349:Machine Learning

Fall 2024

Linear Discriminants
and Perceptron Algorithm

Discrimination Learning Task

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

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*

$$f : X \to Y$$

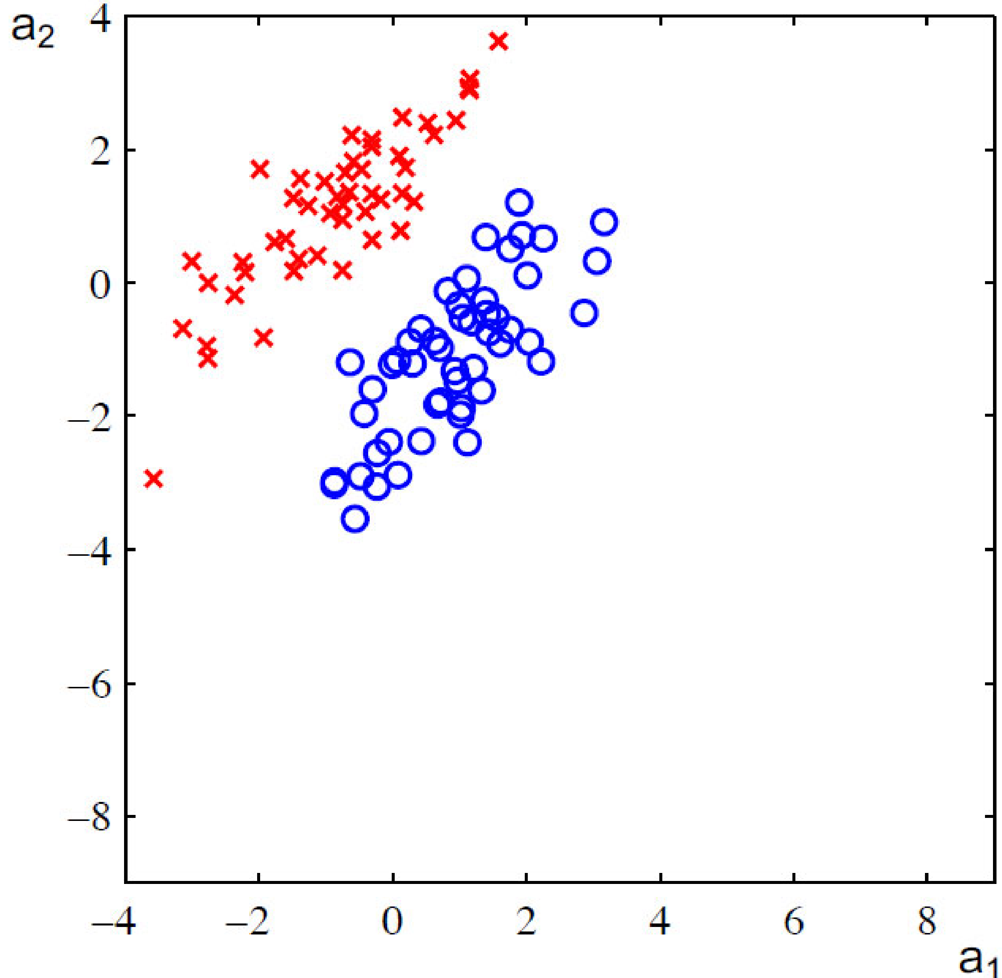
The DATA is a set of tuples <example, response value>

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

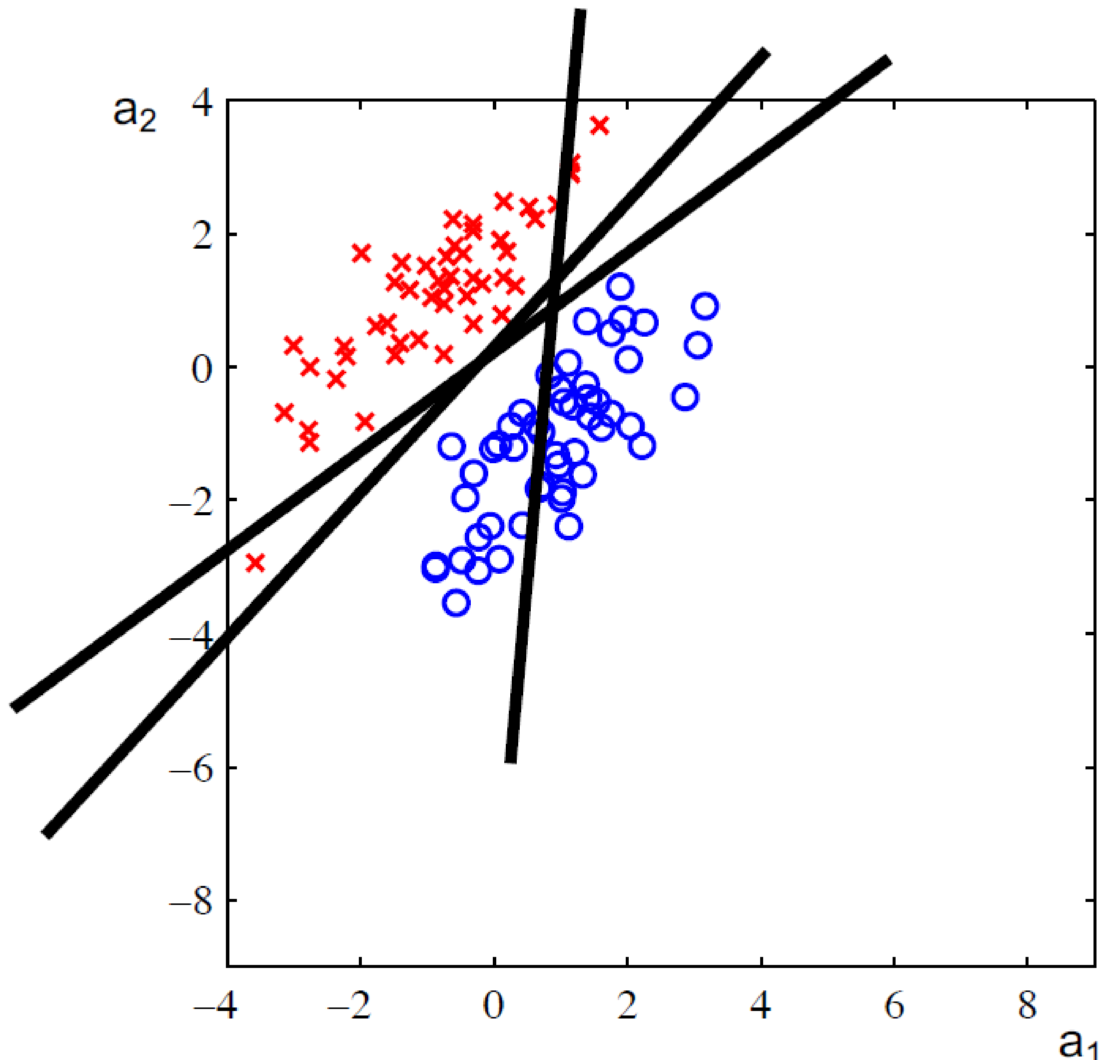
Find a hypothesis ***h*** such that...

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

Visually: Where to draw the line?



Visually: Where to draw the line?



Reminder about notation

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

$$g(x) = w_0 + w_1x_1 + w_2x_2 .... + w_kx_k$$

- 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 **w** to include w_0 and **x** to include an x_0 that is always 1, now…

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

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

- This lets us notate things as…

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

Linear Discriminants

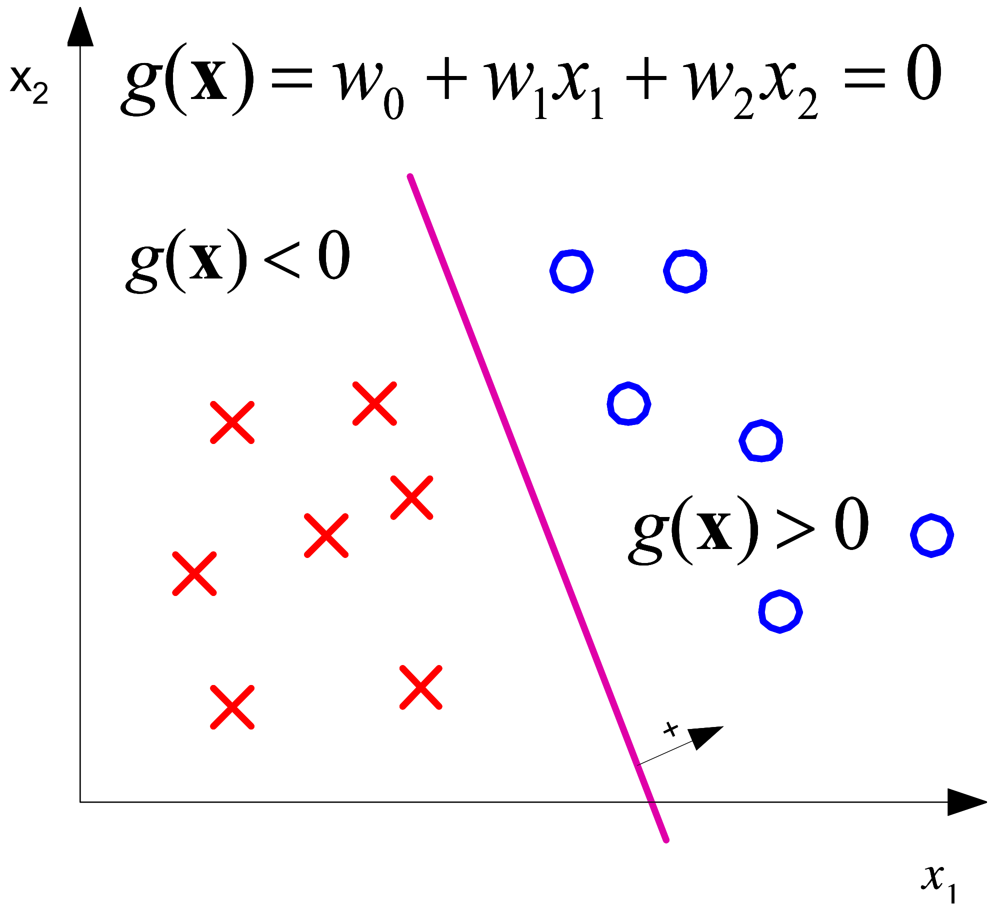
- A linear combination of the attributes.

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

- Easily interpretable

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*

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

Two-Class Classification - Summary

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

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

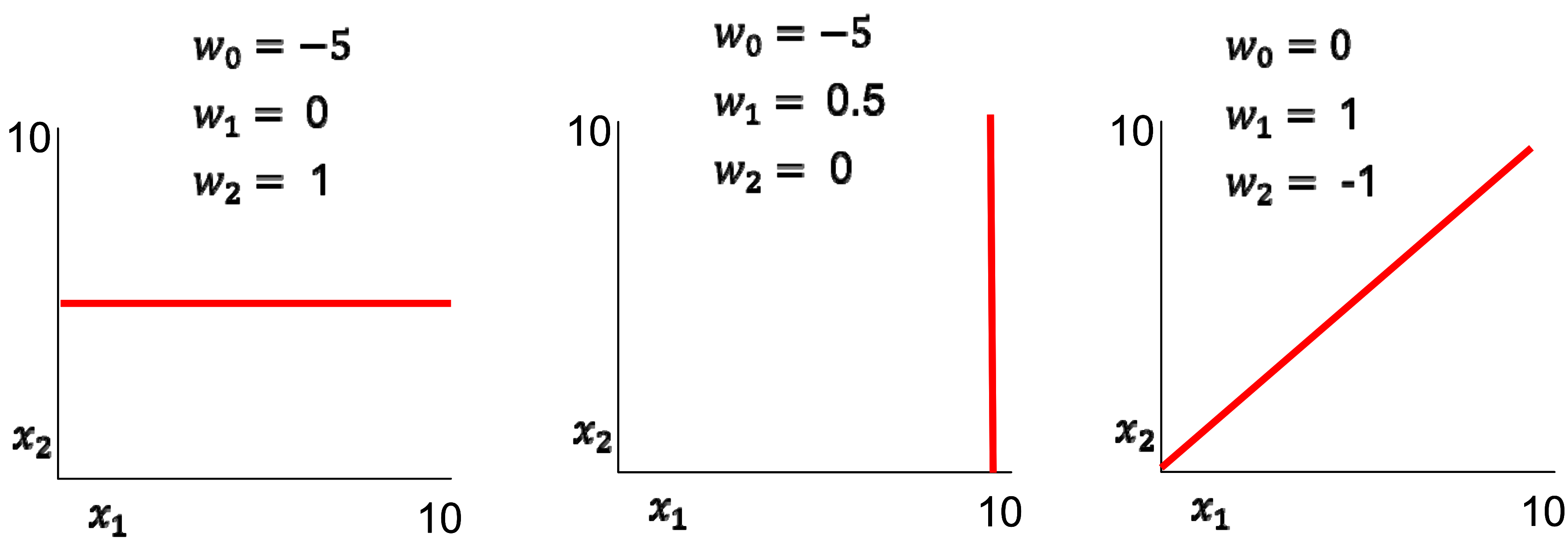
- Things on each side of 0 get their class labels according to the sign of what g(x) outputs.

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

- We will use the Perceptron algorithm.

Example 2-D decision boundaries

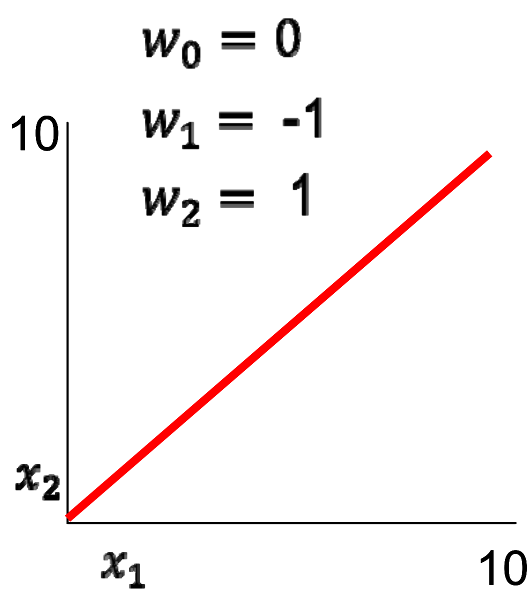
$$0 = g(x) = w_0 + w_1 x_1 + w_2 x_2 = \mathbf{w}^T \mathbf{x}$$

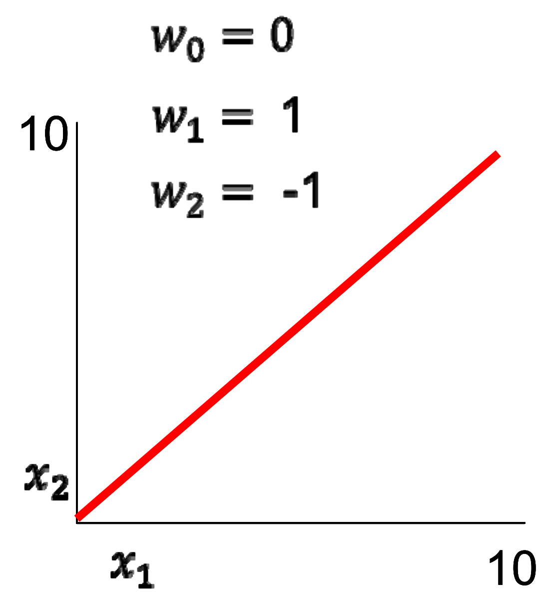


What’s the difference?

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

What’s the difference between these two?





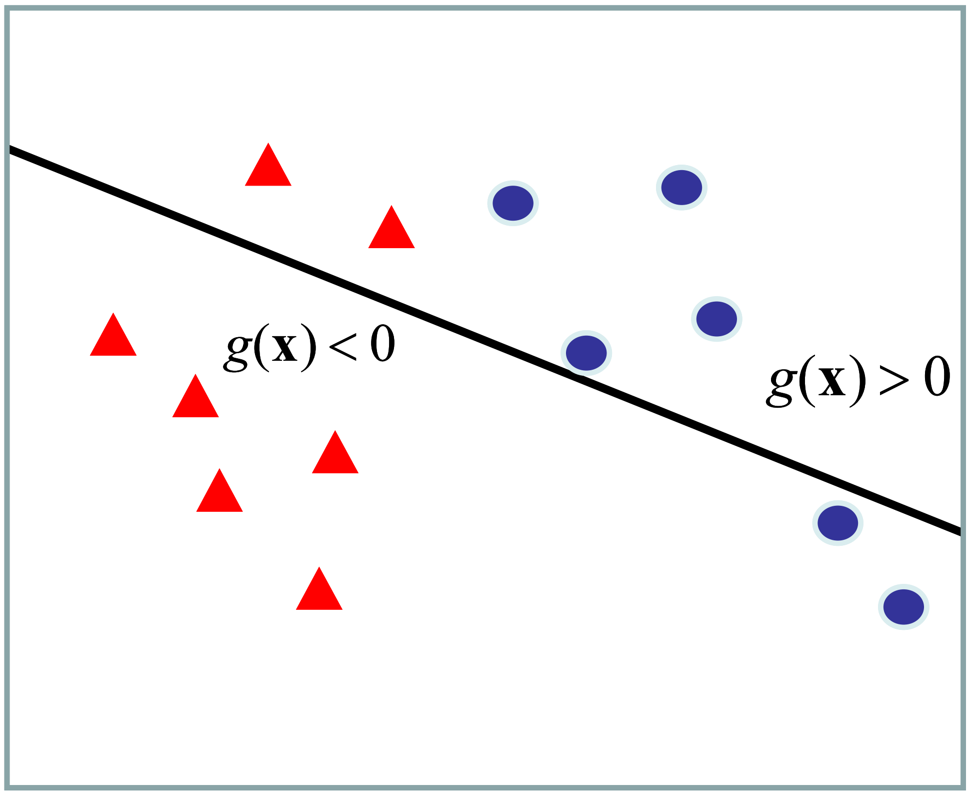
How to we learn w ?

- 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?

Zero-One Loss

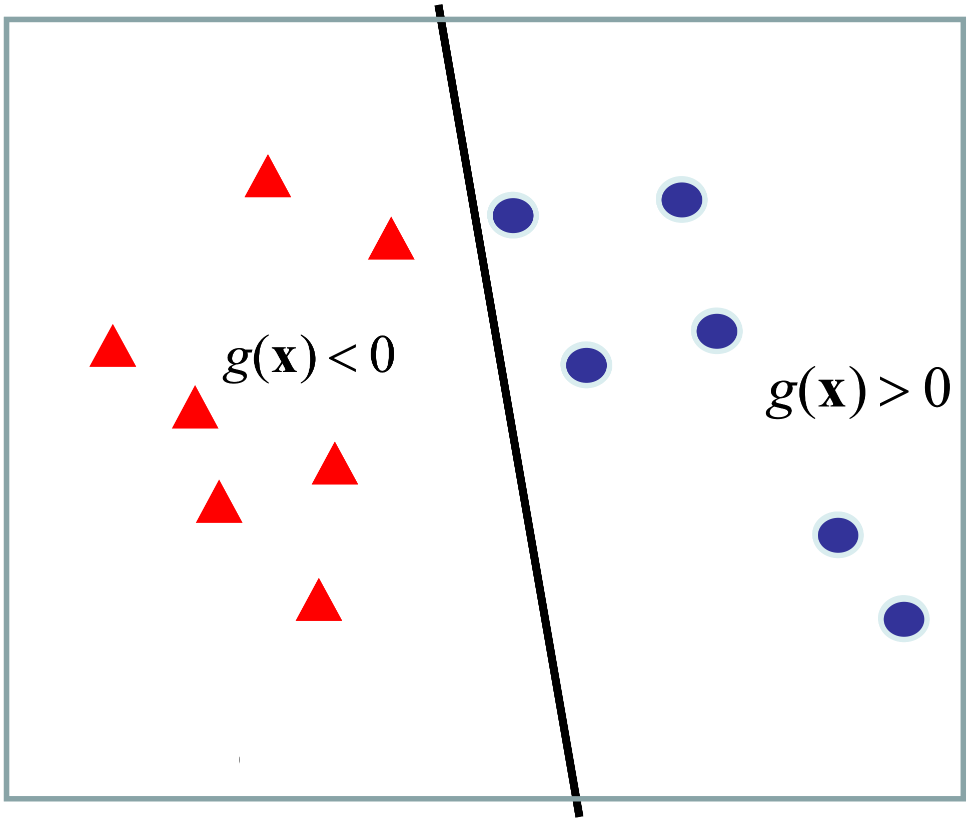


$$g(x) = w_0 + w_1 x_1 + w_2 x_2$$
$$= \mathrm{x}\mathrm{w}$$

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

$$L(X, Y, w) = \sum_{i=1}^{N} 1(y_i = h(x_i))$$
$$= 4$$

Zero-One Loss



$$g(x) = w_0 + w_1 x_1 + w_2 x_2$$
$$= \mathrm{x}\mathrm{w}$$

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

$$L(X, Y, w) = \sum_{i=1}^{N} 1(y_i = h(x_i))$$
$$= 0$$

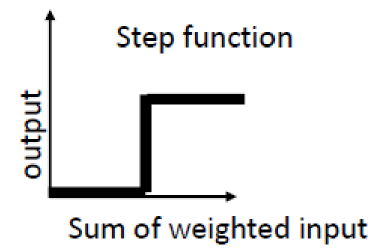
No closed form solution!

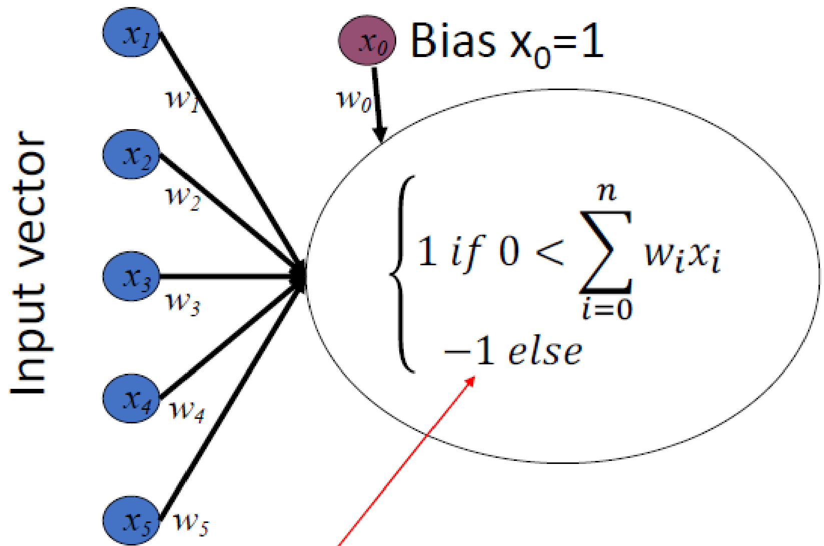
- For many **objective functions** we **can't find** a formula to to get the best model parameters, like we could with regression.
- The objective function from the previous slide is one of those **"no closed form solution"** functions.
- This means we have to **try various guesses** for what the weights should be and try them out.
- Let's look at the **perceptron** approach.

The Perceptron

- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386-408
- The “first wave” in neural networks
- A linear classifier

A Single Perceptron



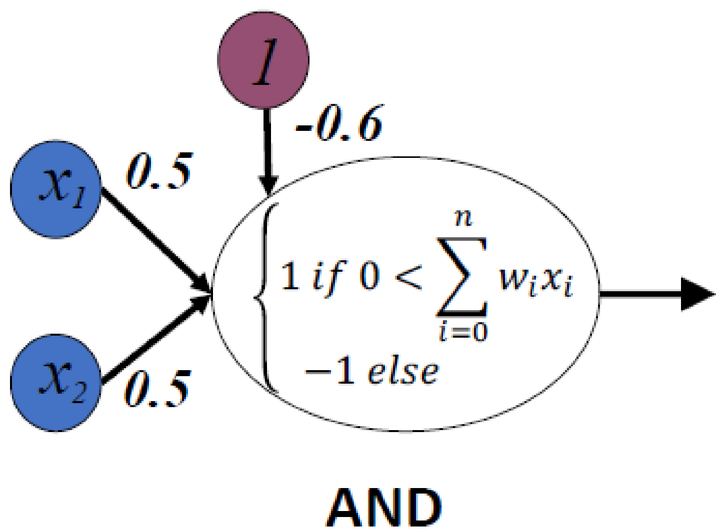


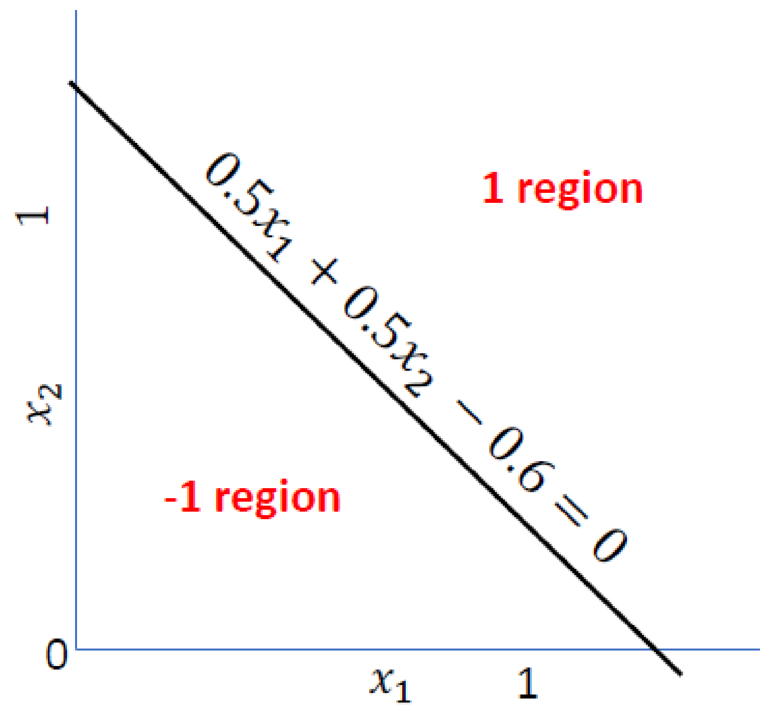
$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

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

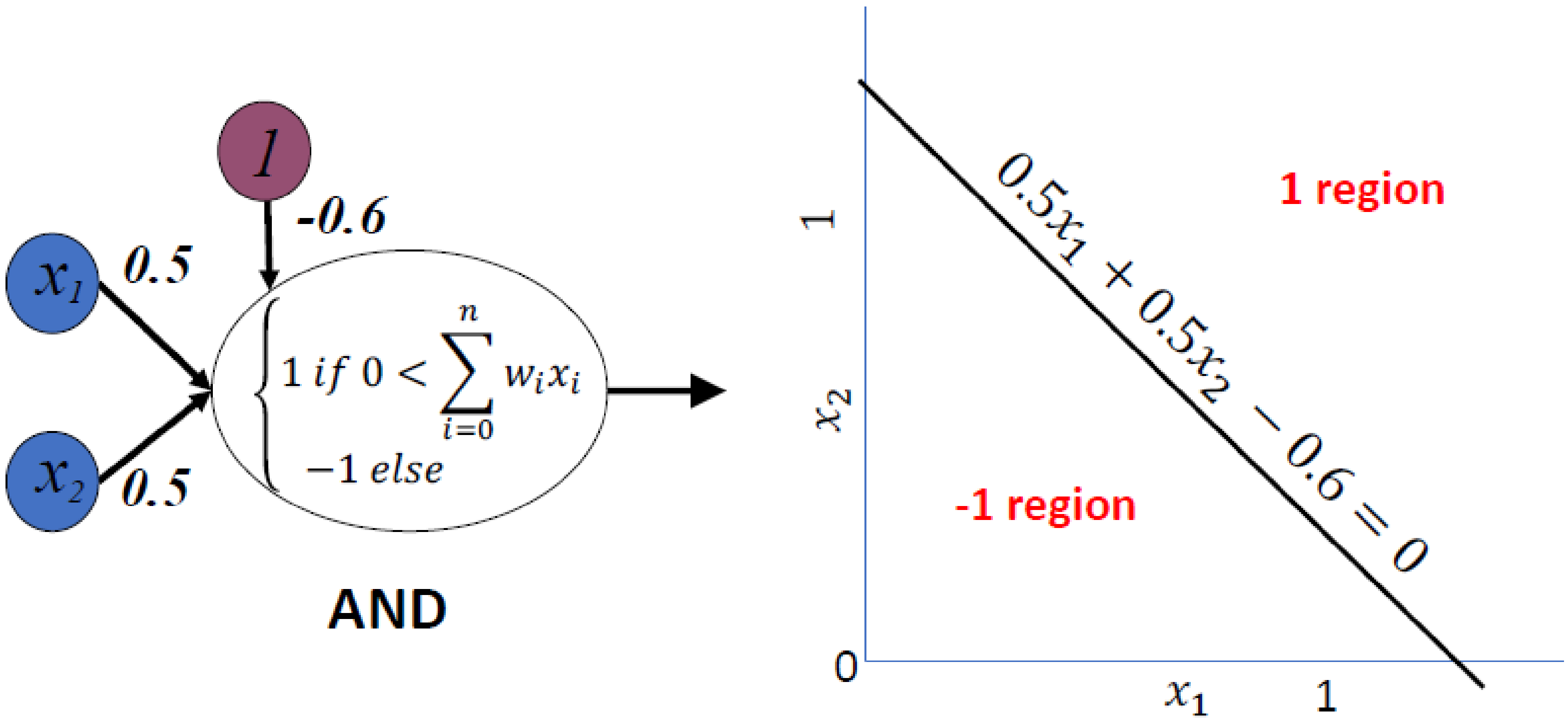
Note: Output can be 0 or 1, depending on whether we are following the perceptron algorithm (presented later in the slides) or are doing Boolean logic.

Weights Define a Hyperplane in the Input Space

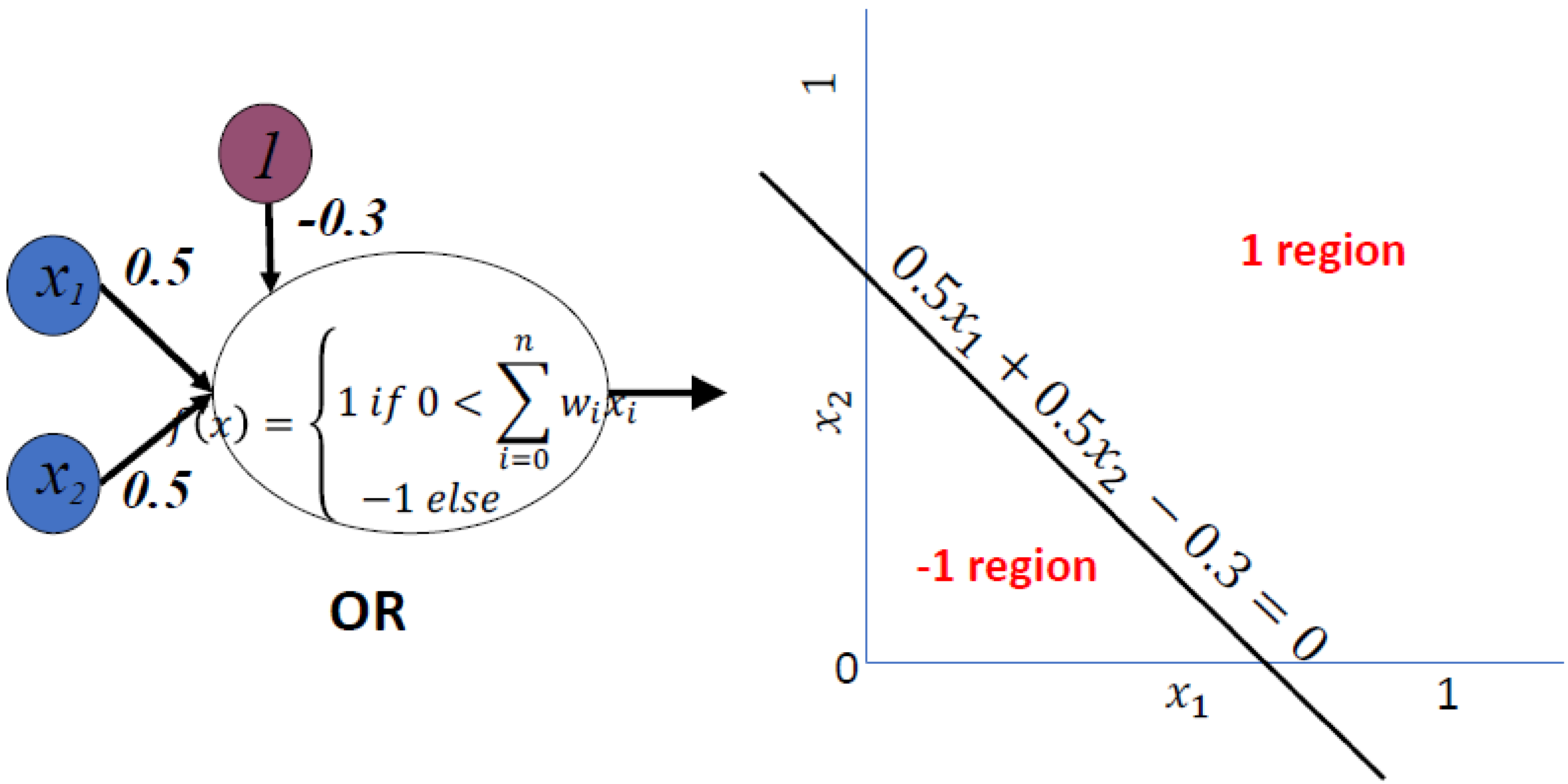




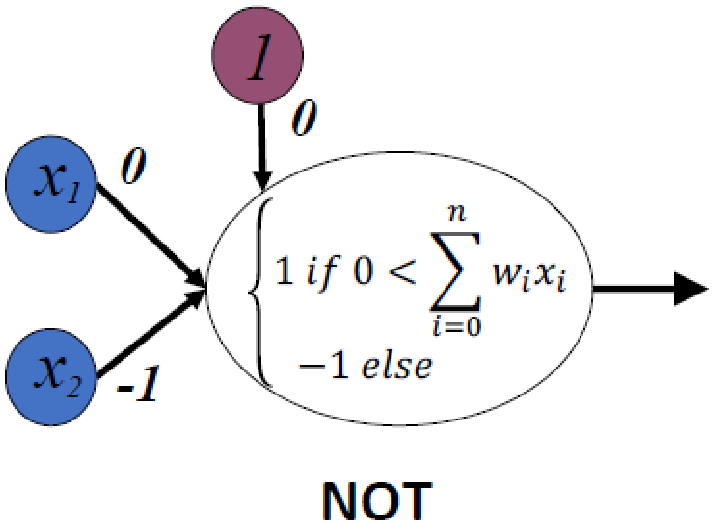
Classifies Any (Linearly Separable) Data

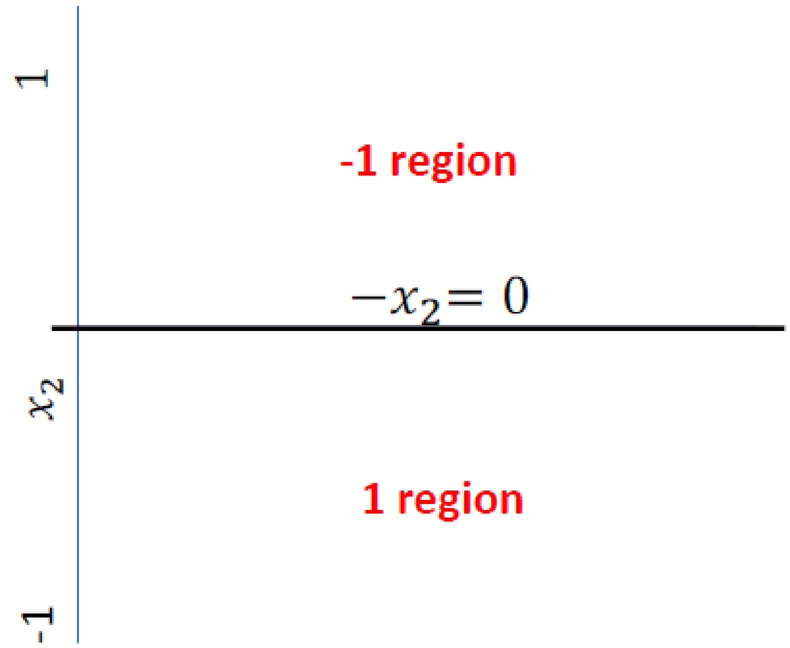


Different Logical Functions Are Possible

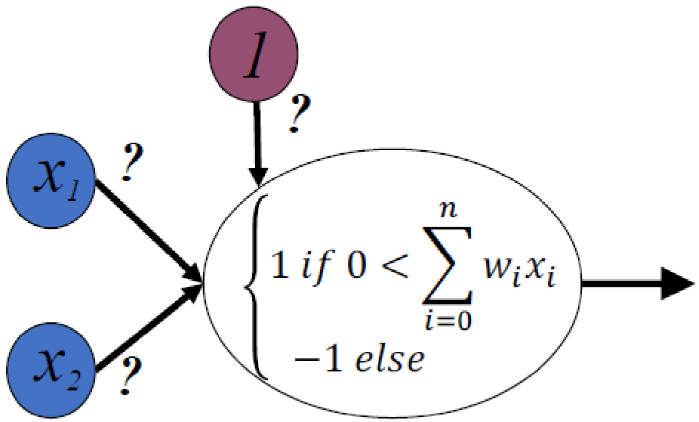


And, Or, Not Are Easy to Define

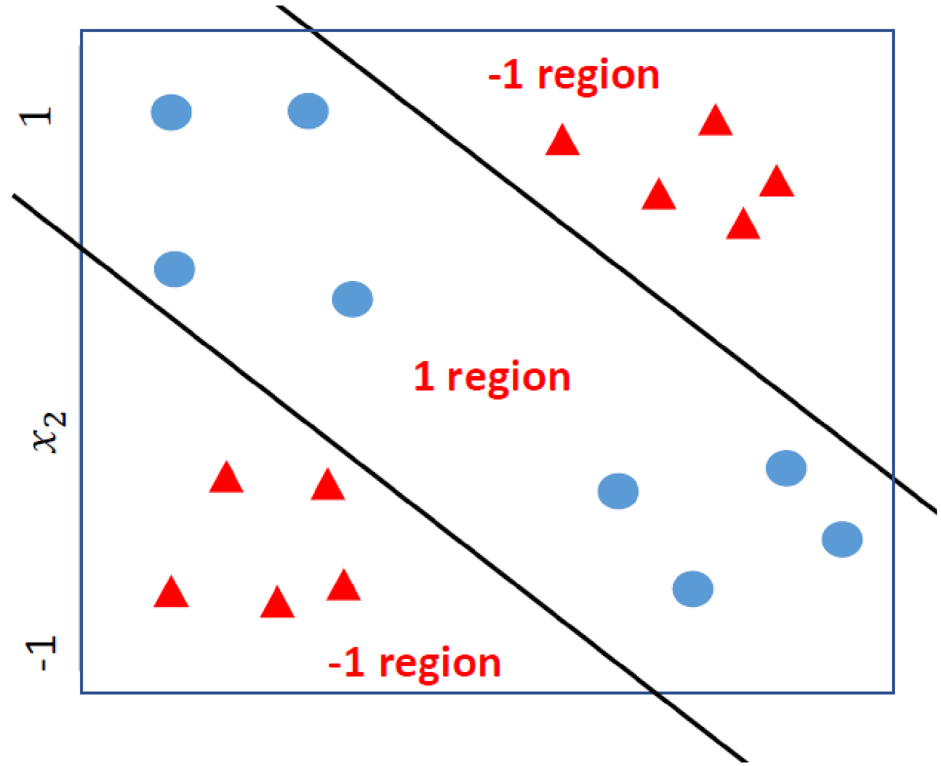




What About XOR?

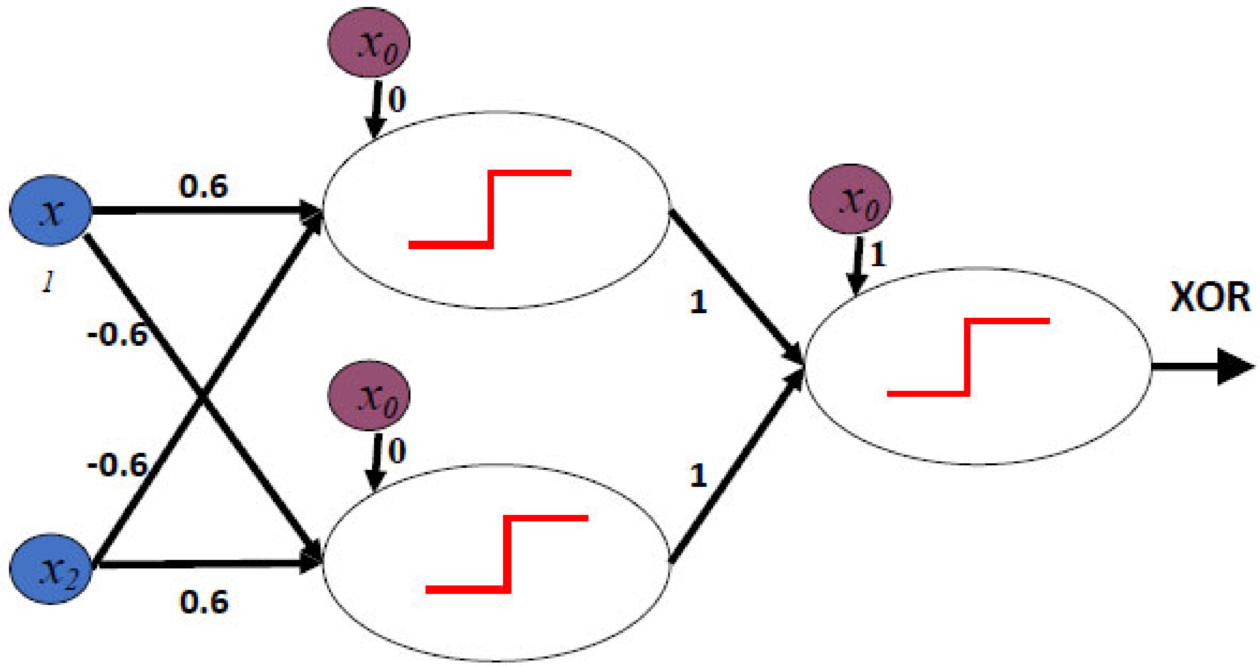


XOR



Perceptron is a *linear* classifier; it only learns linear boundaries.

Combining Perceptrons Can Make Any Boolean Function



...if you can set the weights & connections right

Reframing Our Objective Function

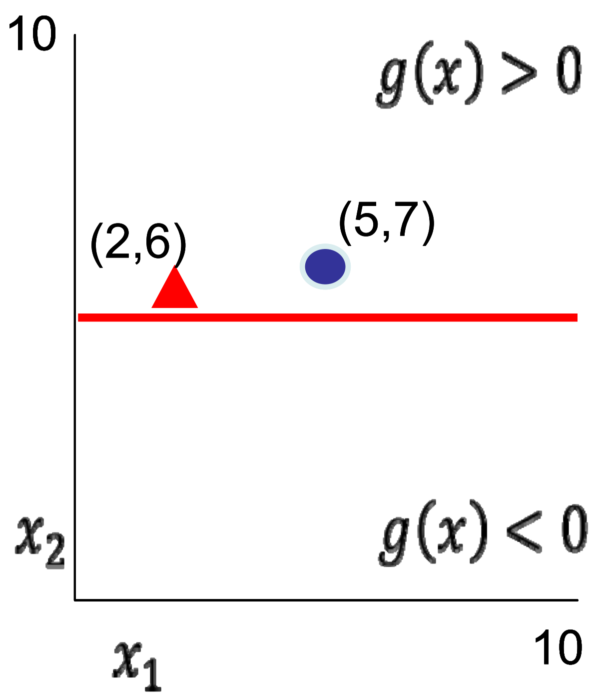
We want to get all positively-labeled points on the positive side of the boundary, all negatively-labeled points on the negative side.

$$J(D, w) = \sum_{i=1}^{N} \mathbf{1}(y_i = x_i \times w)$$

When $y \in \{-1, 1\}$ phrase this as: $(\boldsymbol{xw})y > 0$ for all $(\mathbf{x}, y) \in D$

Why does this equation capture our goal?

An Example



Goal: classify as +1 and as -1 by putting a line between them.

Our objective function is…

$$(\mathbf{w}^T\mathbf{x})y > 0$$

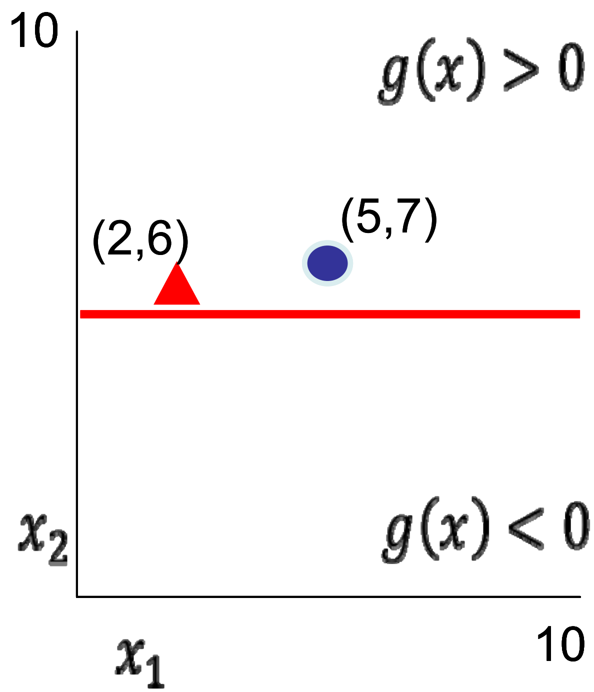
Start with a randomly placed line.

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

Measure the objective for each point.

Move the line if the objective isn't met.

An Example



Goal: classify as +1 and as -1 by putting a line between them.

Our objective function is…

$$(\mathbf{w}^T\mathbf{x})y > 0$$

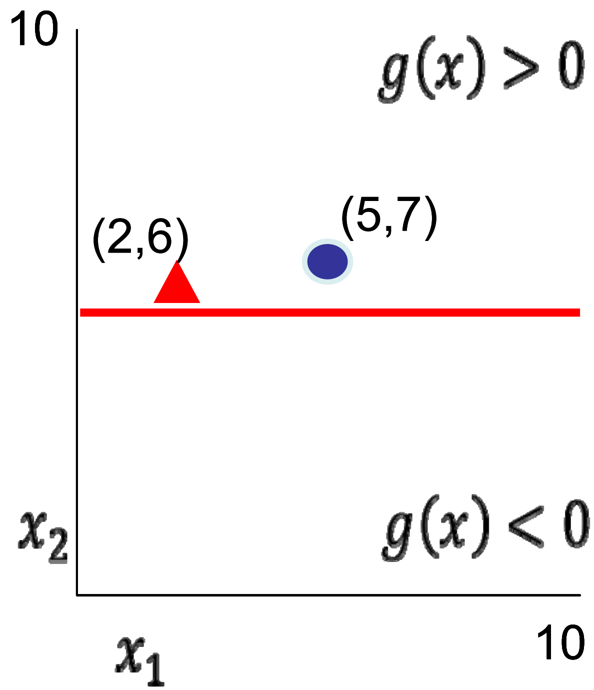
Start with a randomly placed line.

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

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

Objective met. Don't move the line.

An Example



Goal: classify ● as +1 and ▲ as -1 by putting a line between them.

Our objective function is…

$$(\mathbf{w}^T\mathbf{x})y > 0$$

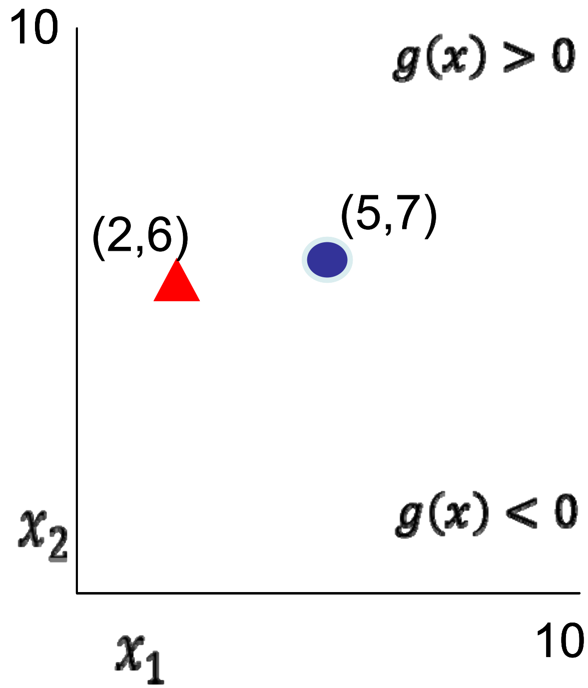
Start with a randomly placed line.

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

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

Objective not met. Move the line.

An Example



Goal: classify ● as +1 and ▲ as -1 by putting a line between them.

Our objective function is…

$$(\mathbf{w}^T\mathbf{x})y > 0$$

Start with a randomly placed line.

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

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 $\mathbf{w} = [-6, -2, -5]$? 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

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

$$m = |D| = \text{size of data set}$$

The classification function

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

The weight update algorithm

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

Do

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

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

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

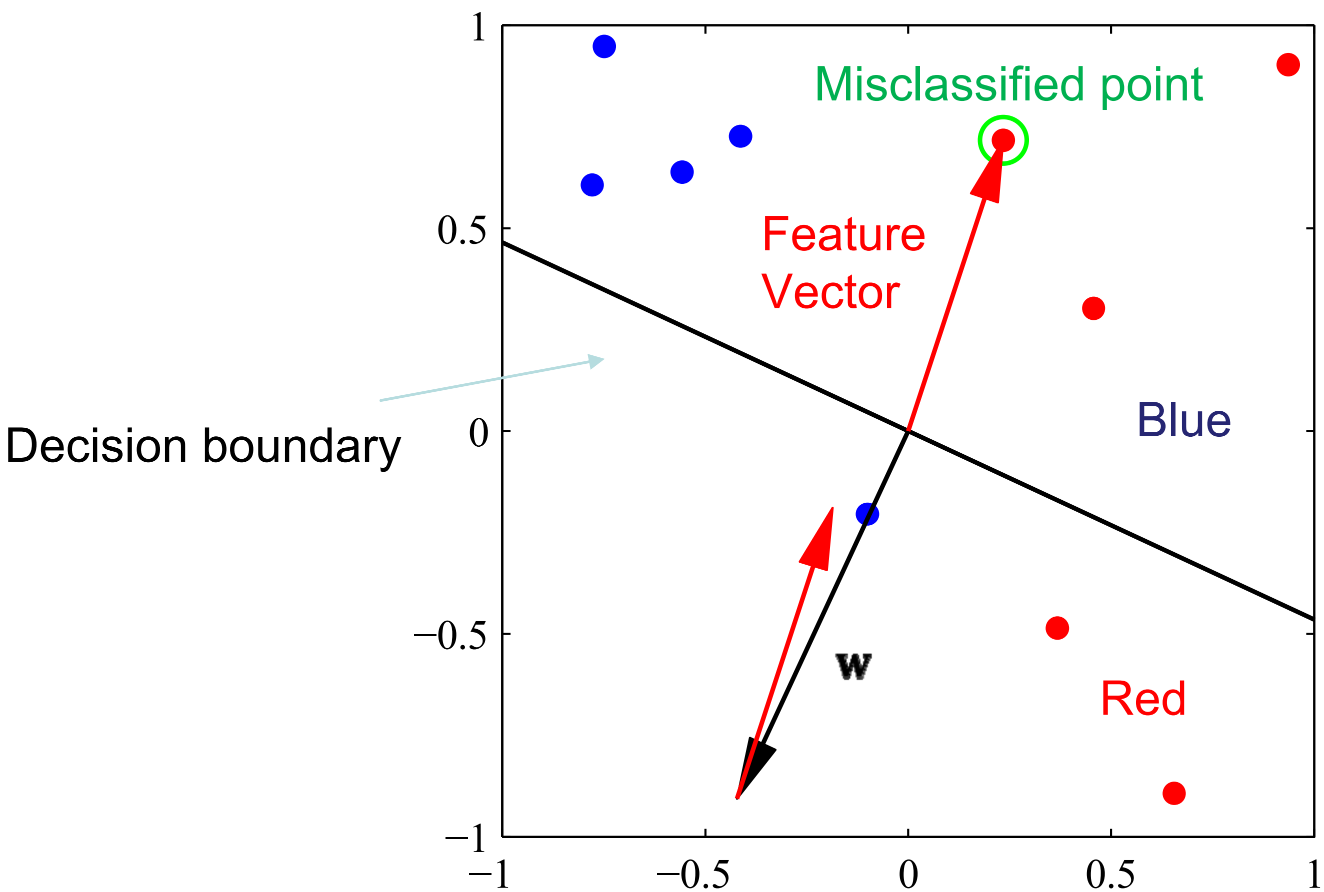
$$\boxed{\text{Until } \forall k,\ h(\mathbf{x}_k) = y_k}$$

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

This means you have to add another exit condition for when you've gone through the data too many times and suspect you'll never terminate.

Perceptron Algorithm

- Example:

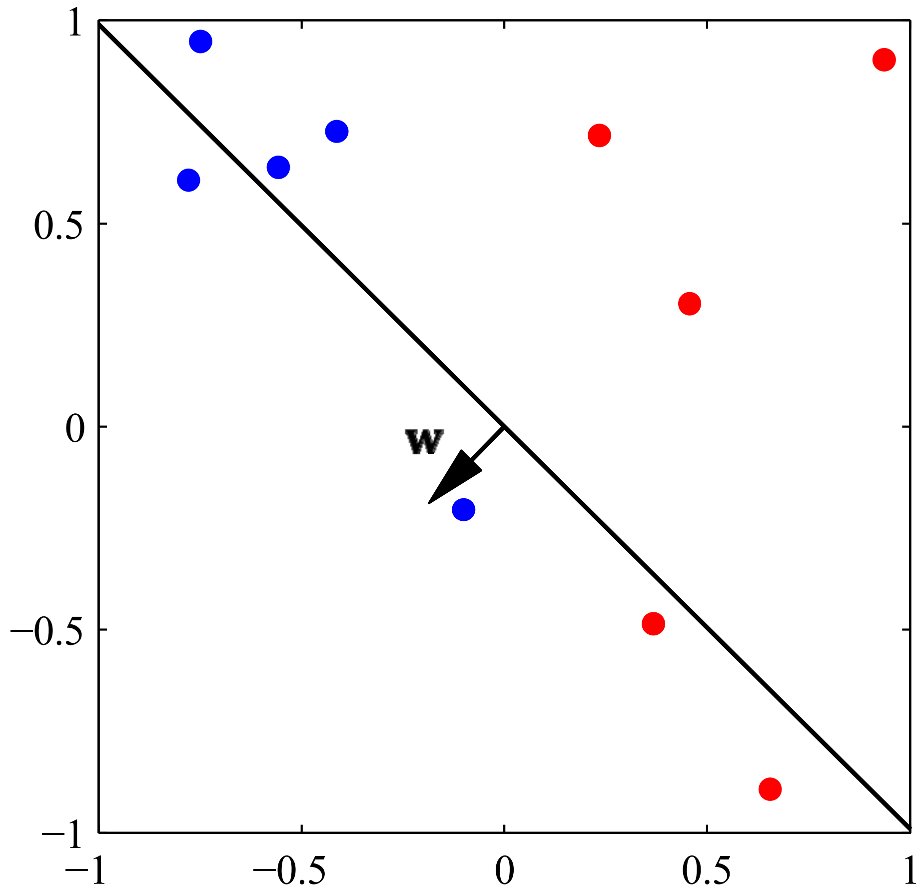


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

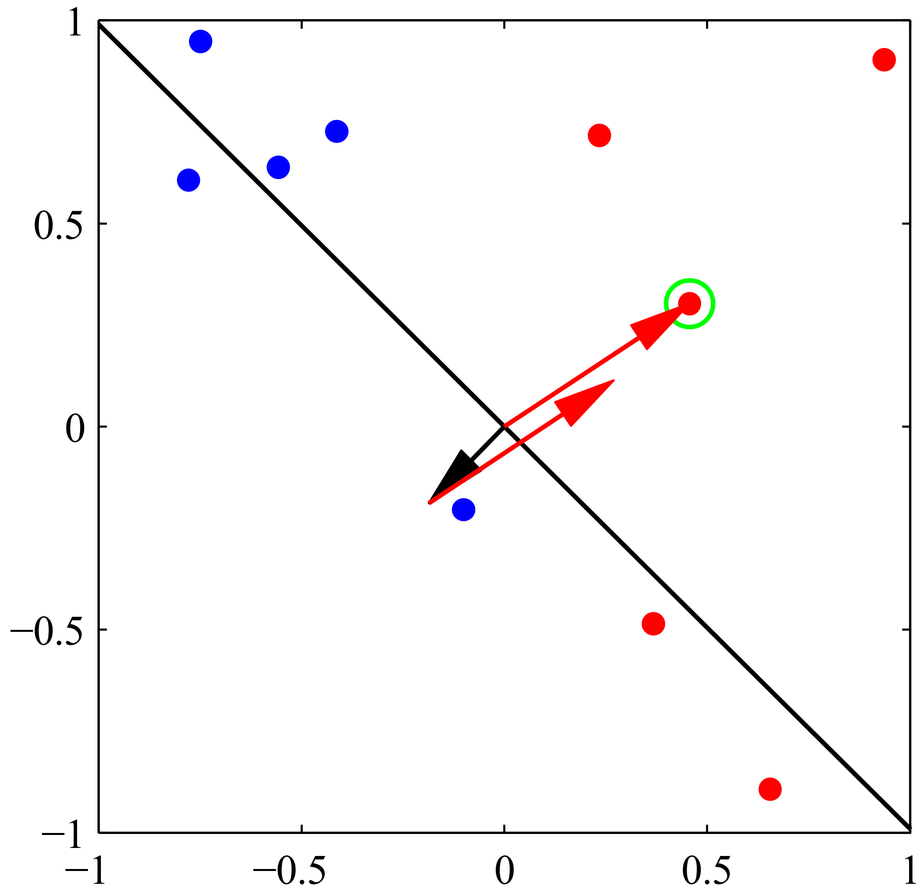


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

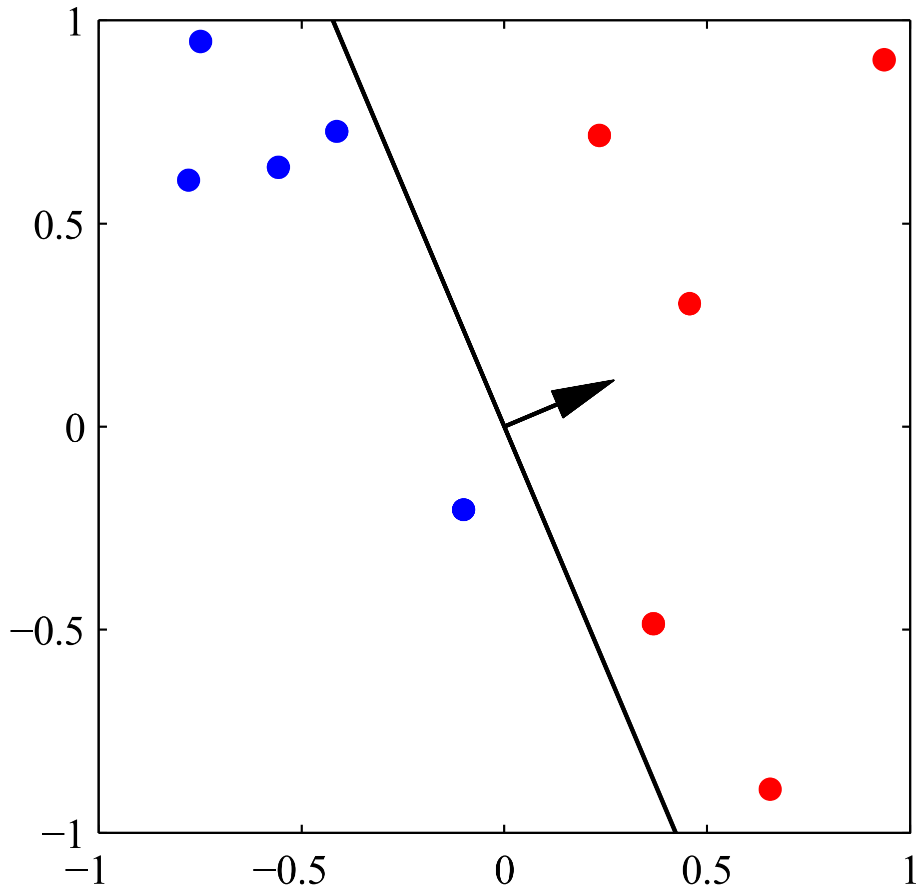


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):



Red is the positive class

Blue is the negative class

Multi-class Classification: Using your linear model

- Regression:

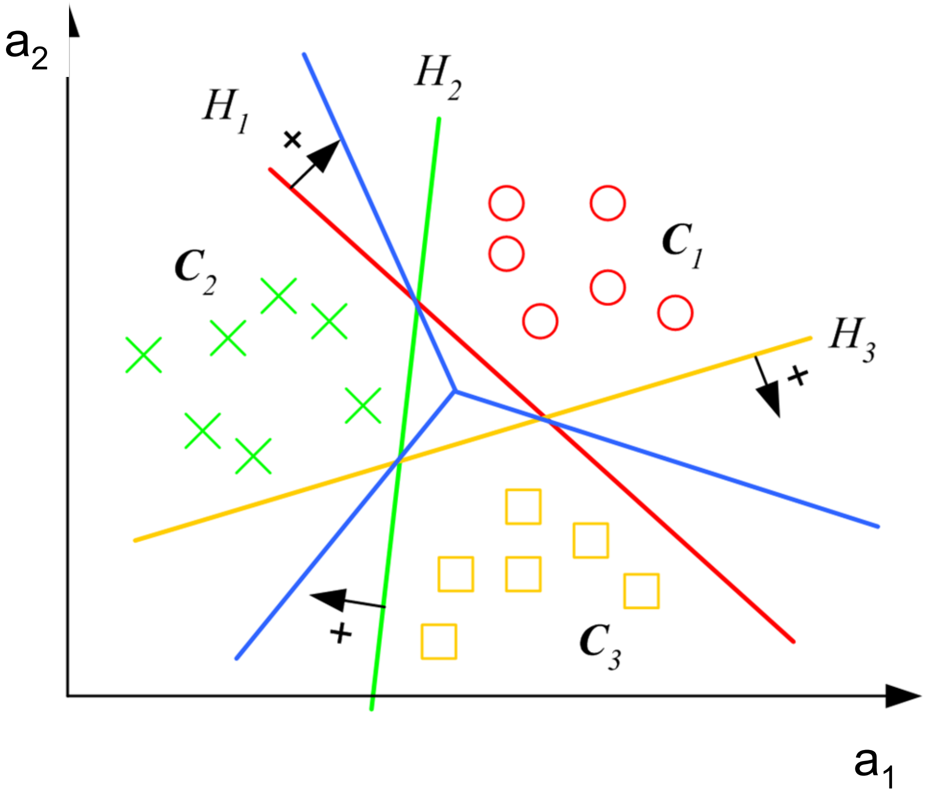
$$\hat{y} = g_\theta(x)$$

- Binary classification:

$$\hat{y} = \begin{cases} 1, & \text{if } g_\theta(x) > 0 \\ -1, & \text{otherwise} \end{cases}$$

- What about classification with more than 2 classes (e.g., *C* classes)? We will discuss 2 approaches:
  - One-vs-One classification
  - One-vs-All classification

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

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

Multi-class Classification

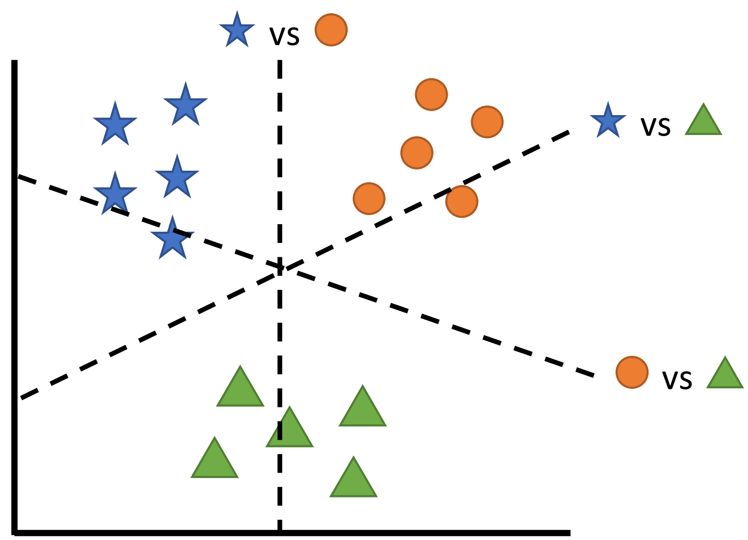
$$c = h(\mathbf{x}) = \underset{c \in C}{\operatorname{argmax}}\, g_c(\mathbf{x})$$

Remember $g_c(\mathbf{x})$ is the inner product of the feature vector for the example $(\mathbf{x})$with the weights of the decision boundary hyperplane for class c. If $g_c(\mathbf{x})$ is getting more positive, that means $(\mathbf{x})$ is deeper inside its "yes" region.

Therefore, if you train a bunch of 2-way classifiers (one for each class) and pick the output of the classifier that says the example is deepest in its region, you have a multi-class classifier.

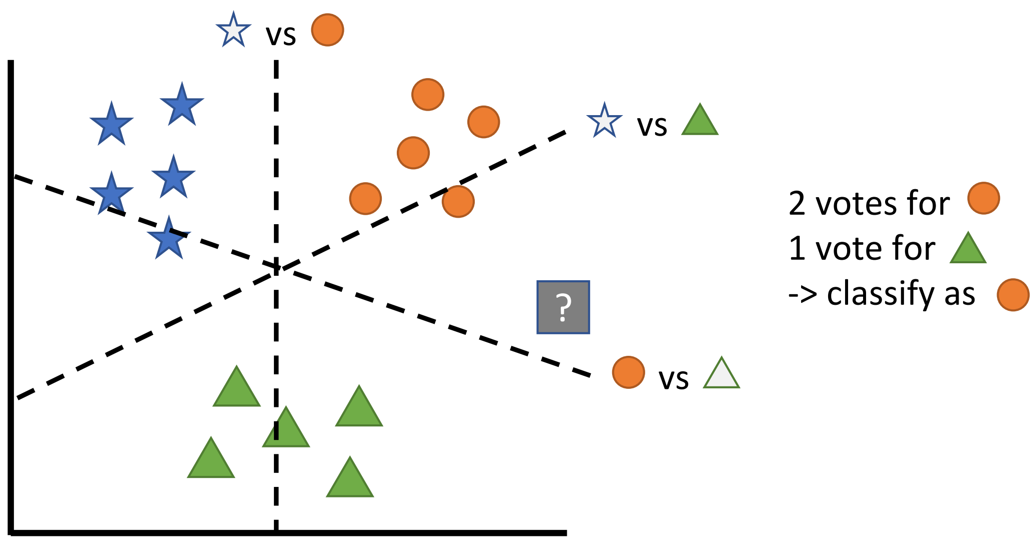
One-vs-One Classification

- Train a binary classifier to disambiguate between each *pair* of classes
- Final prediction is the majority vote among all binary classifiers



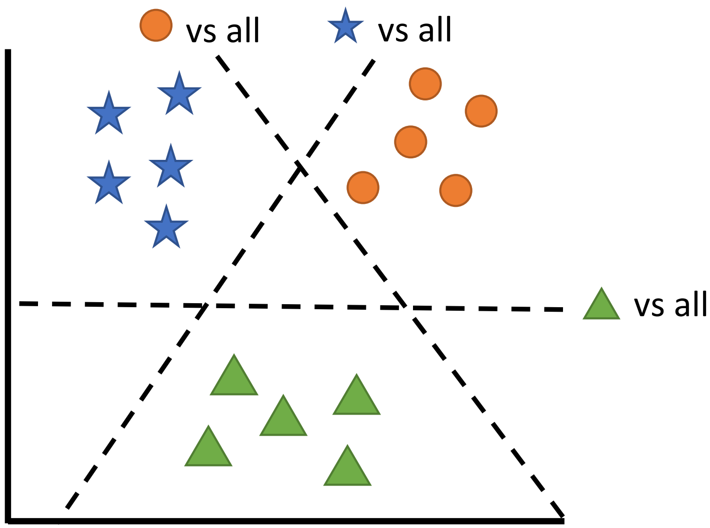
One-vs-One Classification

- Train a binary classifier to disambiguate between each *pair* of classes
- Final prediction is the majority vote among all binary classifiers



One-vs-All Classification

- Train a binary classifier on whether an example does or does not belong to a class
- Predict based on the *highest confidence score* (i.e., the regression output)



$$\hat{y} = \underset{c \in C}{\operatorname{argmax}}\ h_{\theta_c}(x)$$