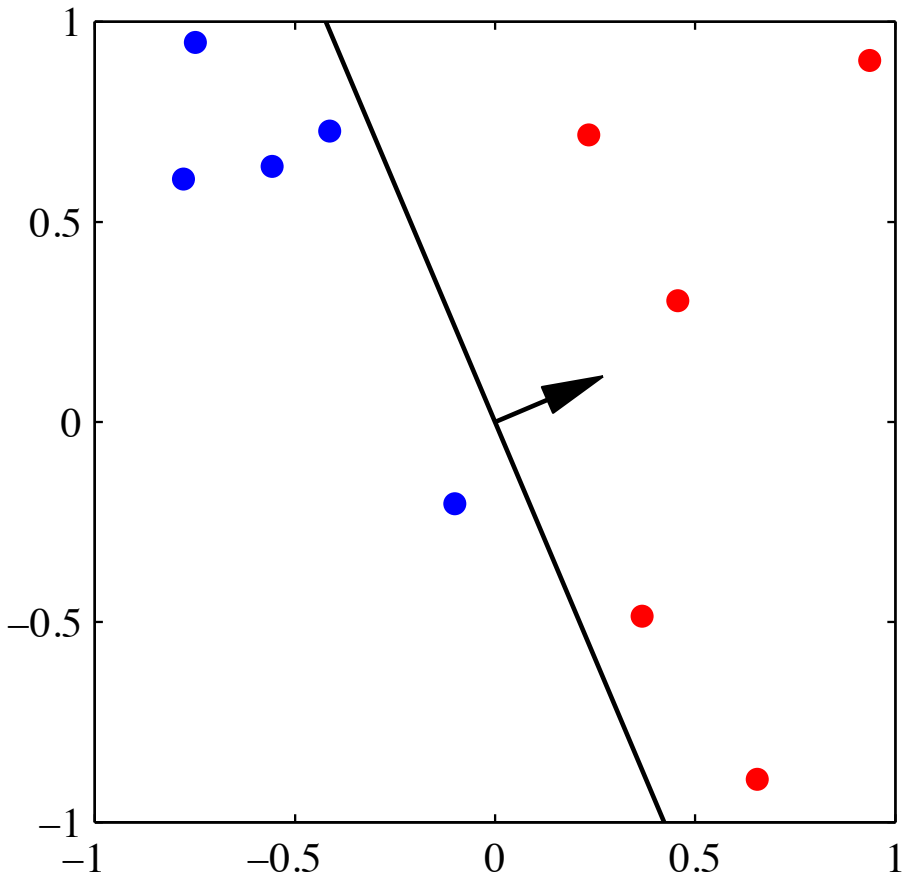


Red is the positive class

Blue is the negative class

Perceptron Algorithm

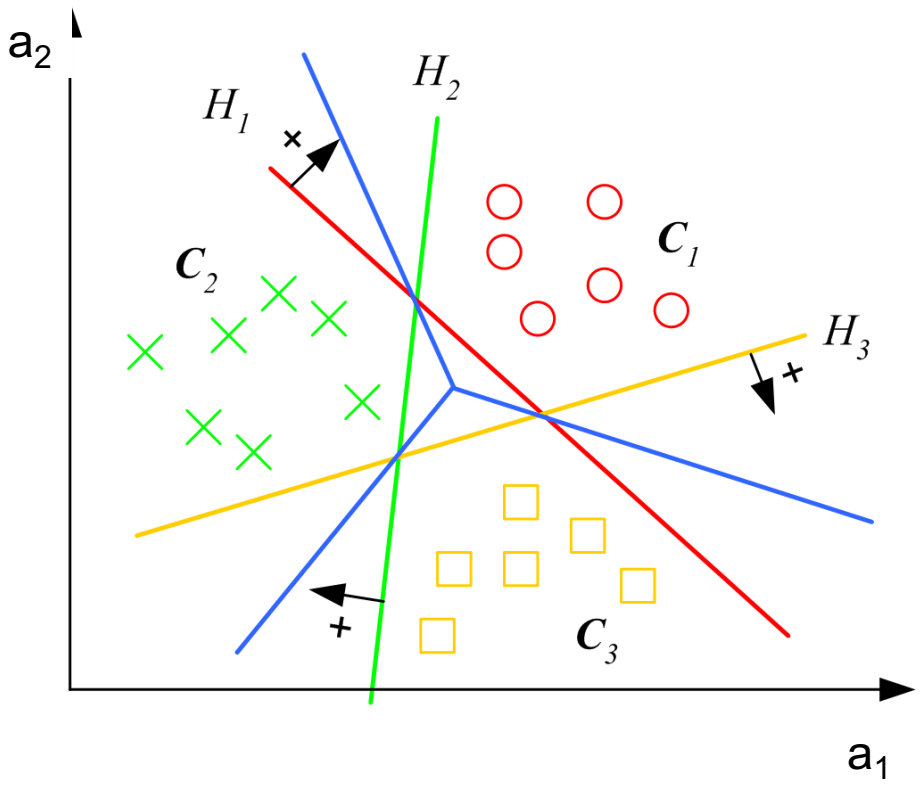
- Example (cont'd):



Red is the positive class

Blue is the negative class

Multi-class Classification



When there are N classes you can classify using N discriminant functions.

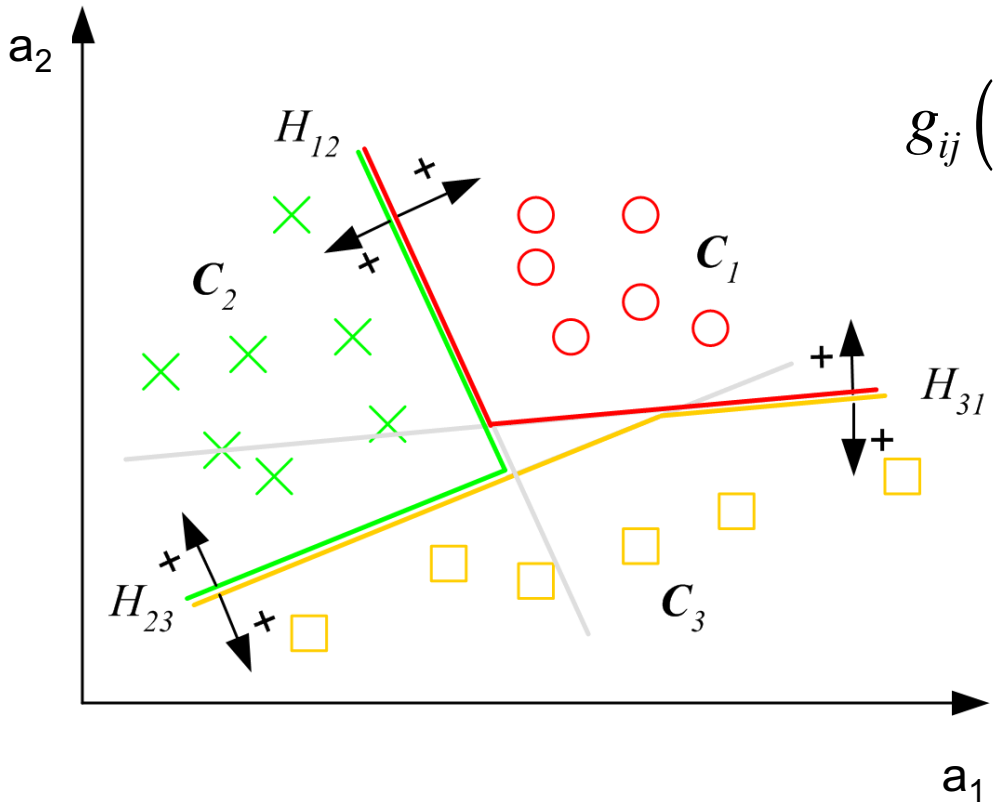
Choose the class c from the set of all classes C whose function $g_c(\mathbf{x})$ has the maximum output

Geometrically divides feature space into N **convex** decision regions

$$h(\mathbf{x}) = \underset{c \in C}{\operatorname{argmax}}\, g_c(\mathbf{x})$$

Pairwise Multi-class Classification

If they are not linearly separable (singly connected convex regions), may still be pair-wise separable, using N(N-1)/2 linear discriminants.



$$g_{ij}\left(\vec{x} \mid \vec{w}_{ij}, w_{ij0}\right) = w_{ij0} + \sum_{l=1}^{K} w_{ijl} x_l$$

choose C_i if

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

A more general idea

- The approach of the perceptron update rule is an example of a more general concept called Gradient Descent.

- In some sense, a lot of methods (neural nets, Hidden Markov Models, Gaussian Mixture Models) use some variant of Gradient Descent.

Gradient Descent

- Simple 1st order numerical optimization method
- Idea: follow the gradient of the objective function to a minimum
- Finds a global minimum when objective function is convex, otherwise it finds a local minimum
- Objective function (the function you are minimizing) must be differentiable
- Used when there is no analytical solution to finding minimum

What is the Gradient?

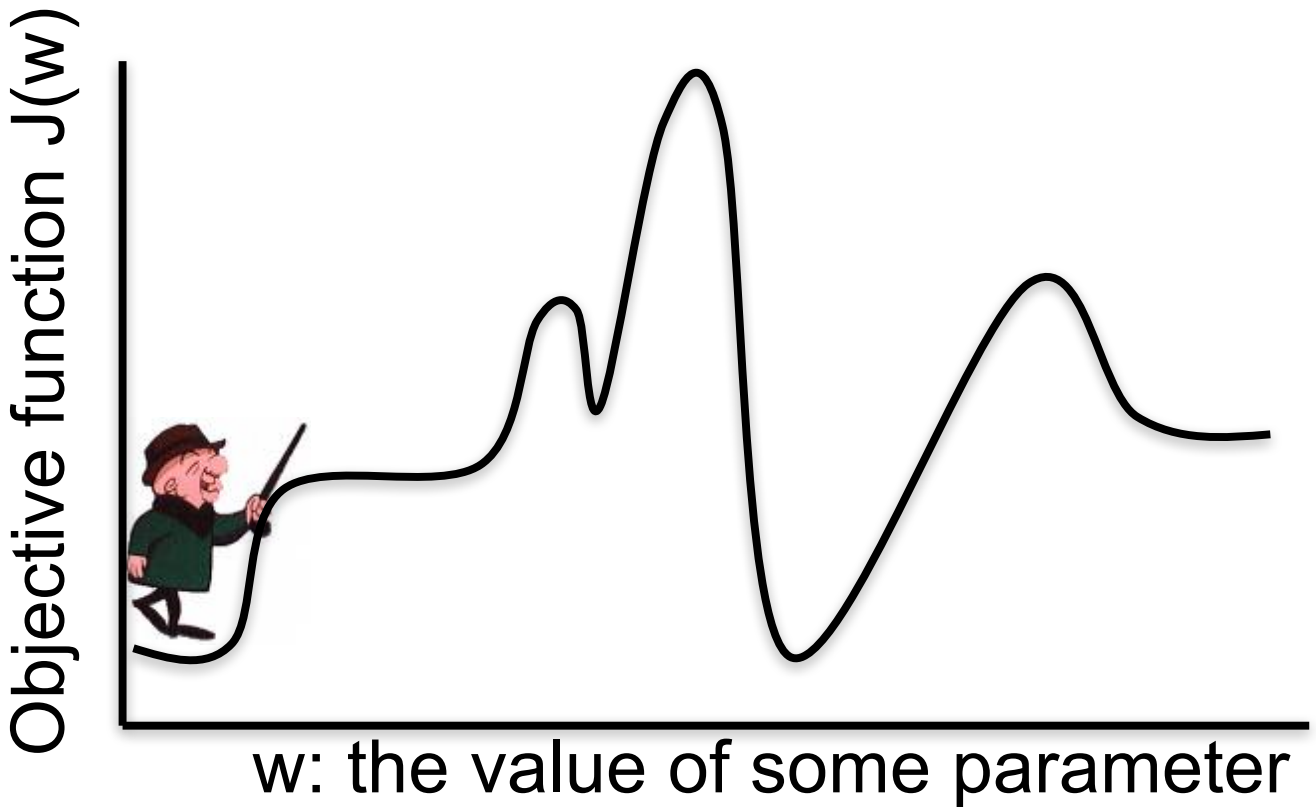
- The gradient is a fancy word for derivative, or the rate of change of a function with a scalar output and many inputs.
- You might also call it the slope (e.g. slope of a hill).

- It's a vector (a direction to move) that points in the direction of greatest increase (or, equivalently, decrease) of a function

- The gradient is zero at local maxima (top of a hill) or local minima (bottom of a valley).

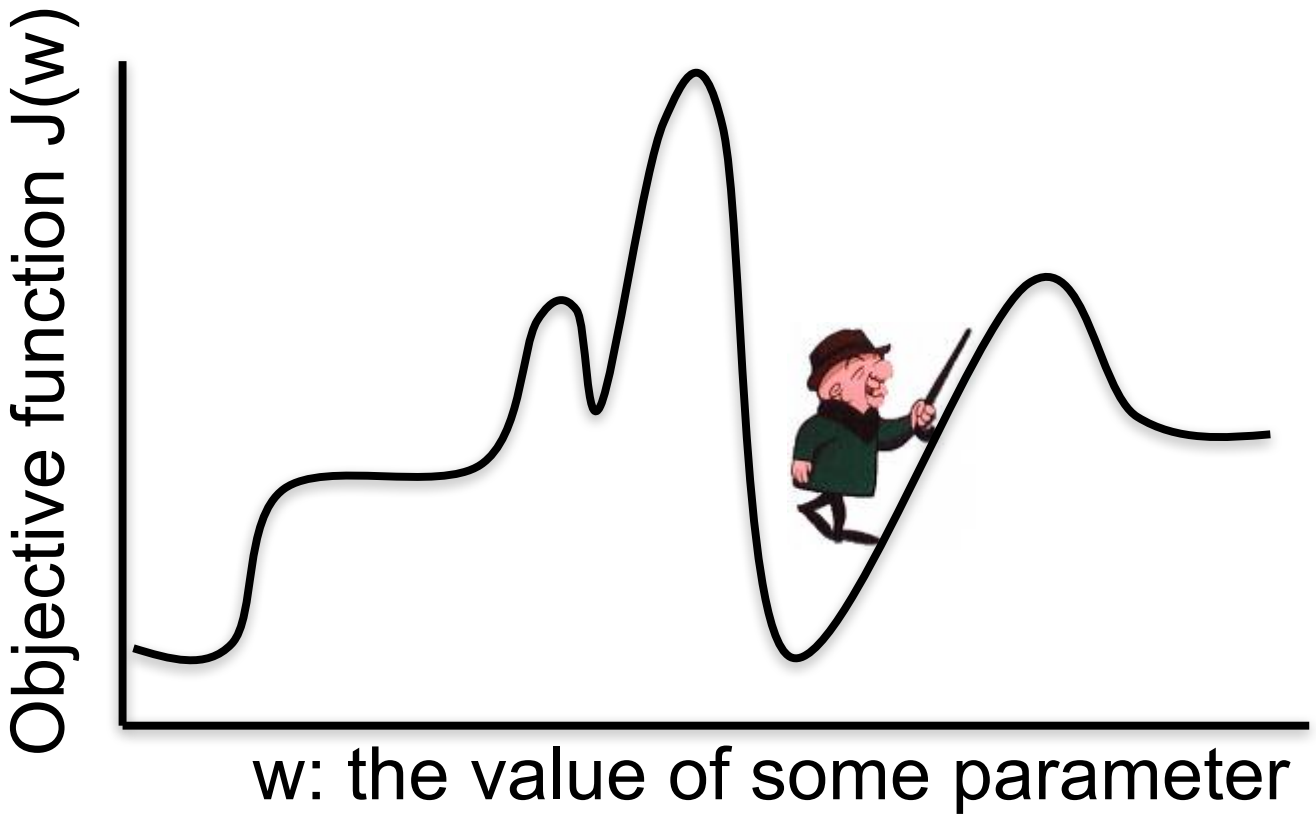
Hill-climbing (aka Gradient Descent)

Start somewhere and head up (or down) hill.



Hill-climbing (aka Gradient Descent)

Easy to get stuck in local maxima (minima)



Gradient Descent

- $\mathbf{w}$ are the model parameters
- D is the set of training data examples
- θ is the convergence threshold
- $\nabla J(\mathbf{w}, \mathrm{D})$ is the gradient of the objective function, with respect to the weights.
- η is the step size. It is often a function of what step we're on, or of the gradient, or on the size of the error

$\mathbf{w} = \ some\ random\ setting$

$\theta = something\ small$

$\eta(k) = a\ starting\ step\ size$

$k = 0$

Do

$k = k + 1$

$D_k = \ a\ random\ subset\ of\ D$

$\mathbf{w} = \mathbf{w} - \eta(k)\ \nabla J(\mathbf{w}, D_k)$

Until $|\eta(k)\ \nabla J(\mathbf{w}, D_k)| < \theta$

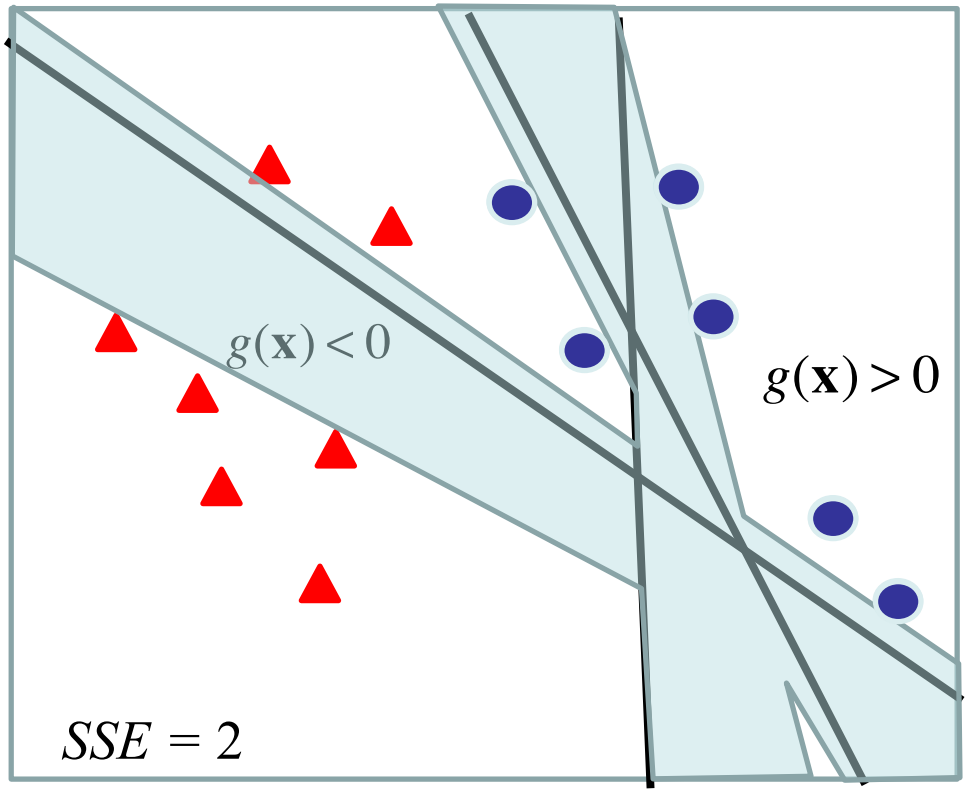
Gradient Descent

- In **batch gradient descent**, the objective function J is a function of both the parameters and ALL training samples, summing the total error, e.g. $D_k \equiv D$

- In **stochastic gradient descent**, J is a function of the parameters and a different single random training sample at each iteration. This is a common choice in when there is a lot of training data, and computing the sum over all samples is expensive.

What if ∇J is 0?

Here, our objective function is the sum of squared classification errors.



$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$SSE = \sum_{i}^{n} (y_i - h(\mathbf{x}_i))^2$$

$$J(\mathbf{w}, D) = SSE$$

The gradient of *J* is 0 in the blue region!
This is a problem because the system can't tell from the gradient which way the line should move.

Perceptron Alg = Gradient descent?

- Gradient descent

Do

$k = k + 1$

$D_k = \ a\ random\ subset\ of\ D$

$\mathbf{w} = \mathbf{w} - \eta(k)\ \nabla J(\mathbf{w}, D_k)$

Until $|\eta(k)\ \nabla J(\mathbf{w}, D_k)| < \theta$

- Perceptron Algorithm

Do

$k = (k + 1) \mathrm{mod}(m)$

if $h(\mathbf{x}_k)! = y_k$

$\mathbf{w} = \mathbf{w} + \mathbf{x}y$

Until $\forall k,\ \mathrm{g}(\mathbf{x}_k) = y_k$

If we assume a random initial ordering of the data D, then those 1st couple lines of the perceptron algorithm just cycle through a random permutation of the data, 1 point at a time (stochastic gradient descent).

So how does the perceptron rule for updating **w** relate to gradient descent? How

Appendix

(stuff I didn't have time to discuss in class…and for which I haven't updated the notation.)

Linear Discriminants

- A linear combination of the attributes.

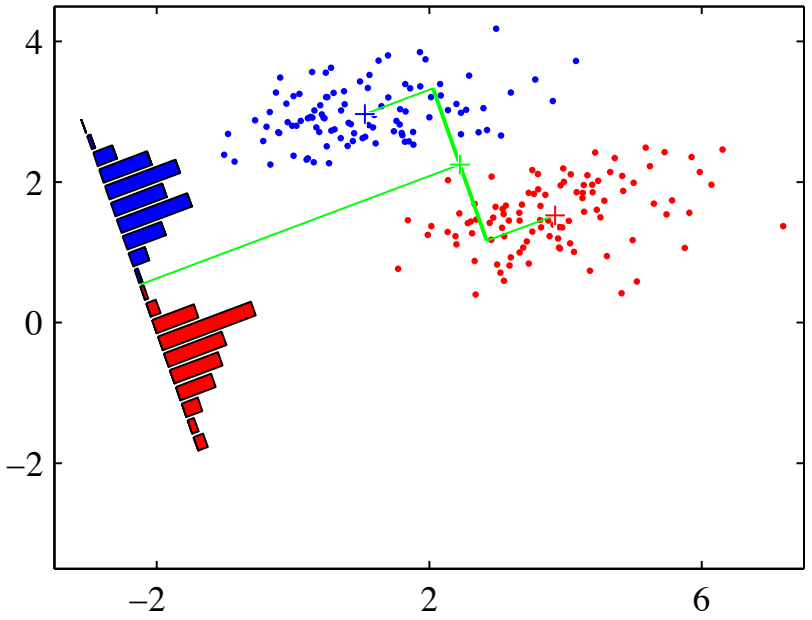
$$g\left(\vec{x} \mid \vec{w}, w_0\right) = w_0 + \vec{w}^T \vec{x} = w_0 + \sum_{i=1}^{k} w_i a_i$$

- Easily interpretable

- Are optimal when classes are Gaussian and share a covariance matrix

Fisher Linear Discriminant Criteria

- Can think of $\vec{w}^T \vec{x}$ as dimensionality reduction from K-dimensions to 1
- Objective:
  - Maximize the difference between class means
  - Minimize the variance within the classes



$$J(\vec{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

where s_i and m_i are the sample variance and mean for class i in the projected dimension. We want to maximize J.

Fisher Linear Discriminant Criteria

- Solution:

$$\vec{w} = \mathbf{S}_W^{-1}(\vec{m}_2 - \vec{m}_1)$$

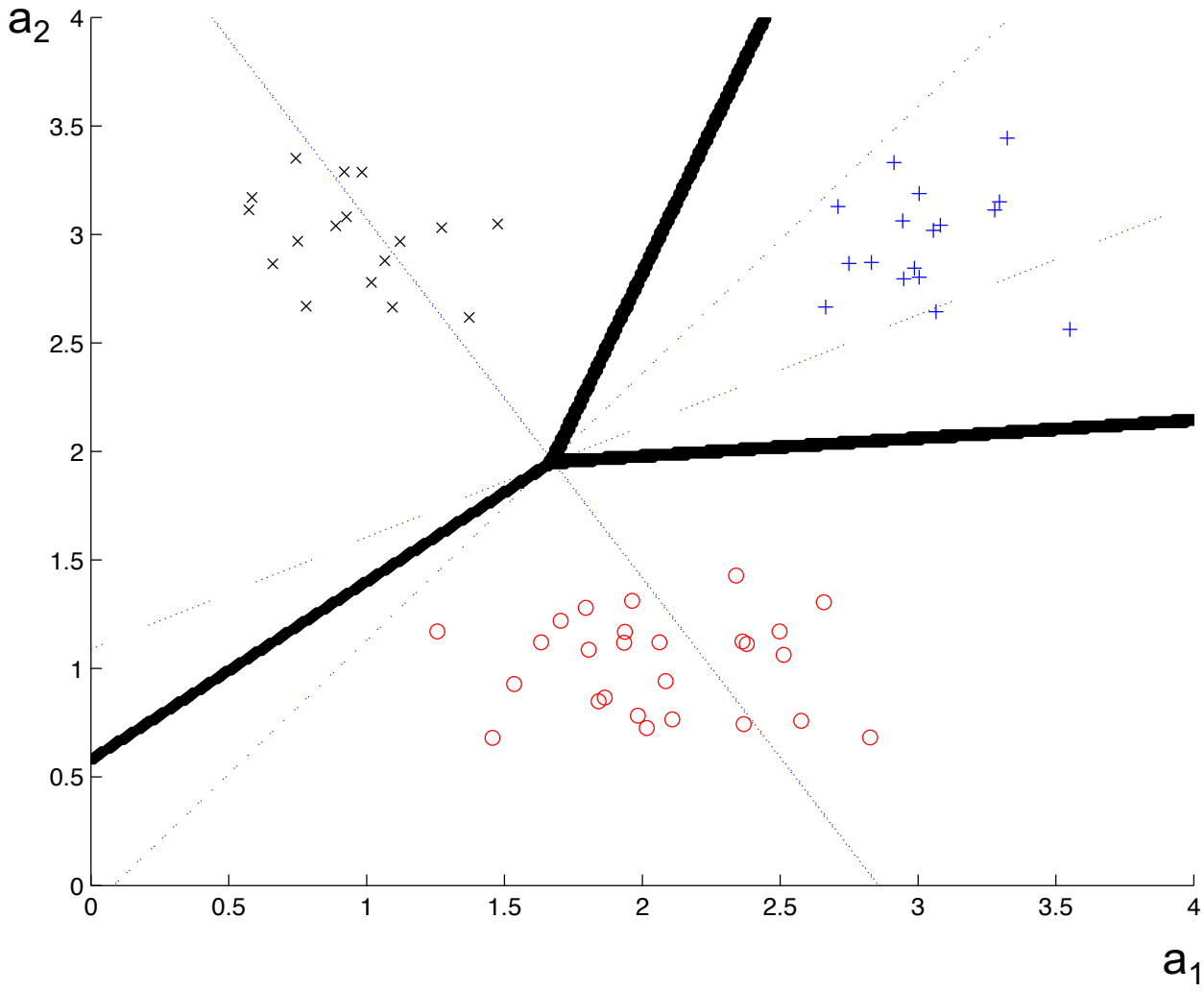
where

$$\mathbf{S}_W = \sum_{n \in C_1} (\vec{x}_n - \vec{m}_1)(\vec{x}_n - \vec{m}_1)^T + \sum_{n \in C_2} (\vec{x}_n - \vec{m}_2)(\vec{x}_n - \vec{m}_2)^T$$

- However, while this finds finds the direction ($\vec{w}$) of decision boundary. Must still solve for w_0 to find the threshold.

- Can be expanded to multiple classes

Logistic Regression (Discrimination)



Logistic Regression (Discrimination)

- Discriminant model but well-grounded in probability
- Flexible assumptions (exponential family class-conditional densities)
- Differentiable error function ("cross entropy")
- Works very well when classes are linearly separable

Logistic Regression (Discrimination)

- Probabilistic discriminative model
- Models posterior probability $p(C_1|\overrightarrow{x})$
- To see this, let's start with the 2-class formulation:

$$
\begin{aligned}
p(C_1|x) &= \frac{p(\overrightarrow{x}|C_1)p(C_1)}{p(\overrightarrow{x}|C_1)p(C_1) + p(\overrightarrow{x}|C_2)p(C_2)} \\
&= \frac{1}{1 + \exp\left(-\log \frac{p(\overrightarrow{x}|C_1)p(C_1)}{p(\overrightarrow{x}|C_2)p(C_2)}\right)} \\
&= \frac{1}{1 + \exp(-\alpha)} \qquad \text{logistic sigmoid function} \\
&= \sigma(\alpha)
\end{aligned}
$$

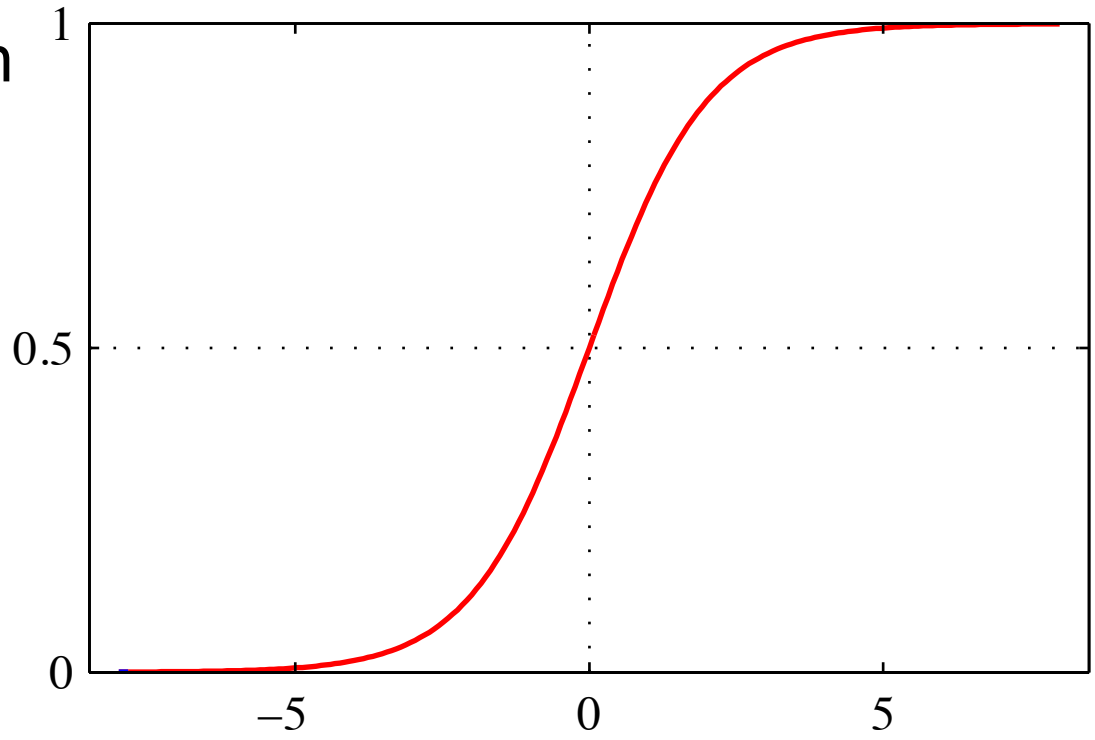
where

$$
\alpha = \log \frac{p(\overrightarrow{x}|C_1)p(C_1)}{p(\overrightarrow{x}|C_2)p(C_2)}
$$

Logistic Regression (Discrimination)

logistic sigmoid function

$$\sigma(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$



"Squashing function" that maps $(-\infty, +\infty) \rightarrow (0, 1)$

Logistic Regression (Discrimination)

For exponential family of densities,

$$\alpha = \log \frac{p(\overrightarrow{x}|C_1)p(C_1)}{p(\overrightarrow{x}|C_2)p(C_2)}$$

is a linear function of x.

Therefore we can model the posterior probability as a logistic sigmoid acting on a linear function of the attribute vector, and simply solve for the weight vector **w** (e.g. treat it as a discriminant model):

$$y = p(C_1|\overrightarrow{x}) = \sigma(w_0 + \sum_{i=1}^{k} w_i a_i) \qquad p(C_2|\overrightarrow{x}) = 1 - p(C_1|\overrightarrow{x})$$

To classify: $h(\overrightarrow{x}_i) = \begin{cases} C_1 & y_i > 0.5 \\ C_2 & o.w. \end{cases}$