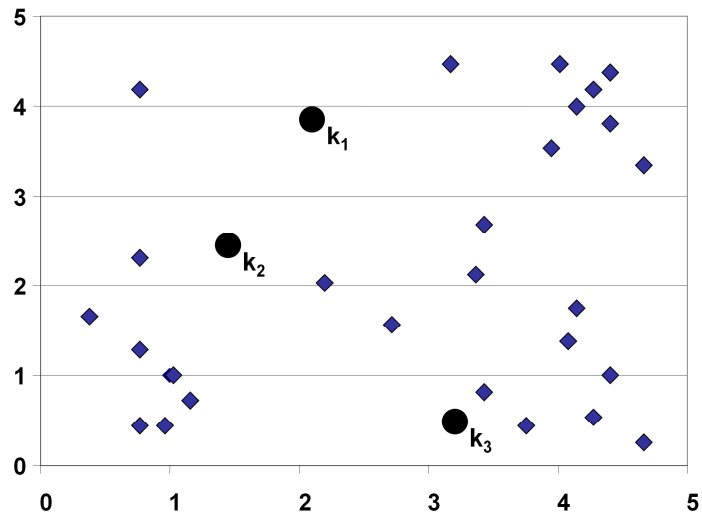


K-means Clustering: Step 1

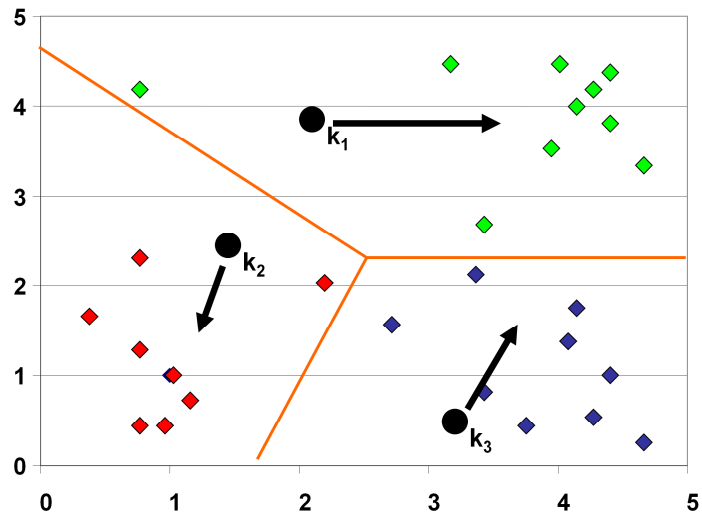
Algorithm: K-means, Distance Metric: Euclidean Distance



Review from last lecture

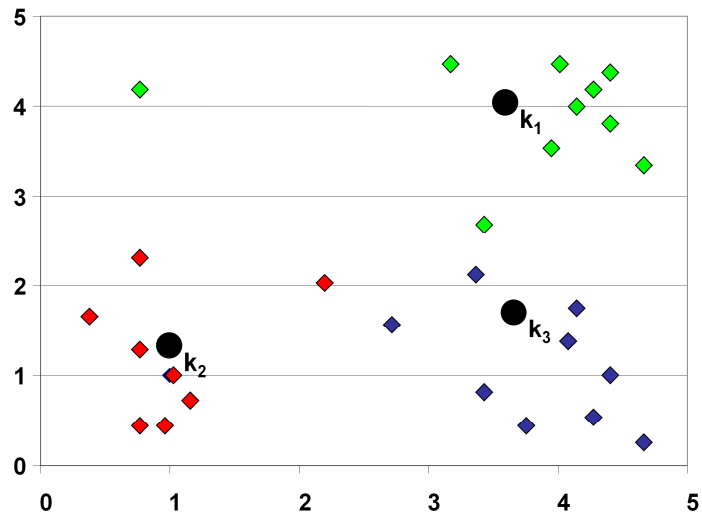
K-means Clustering: Step 2

Algorithm: K-means, Distance Metric: Euclidean Distance



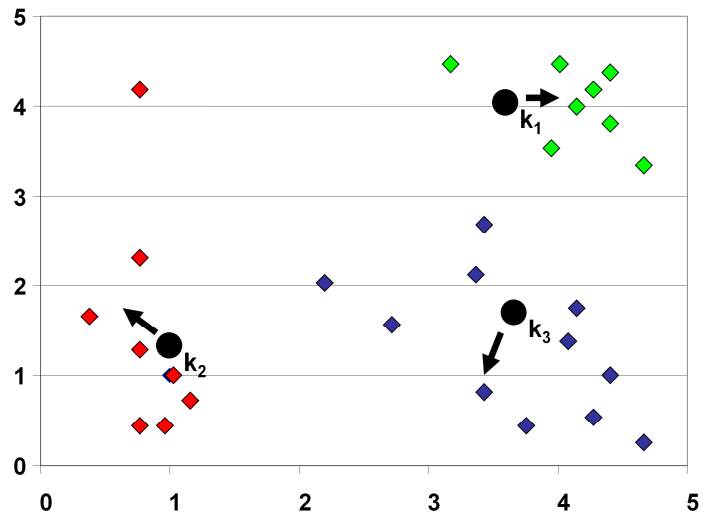
K-means Clustering: Step 3

Algorithm: K-means, Distance Metric: Euclidean Distance



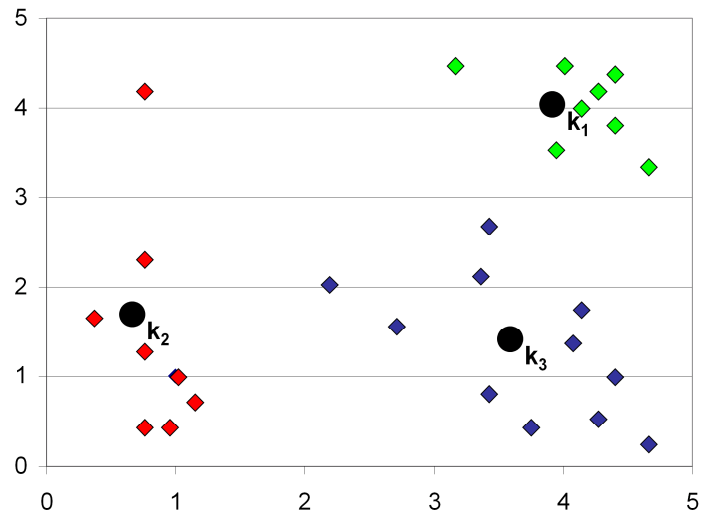
K-means Clustering: Step 4

Algorithm: K-means, Distance Metric: Euclidean Distance



K-means Clustering: Step 5

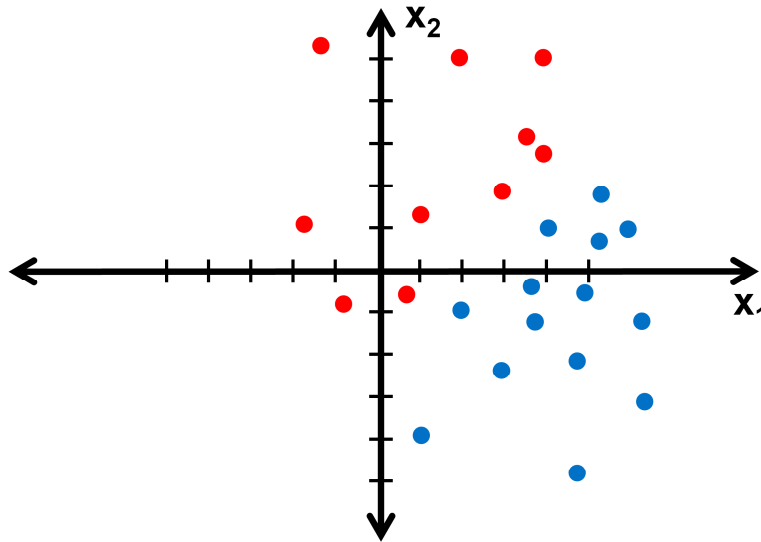
Algorithm: K-means, Distance Metric: Euclidean Distance



Linear Separators

(Russell and Norvig Chapter 20.6)

Two-Dimensional Learning Problem

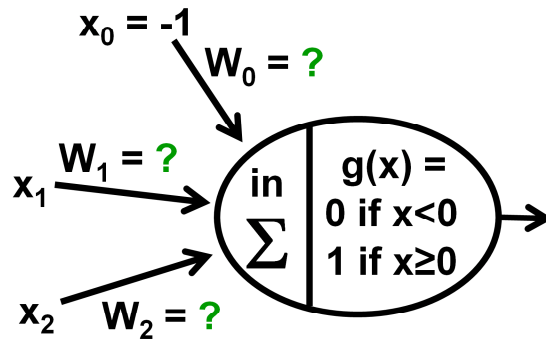


Back to standard machine learning problem. 2 dimensions (2 attributes, x_1 and x_2). 2 classes (red and blue). A classifier would classify each point as red or blue.

The points are clearly linearly separable (blue can be separated from red with a line)

Perceptron Learning

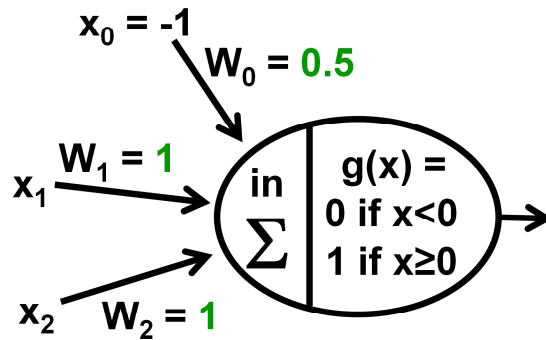
1. Pick some weights arbitrarily



Outputs 1 exactly when:
 $(-1)w_0 + x_1w_1 + x_2w_2 \geq 0$

Perceptron Learning

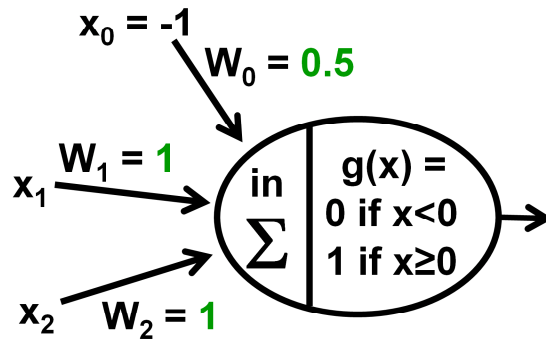
1. Pick some weights arbitrarily



Outputs 1 exactly when:
 $(-1)w_0 + x_1w_1 + x_2w_2 \geq 0$

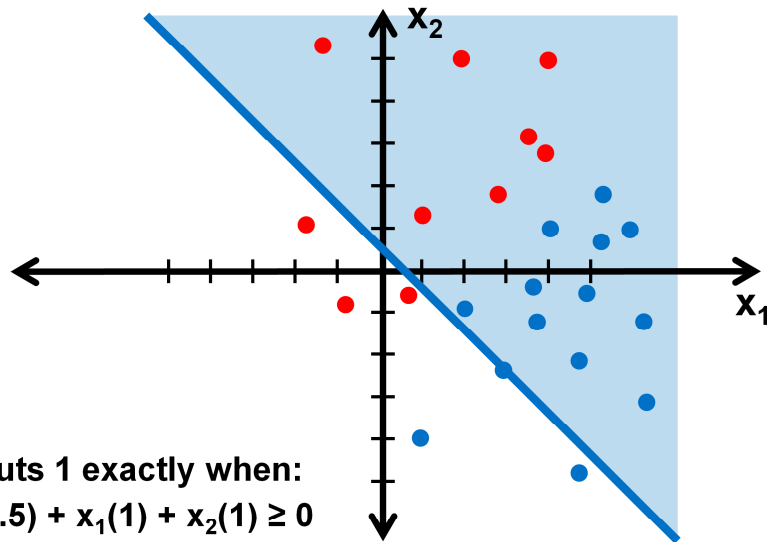
Perceptron Learning

1. Pick some weights arbitrarily



Outputs 1 exactly when:
 $(-1)(0.5) + x_1(1) + x_2(1) \geq 0$

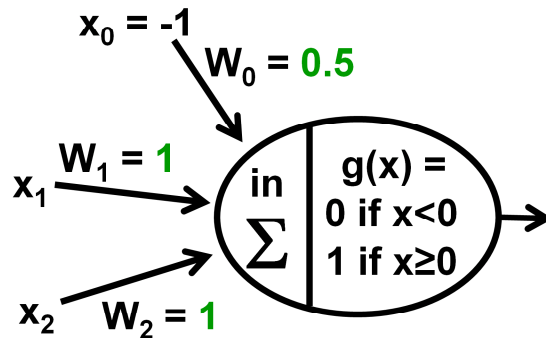
Perceptron Classifier



This is how the perceptron with the initial weights will classify points. Points above the line are classified as 1.

Not very surprising that this isn't very good, as we randomly set the weights

Perceptron Learning



Outputs 1 exactly when:
 $(-1)W_0 + x_1W_1 + x_2W_2 \geq 0$

1. Pick some weights arbitrarily

2. Feed examples one by one and update the weights according to:

$$W_j \leftarrow W_j + \alpha(y - \text{out})x_j$$

(We will use $\alpha = 0.1$)

The algorithm we showed you last time needs to take the derivative of g . Here, g does not have a derivative. For now, we will just ignore that.

Perceptron Classifier

$$-W_0 + x_1W_1 + x_2W_2 \geq 0$$

$$W_0 = 0.5$$

$$W_1 = 1$$

$$W_2 = 1$$

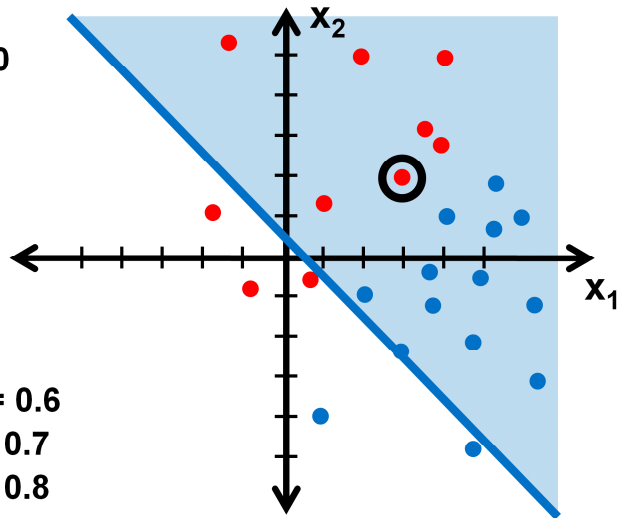
Update weights:

$$W_j \leftarrow W_j + \alpha(y - \text{out})x_j$$

$$W_0 \leftarrow W_0 + \alpha(0 - 1)(-1) = 0.6$$

$$W_1 \leftarrow W_1 + \alpha(0 - 1)(3) = 0.7$$

$$W_2 \leftarrow W_2 + \alpha(0 - 1)(2) = 0.8$$



So here's what's happening in a perceptron with those weights.

Picked a mislabeled point, and update the weights

Perceptron Classifier

$$-W_0 + x_1W_1 + x_2W_2 \geq 0$$

$$W_0 = 0.6$$

$$W_1 = 0.7$$

$$W_2 = 0.8$$

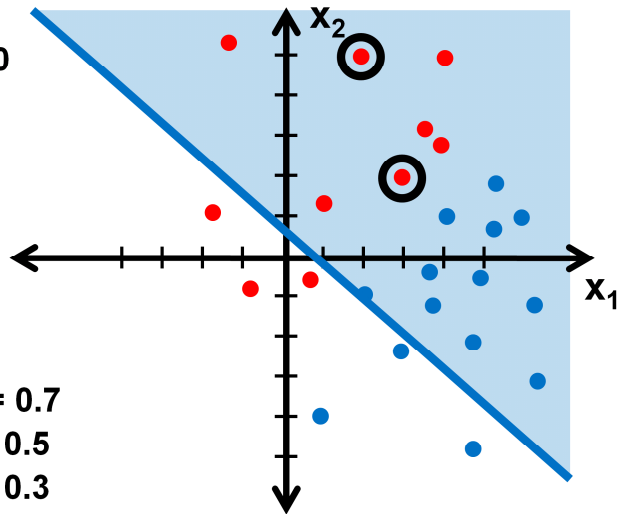
Update weights:

$$W_j \leftarrow W_j + \alpha(y - \text{out})x_j$$

$$W_0 \leftarrow W_0 + \alpha(0 - 1)(-1) = 0.7$$

$$W_1 \leftarrow W_1 + \alpha(0 - 1)(2) = 0.5$$

$$W_2 \leftarrow W_2 + \alpha(0 - 1)(5) = 0.3$$



Pick another mislabeled point, update weights again

Perceptron Classifier

$$-W_0 + x_1W_1 + x_2W_2 \geq 0$$

$$W_0 = 0.7$$

$$W_1 = 0.5$$

$$W_2 = 0.3$$

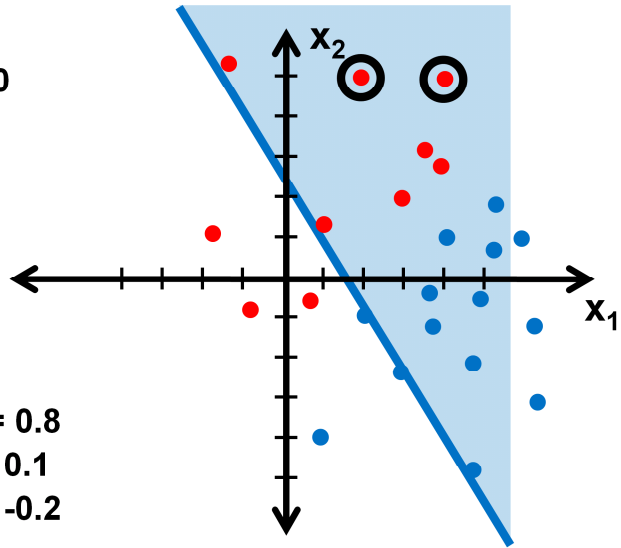
Update weights:

$$W_j \leftarrow W_j + \alpha(y - \text{out})x_j$$

$$W_0 \leftarrow W_0 + \alpha(0 - 1)(-1) = 0.8$$

$$W_1 \leftarrow W_1 + \alpha(0 - 1)(4) = 0.1$$

$$W_2 \leftarrow W_2 + \alpha(0 - 1)(5) = -0.2$$



And again...

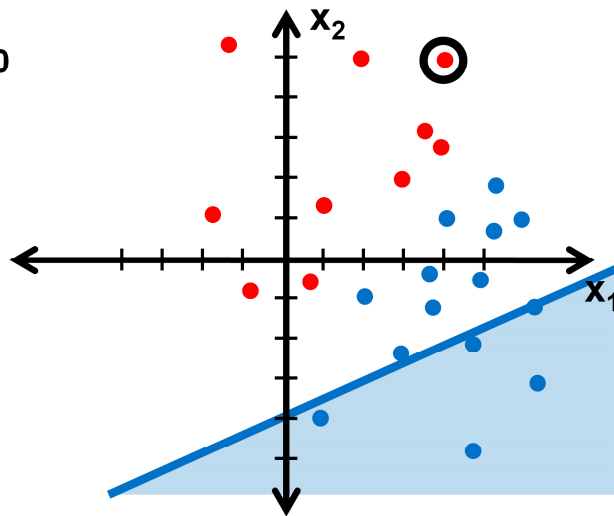
Perceptron Classifier

$$-W_0 + x_1W_1 + x_2W_2 \geq 0$$

$$W_0 = 0.8$$

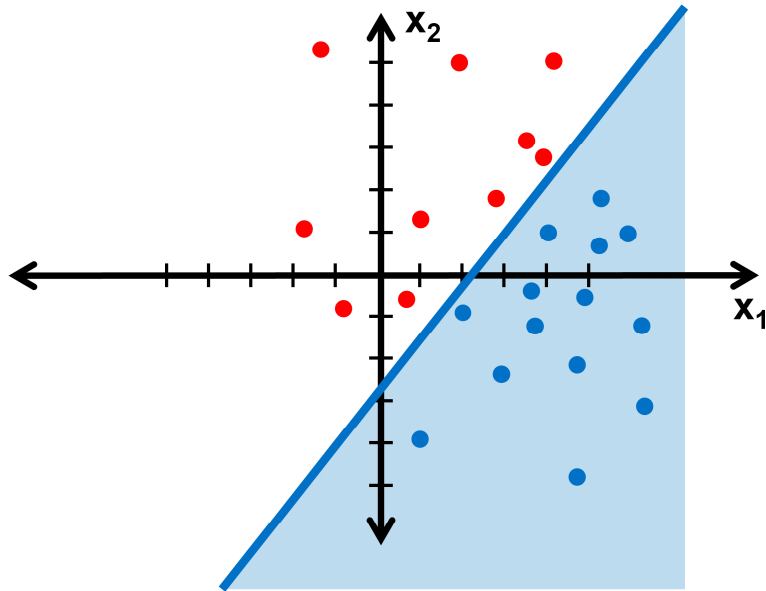
$$W_1 = 0.1$$

$$W_2 = -0.2$$



Note that each time we pick a blue point essentially nothing happens.

After Many Iterations

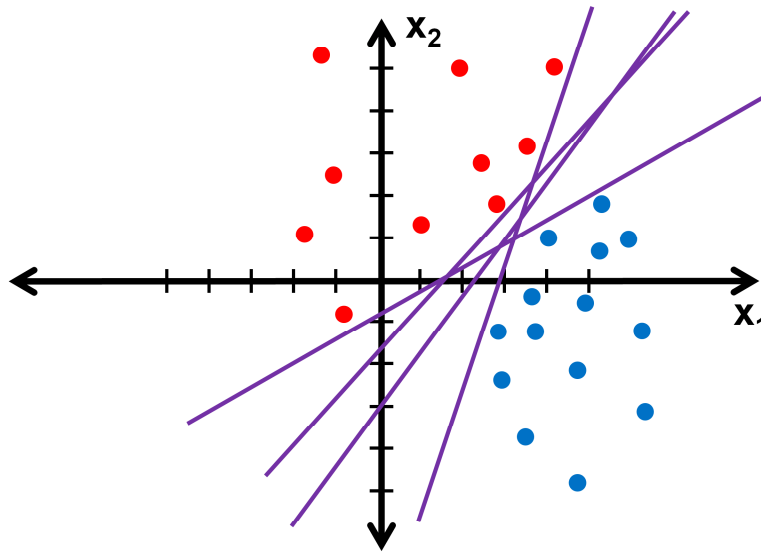


Note: w_0 could only ever change by α . If α is too small, it will move too slow. If α is too big, you might not have the resolution to find the answer.

**How does α affect
the learning?**

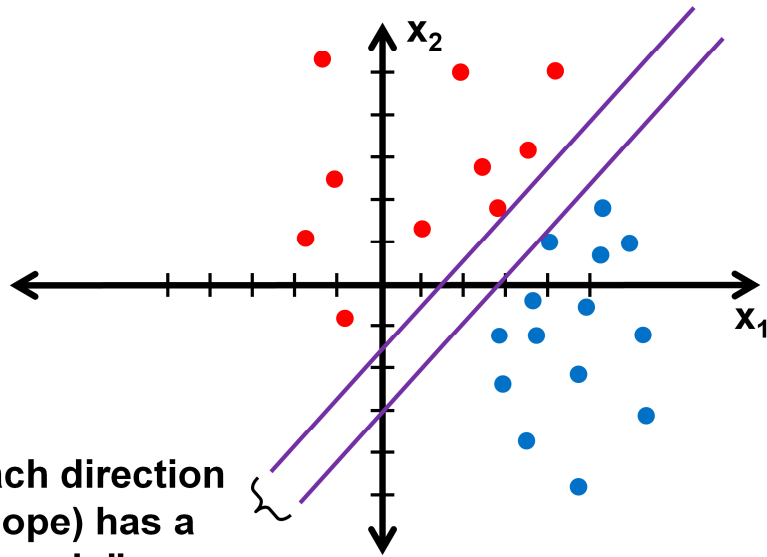
$$\mathbf{W}_j \leftarrow \mathbf{W}_j + \alpha(\mathbf{y} - \text{out})\mathbf{x}_j$$

Many Possible Separators



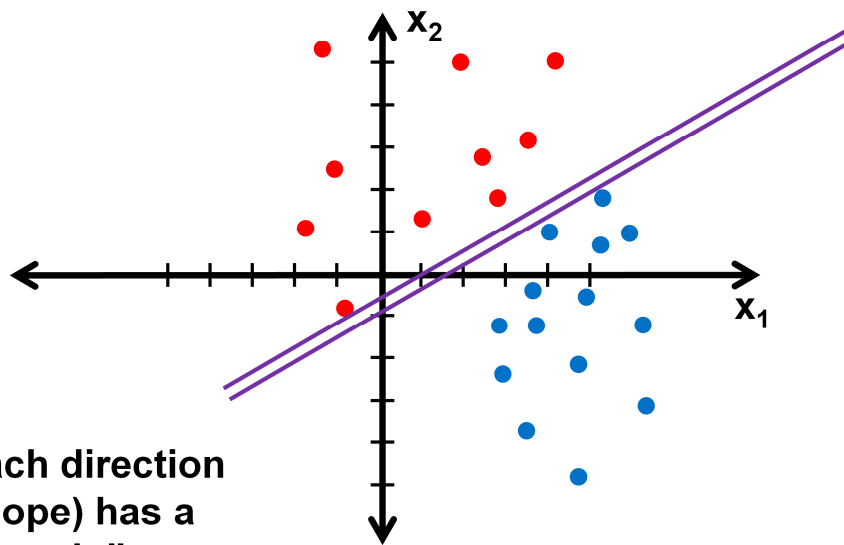
All these lines separate red from blue. But is one better than the other?

Which is Best?



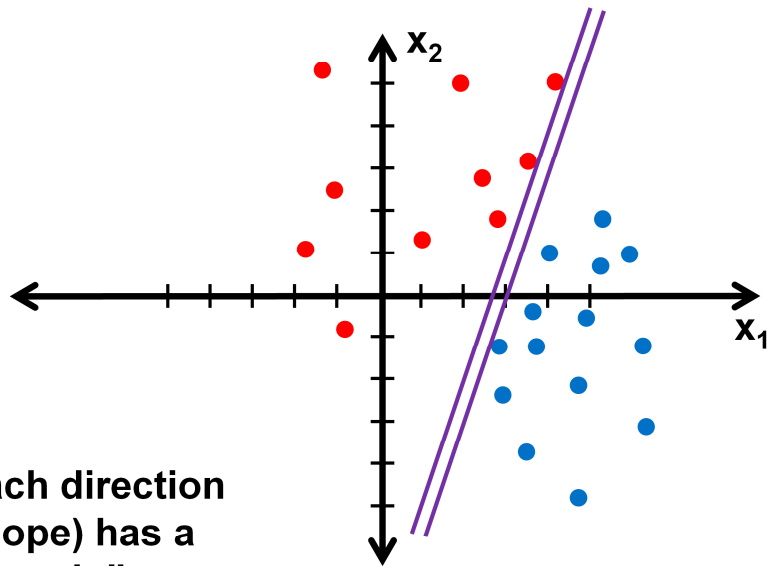
Each direction
(slope) has a
"margin"

Which is Best?



Each direction
(slope) has a
“margin”

Which is Best?

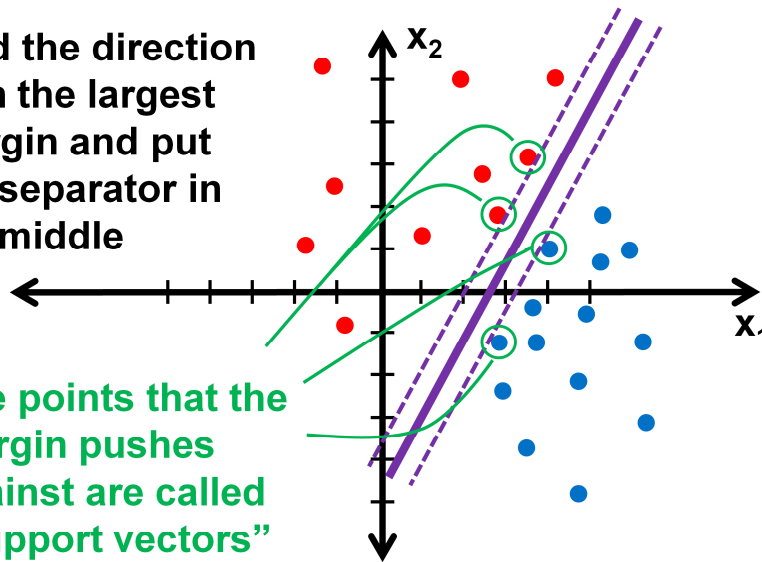


Each direction
(slope) has a
“margin”

Idea: Maximizing Margin

Find the direction with the largest margin and put the separator in the middle

The points that the margin pushes against are called “support vectors”



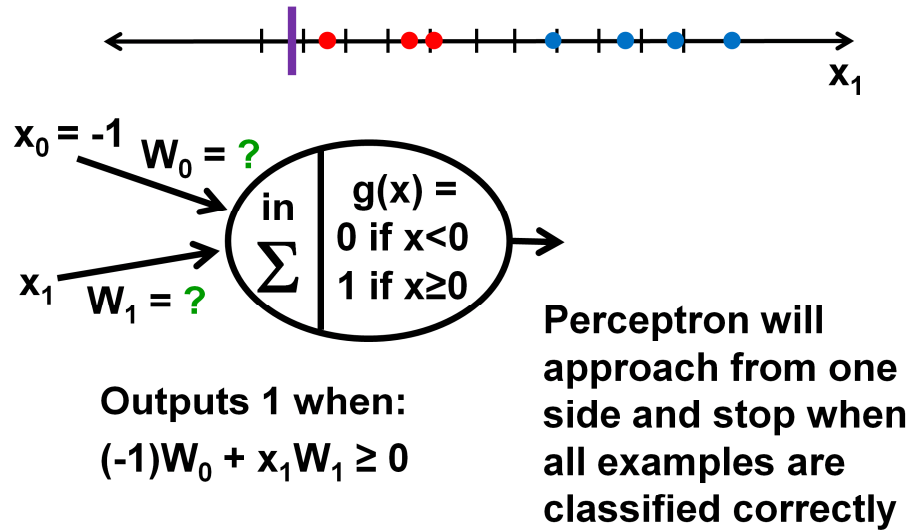
Pick the direction with the largest margin, and then put the separator in the middle of the margin. There is a pretty efficient algorithm to find this separator (but the algorithm is a bit mathy to present here)

When the data is not linearly separable then there's a thing called “soft margin” you can look at.

The points on the dotted line are considered the “support vectors”

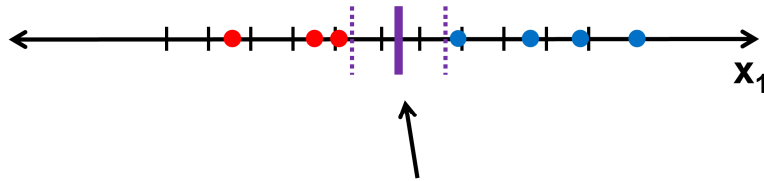
**Given all the examples,
the margin can be
maximized efficiently**

One-Dimensional Learning



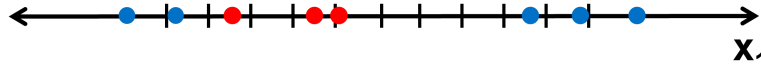
Running a perceptron on this data

One-Dimensional Learning



The maximum margin separator
will be between the two “support
vectors”

One-Dimensional Learning

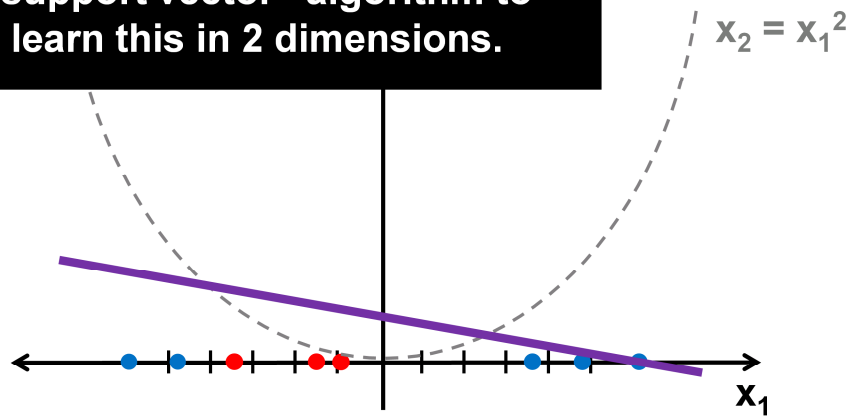


The data isn't linearly separable. Boo.

How can we make this data linearly separable?

Mapping These Data to 2 Dimensions Makes it Linearly Separable!

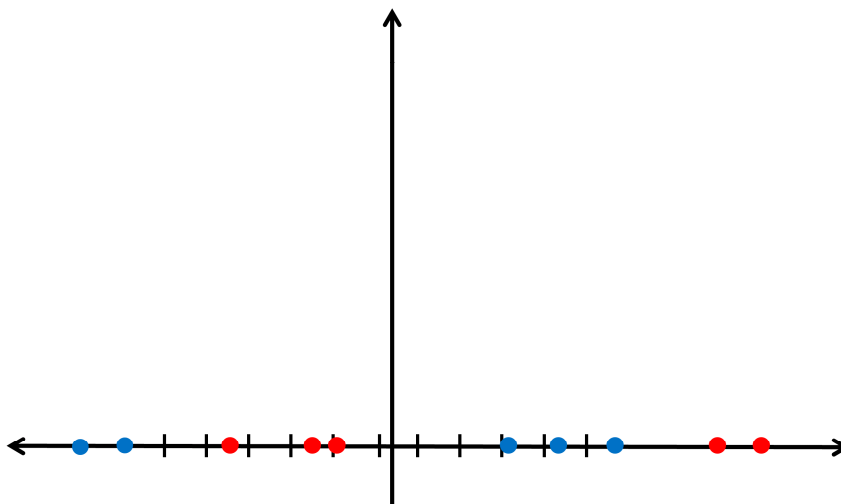
We can use a perceptron or a
“support vector” algorithm to
learn this in 2 dimensions.



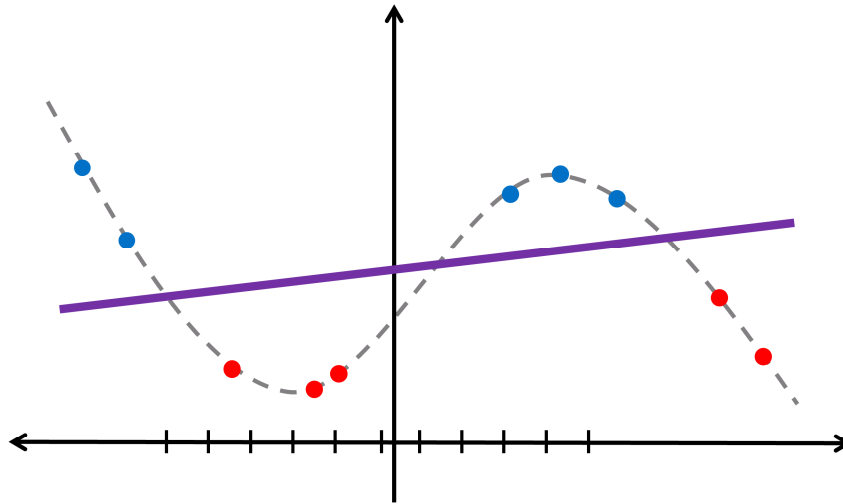
But you can make it linearly separable by using a transformation . For example, you can make points (x_1, x_1^2) .

By mapping the one dimensional examples to higher dimensions you can make them linearly separable!

How to do the mapping?



How to do the mapping?



Oh hey check this out mapping to higher dimensions is actually a pretty good idea. The trick is just choosing how to map it.

This works in higher dimensions:
<http://www.youtube.com/watch?v=3liCbRZPrZA>

This is very pretty, watch it!

Demo:

<http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml>