Linear Models

Lecture 5

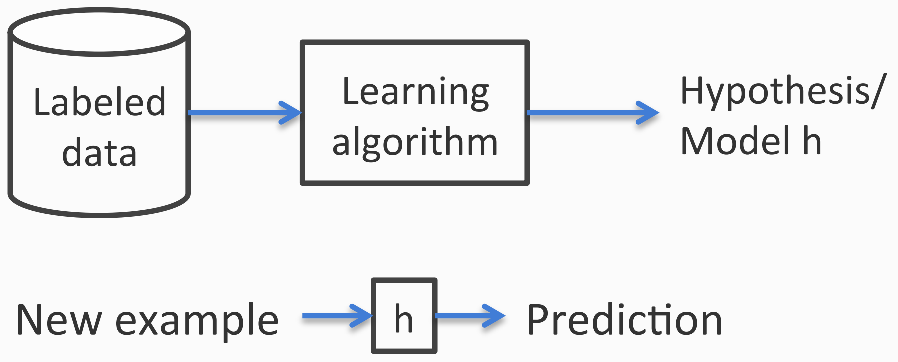
Machine Learning
Fall 2015

THE UNIVERSITY OF UTAH

Checkpoint: The bigger picture

- Supervised learning: instances, concepts, and hypotheses

- Specific learners
  - Decision trees
  - Nearest neighbors



- General ML ideas
  - Features as high dimensional vectors
  - Overfitting
  - The curse of dimensionality

Questions?

This lecture

- Linear classifiers
- Expressivity of linear classifiers
- Least Squares Method for Regression
- Learning linear classifiers: The lay of the land

Where are we?

- Linear classifiers
- Expressivity of linear classifiers
- Least Squares Method for Regression
- Learning linear classifiers: The lay of the land

Is learning possible at all?

- There are $2^{16} = 65536$ possible Boolean functions over 4 inputs
  - Why? There are 16 possible outputs. Each way to fill these 16 slots is a different function, giving 2^{16} functions.
- We have seen only 7 outputs
- We *cannot* know what the rest are without seeing them
  - Think of an adversary filling in the labels every time you make a guess at the function

x_1	x_2	x_3	x_4	y
0	0	0	0	?
0	0	0	1	?
0	0	1	0	**0** $\leftarrow$
0	0	1	1	**1** $\leftarrow$
0	1	0	0	**0** $\leftarrow$
0	1	0	1	**0** $\leftarrow$
0	1	1	0	**0** $\leftarrow$
0	1	1	1	?
1	0	0	0	?
1	0	0	1	**1** $\leftarrow$
1	0	1	0	?
1	0	1	1	?
1	1	0	0	**0** $\leftarrow$
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

Is learning possible at all?

- There are 2^{16} = 65536 possible Boolean functions over 4 inputs
  - Why? There are 16 possible outputs. Each way to fill these 16 slots is a different function, giving 2^{16} functions.

How could we possibly learn anything?

- We have seen only 7 outputs
- We *cannot* know what the rest are without seeing them
  - Think of an adversary filling in the labels every time you make a guess at the function

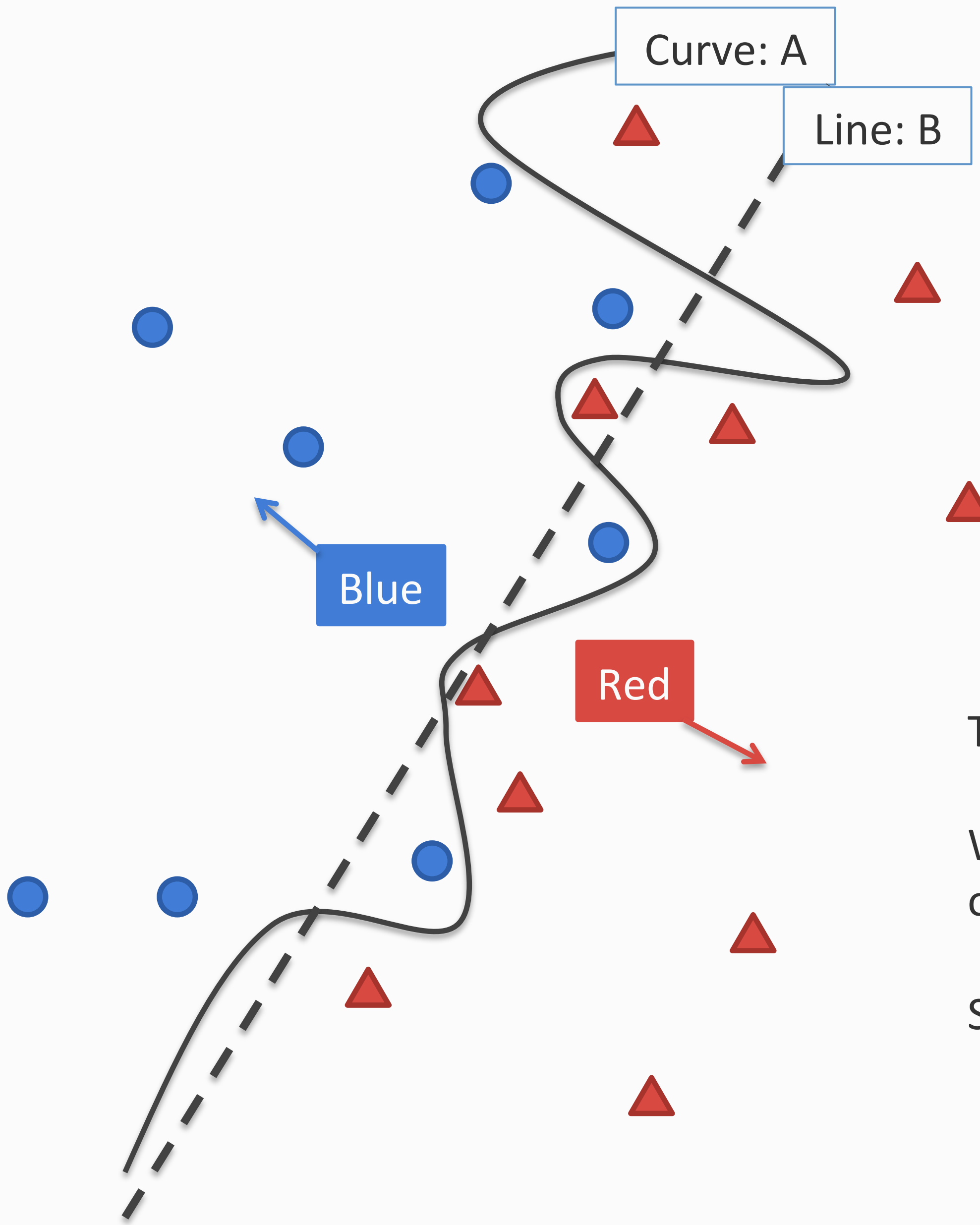
x_1	x_2	x_3	x_4	y	
0	0	0	0	?	
0	0	0	1	?	
0	0	1	0	**0**	$\leftarrow$
0	0	1	1	**1**	$\leftarrow$
0	1	0	0	**0**	$\leftarrow$
0	1	0	1	**0**	$\leftarrow$
[illegible]	[illegible]	[illegible]	0	**0**	$\leftarrow$
0	1	1	1	?	
1	0	0	0	?	
1	0	0	1	**1**	$\leftarrow$
1	0	1	0	?	
1	0	1	1	?	
1	1	0	0	**0**	$\leftarrow$
1	1	0	1	?	
1	1	1	0	?	
1	1	1	1	?	

Solution: Restrict the search space

- A *hypothesis space* is the set of possible functions we consider
  - We were looking at the space of all Boolean functions
  - Instead choose a hypothesis space that is smaller than the space of all functions
    - Only simple conjunctions (with four variables, there are only 16 conjunctions without negations)
    - Simple disjunctions
    - m-of-n rules: Fix a set of n variables. At least m of them must be true
    - Linear functions

      ⋮

Which is the better classifier?



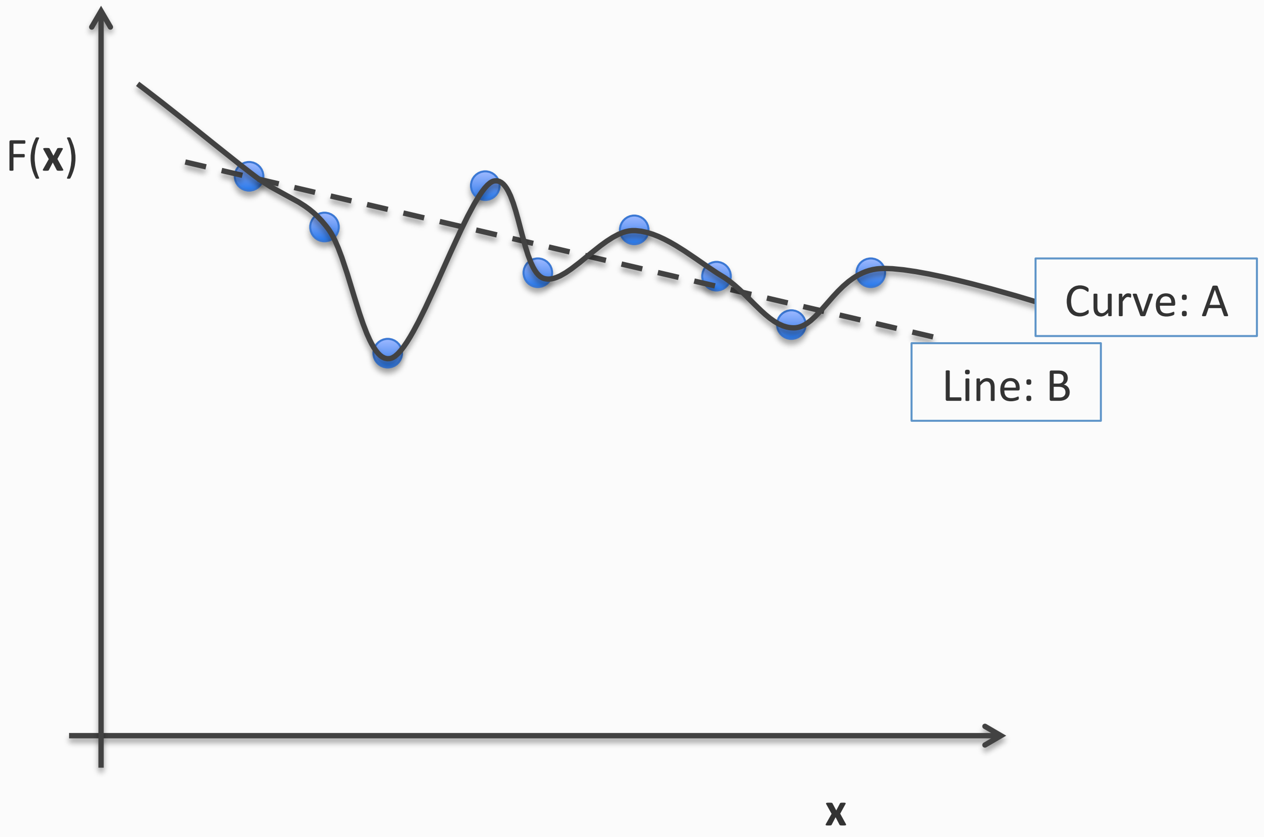
Suppose this our training set and we have to separate the blue circles from the red triangles

Think about overfitting

Which curve runs the risk of overfitting?

Simplicity vs Accuracy

Similar argument for regression



Linear regression might make smaller errors on new points

Linear Classifiers

Input is a n dimensional vector **x**

Output is a label $y \in \{-1, 1\}$

Linear Threshold Units classify an example **x** using a parameters **w** (a vector) and b (a real number) according the following classification rule

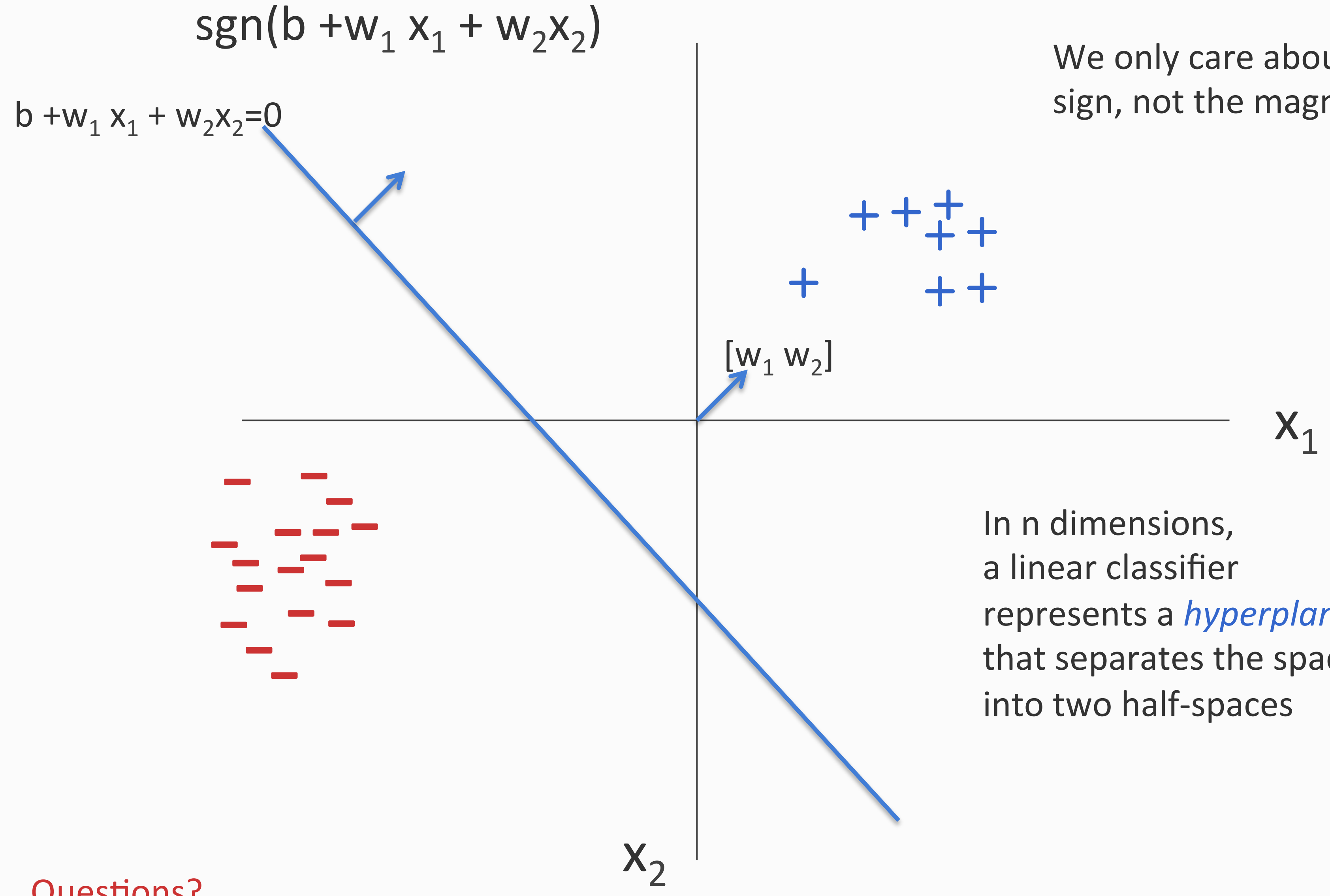
$$\text{Output} = \text{sign}(\mathbf{w}^T\mathbf{x} + b) = \text{sign}(b + \sum w_i x_i)$$

$$\mathbf{w}^T\mathbf{x} + b \geq 0 \Rightarrow \text{Predict } y = 1$$

$$\mathbf{w}^T\mathbf{x} + b < 0 \Rightarrow \text{Predict } y = -1$$

b is called the bias term

The geometry of a linear classifier



$\mathrm{sgn}(b + w_1 x_1 + w_2 x_2)$
$b + w_1 x_1 + w_2 x_2 = 0$
$[w_1\ w_2]$
x_1
x_2

We only care about the sign, not the magnitude

In n dimensions,
a linear classifier
represents a *hyperplane*
that separates the space
into two half-spaces

Questions?

Where are we?

- Linear classifiers
- Expressivity of linear classifiers
- Least Squares Method for Regression
- Learning linear classifiers: The lay of the land

Which Boolean functions can linear classifiers represent?

- Linear classifiers are an expressive hypothesis class

- Many Boolean functions are linearly separable
  - Not all though
  - Recall: Decision trees can represent any Boolean function

Conjunctions and disjunctions

$y = x_1 \wedge x_2 \wedge x_3$ is equivalent to "y = *1 if* $x_1 + x_2 + x_3 \geq 3$"

x_1	x_2	x_3	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Conjunctions and disjunctions

$y = x_1 \wedge x_2 \wedge x_3$ is equivalent to "y = 1 *if* $x_1 + x_2 + x_3 \geq 3$"

x_1	x_2	x_3	y	$x_1 + x_2 + x_3 - 3$	sign
0	0	0	0	-3	0
0	0	1	0	-2	0
0	1	0	0	-2	0
0	1	1	0	-1	0
1	0	0	0	-2	0
1	0	1	0	-1	0
1	1	0	0	-1	0
1	1	1	1	0	1

Negations are okay too

$y = x_1 \wedge x_2 \wedge \neg x_3$

is equivalent to

y' = 1 *if* $x_1 + x_2 - x_3 \geq 2$

Exercise: What would the linear threshold function be if the conjunctions here were replaced with disjunctions?

Questions?

m-of-n functions

m-of-n rules

- There is a fixed set of n variables
- y = `true` if, and only if, at least m of them are `true`
- All other variables are ignored

Suppose there are five Boolean variables: x_1, x_2, x_3, x_4, x_5

What is a linear threshold unit that is equivalent to the classification rule "at least 2 of $\{x_1, x_2, x_3\}$"?

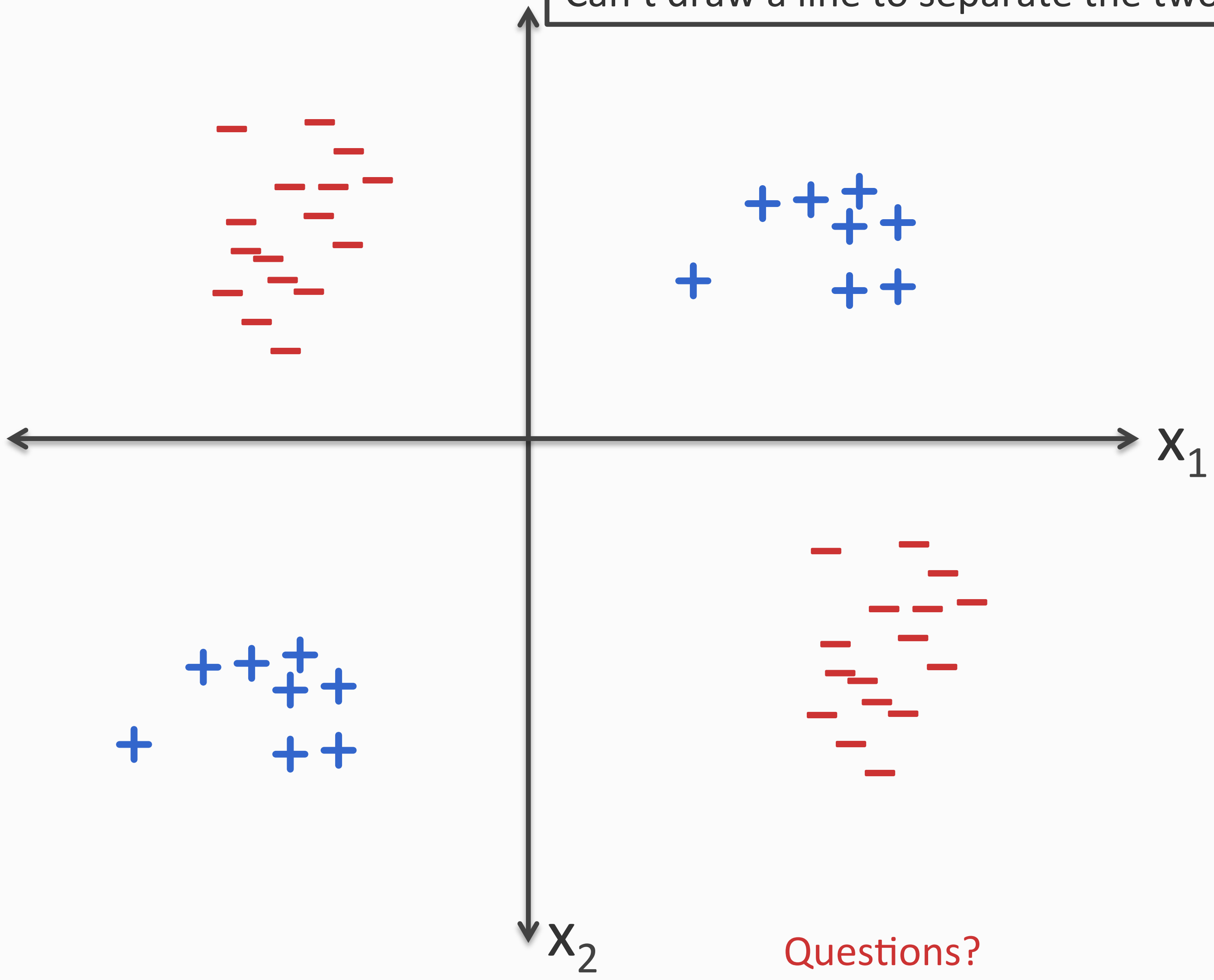
$$x_1 + x_2 + x_3 \geq 2$$

Questions?

Parity is not linearly separable

(The XOR function)

Can't draw a line to separate the two classes



Questions?

Not all functions are linearly separable

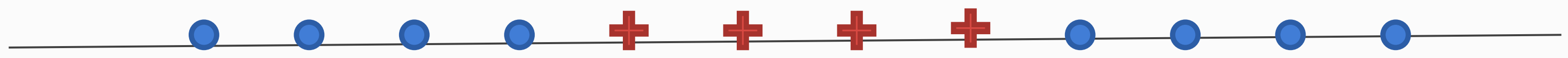
- XOR is not linear
  - $y = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$
  - *Parity* cannot be represented as a linear classifier
    - f(**x**) = 1 if the number of 1's is even

- Many non-trivial Boolean functions
  - $y = (x_1 \wedge x_2) \vee (x_3 \wedge \neg x_4)$
  - The function is not linear in the four variables

Even these functions can be *made* linear

These points are not separable in 1-dimension by a line

What is a one-dimensional line, by the way?

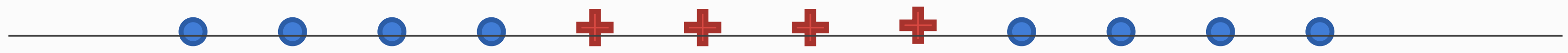


The trick: Change the representation

The blown up feature space

The trick: Use feature *conjunctions*

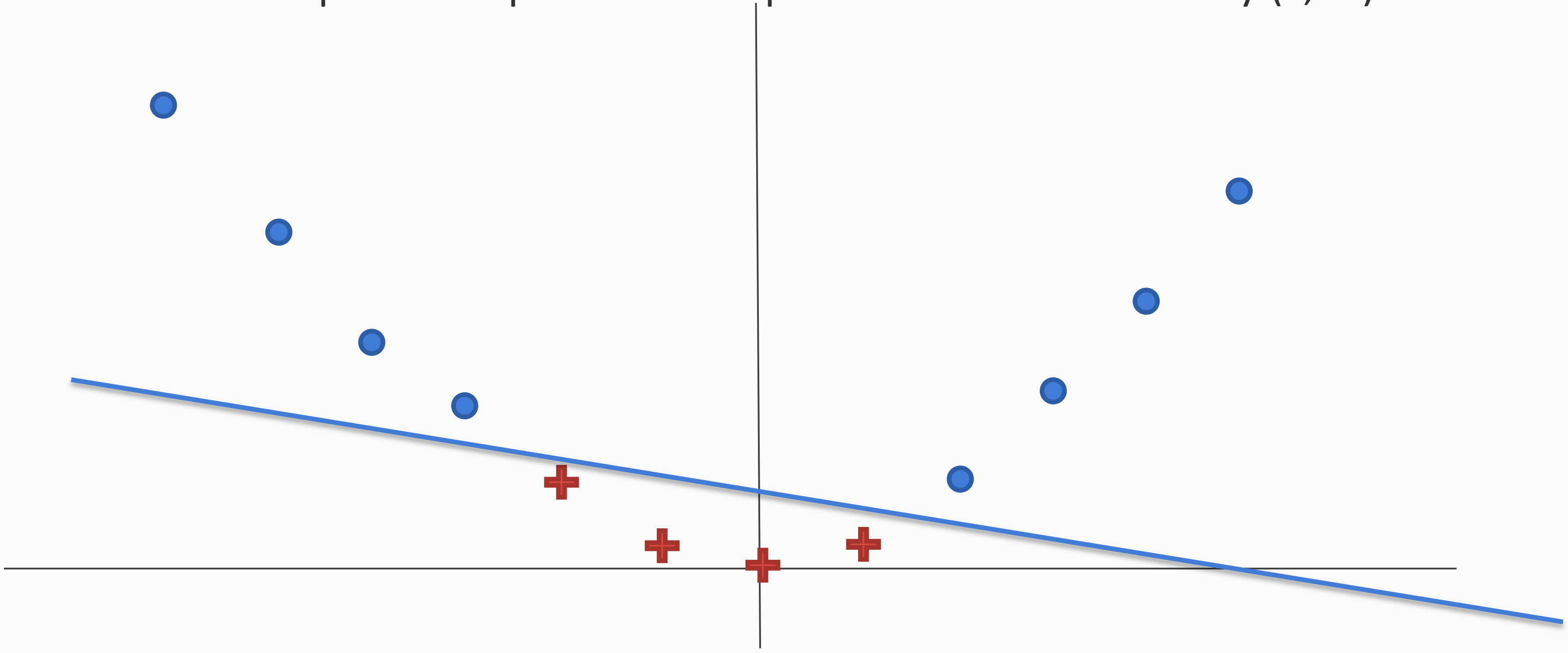
Transform points: Represent each point x in 2 dimensions by (x, x^2)



The blown up feature space

The trick: Use feature *conjunctions*

Transform points: Represent each point x in 2 dimensions by (x, x^2)

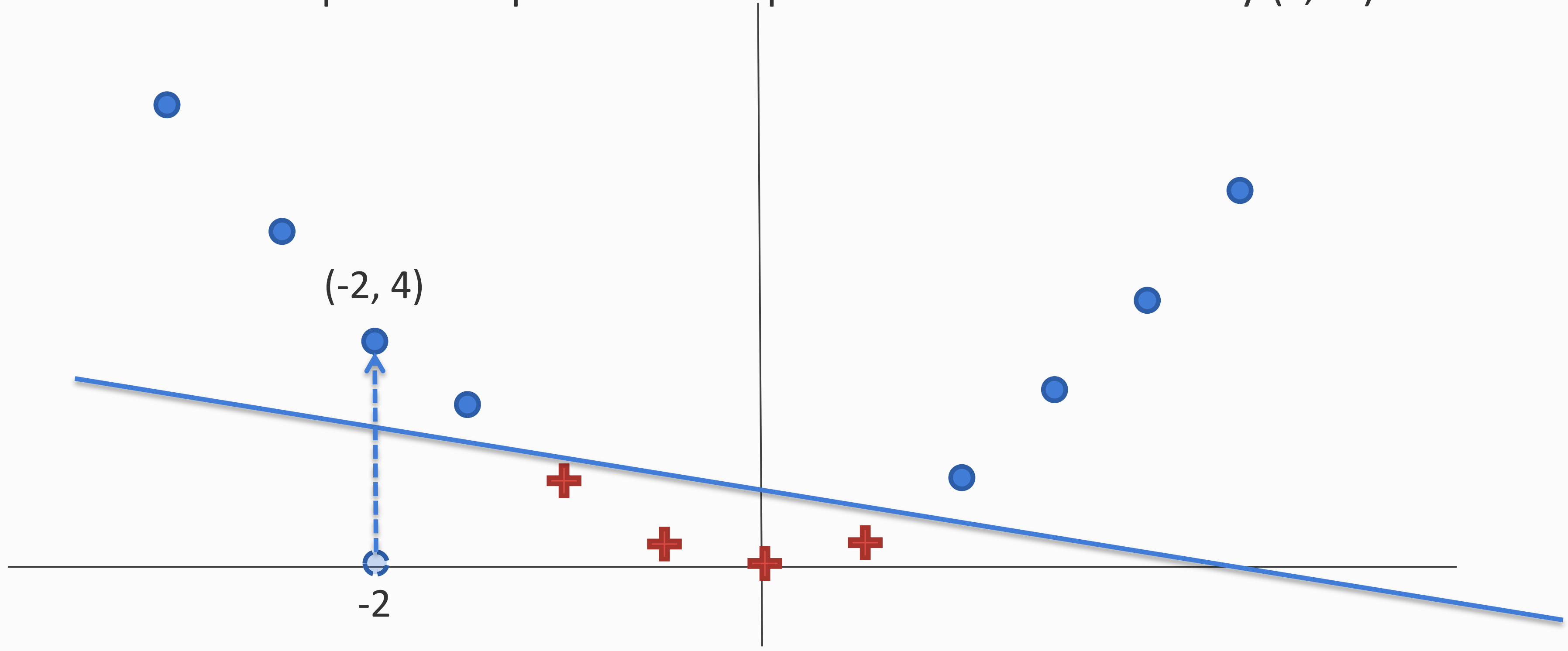


Now the data is linearly separable in this space!

The blown up feature space

The trick: Use feature *conjunctions*

Transform points: Represent each point x in 2 dimensions by (x, x^2)



Now the data is linearly separable in this space!

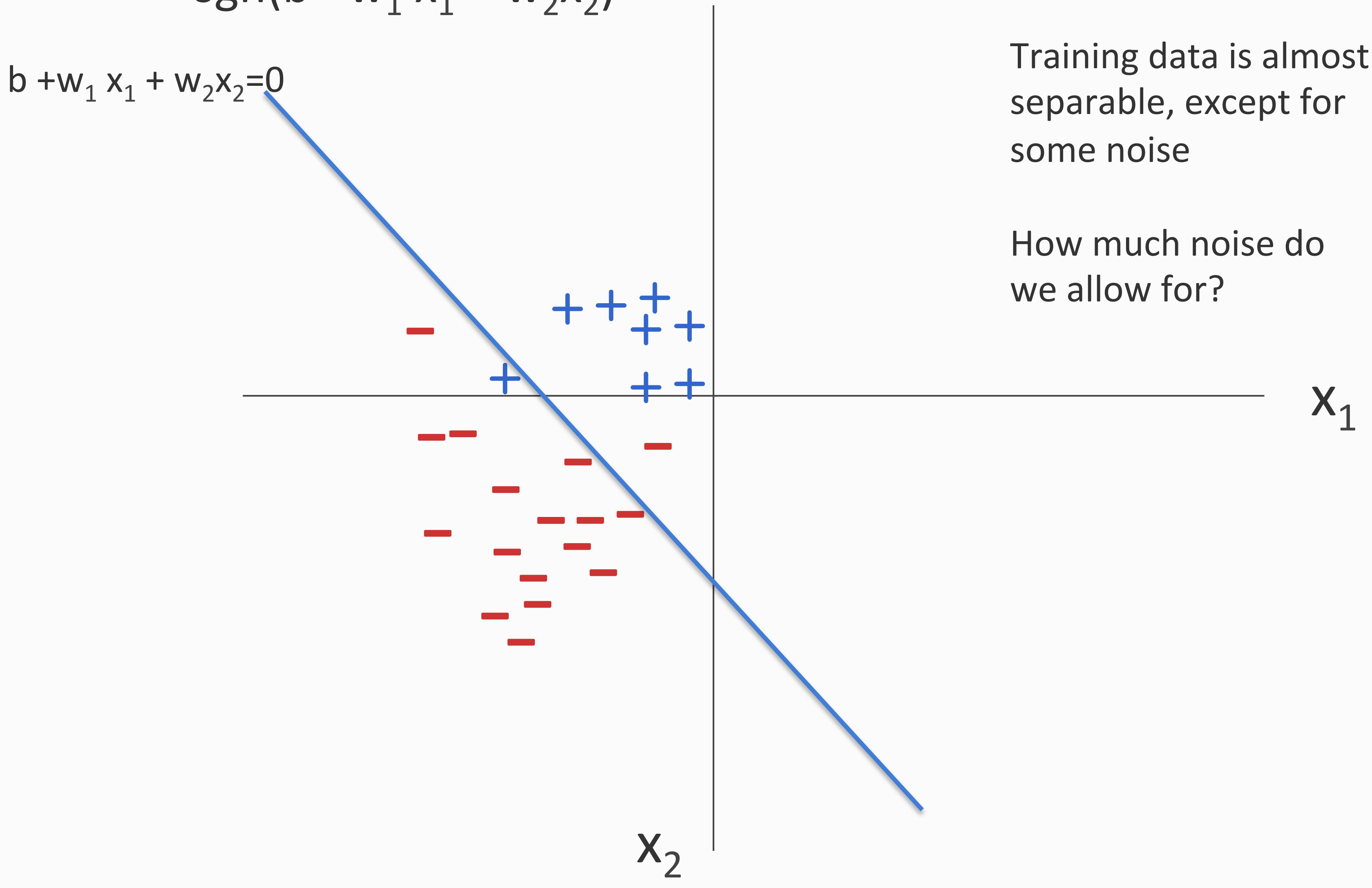
Exercise

How would you use the feature transformation idea to make XOR in two dimensions linearly separable in a new space?

To answer this question, you need to think about a function that maps examples from two dimensional space to a higher dimensional space.

Almost linearly separable data

$\text{sgn}(b + w_1 x_1 + w_2 x_2)$



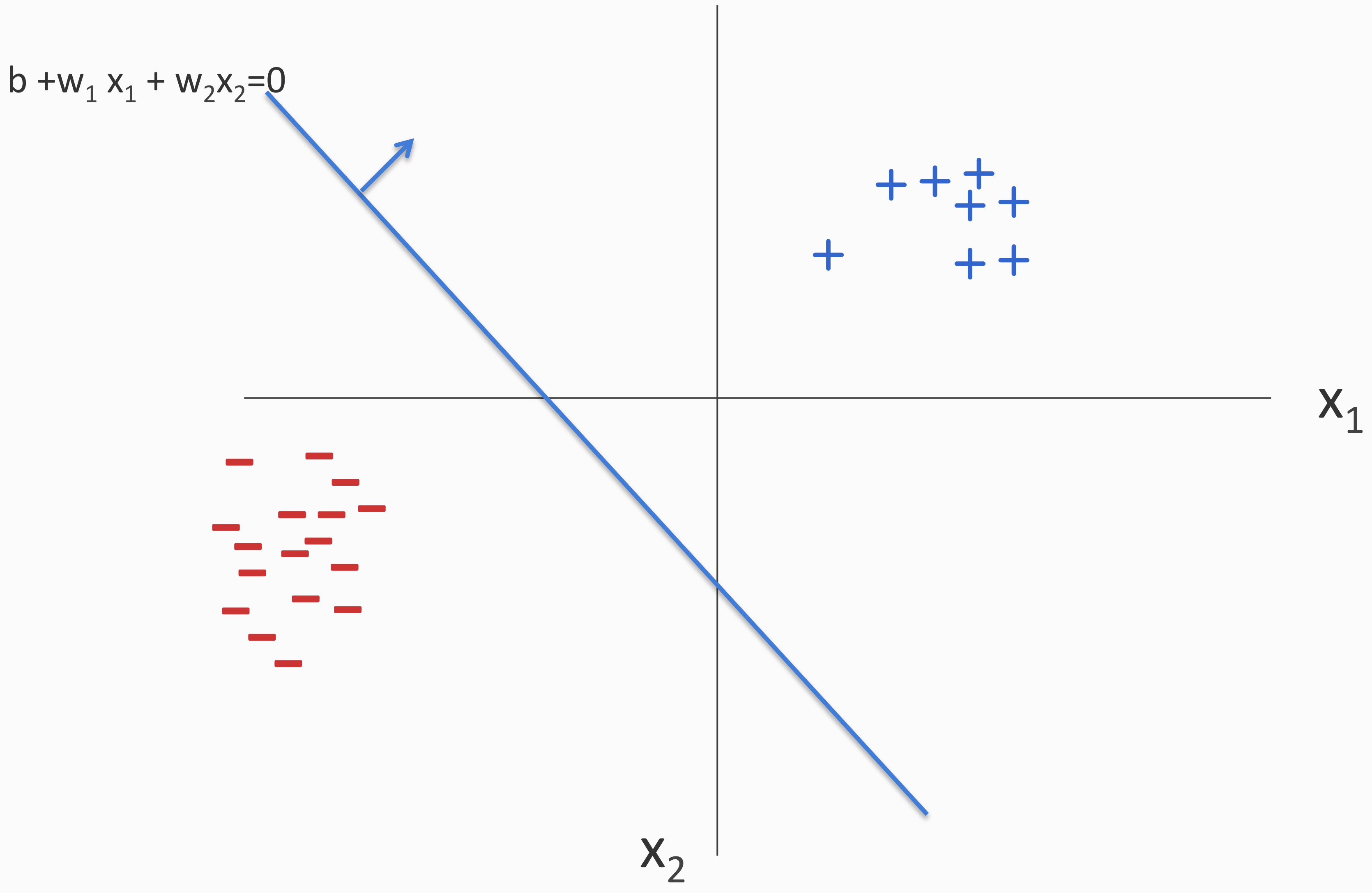
Training data is almost separable, except for some noise

How much noise do we allow for?

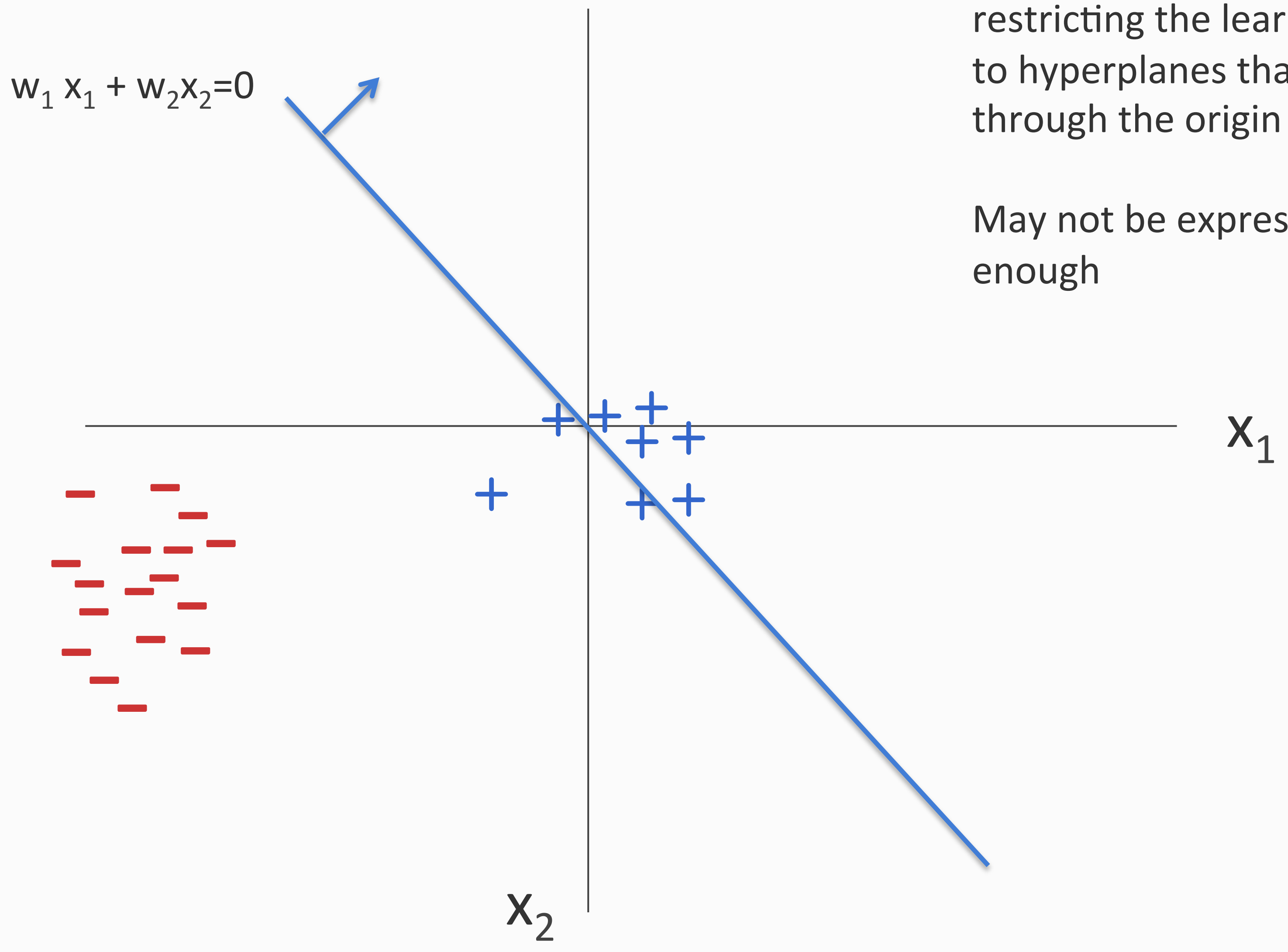
Linear classifiers: An expressive hypothesis class

- Many functions are linear
- Often a good guess for a hypothesis space
- Some functions are not linear
  - The XOR function
  - Non-trivial Boolean functions
- But there are ways of making them linear in a higher dimensional feature space

Why is the bias term needed?



Why is the bias term needed?



If b is zero, then we are restricting the learner only to hyperplanes that go through the origin

May not be expressive enough

Simplifying notation

We can stop writing b at each step using notational sugar:

- The prediction function is $\mathrm{sgn}(b + \mathbf{w}^T\mathbf{x})$
- Rewrite **x** as [1, x] $\rightarrow$ **x'**
- Rewrite **w** as [b, **w**] $\rightarrow$ **w'**
  - Increases dimensionality by one
  - Equivalent to adding a feature that is always 1
- The prediction is now $\mathrm{sgn}(\mathbf{w'}^T\mathbf{x'})$

In the increased dimensional space, **w'** goes through the origin

We sometimes show b, and instead fold the bias term into the input by adding an extra constant feature. But remember that it is there

Questions?

Exercises

1. Represent the simple disjunction as a linear classifier (slide 15)

2. How would you apply the feature space expansion idea for the XOR function?

Exercises

1. Represent the simple disjunction as a linear classifier (slide 15)

   $x_1 \vee x_2 \vee x_3$ can be represented as $x_1 + x_2 + x_3 \geq 1$

2. How would you apply the feature space expansion idea for the XOR function? (slide 19)

   Use features for conjunctions of Boolean variables rather than single Boolean variables

Where are we?

- Linear classifiers
- Expressivity of linear classifiers
- Least Squares Method for regression
  - A detour via regression
- Learning linear classifiers: The lay of the land

What's the mileage?

Suppose we want to predict the mileage of a car from its weight and age

Weight (x 100 lb) x_1	Age (years) x_2	Mileage
31.5	6	21
36.2	2	25
43.1	0	18
27.6	2	30

What we want: A function that can predict mileage using x_1 and x_2

Linear regression: The strategy

Predicting continuous values using a linear model

Assumption: The output is a linear function of the inputs

$$\text{Mileage} = \underline{w_0} + \underline{w_1}\, x_1 + \underline{w_2}\, x_2$$

Parameters of the model
Also called weights
Collectively, a vector

Learning: Using the training data to find the *best* possible value of **w**

Prediction: Given the values for x_1, x_2 for a new car, use the learned **w** to predict the `Mileage` for the new car

Linear regression: The strategy

- Inputs are vectors: $\mathbf{x} \in \Re^d$
- Outputs are real numbers: $y \in \Re$

- We have a training set

$$D = \{ (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_m, y_m)\}$$

- We want to approximate y as

$$y = f_w(x) = w_1 + w_2\, x_2 + \cdots + w_n\, x_n$$
$$= \mathbf{w}^T\mathbf{x}$$

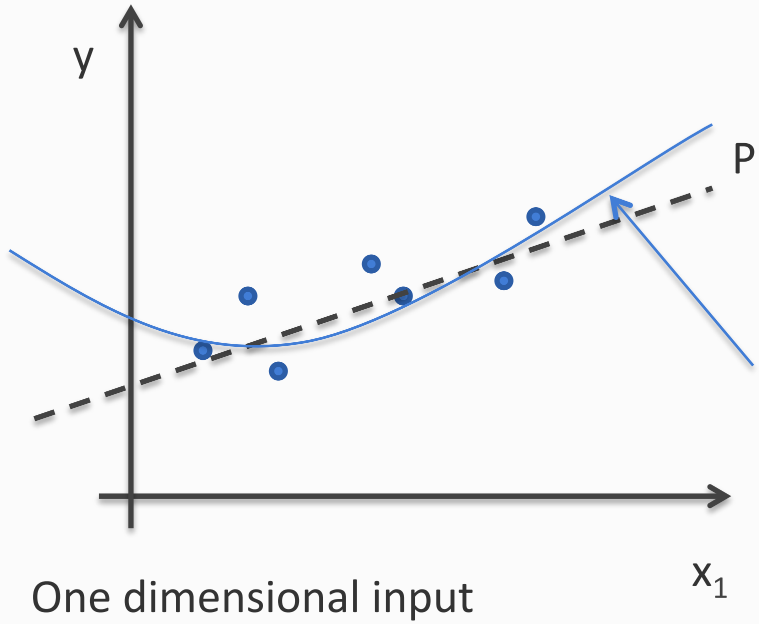
w is the learned weight vector in $\Re^d$

For simplicity, we will assume that x_1 is always 1.

That is $\mathbf{x} = [1\ x_2\ x_3\ \ldots\ x_d]^T$

This lets makes notation easier

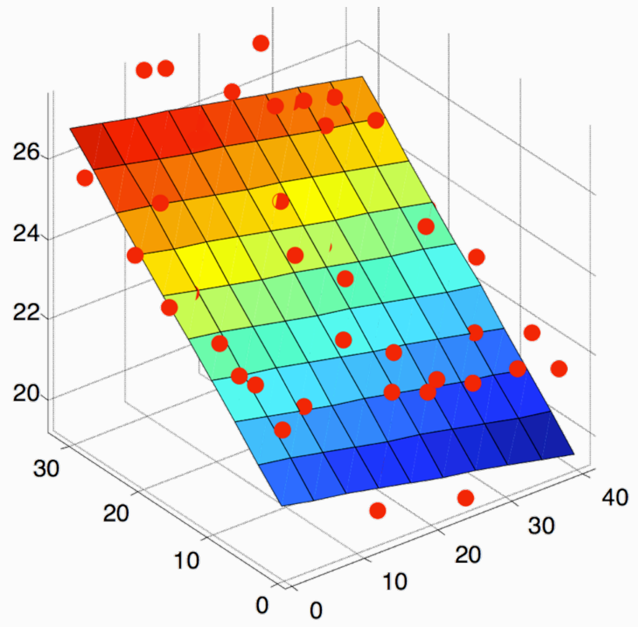
Examples



Predict using $y = w_1 + w_2 x_2$

The linear function is not our only choice. We could have tried to fit the data as another polynomial

One dimensional input



Two dimensional input

Predict using $y = w_1 + w_2 x_2 + w_3 x_3$

What is the best weight vector?

Question: How do we know which weight vector is the *best* one for a training set?

For an input ($\mathbf{x}_i$, y_i) in the training set, the ***cost*** of a mistake is

$$\left|y_i - \mathbf{w}^T\mathbf{x}_i\right|$$

Define the cost (or ***loss***) for a particular weight vector **w** to be

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

Sum of squared costs over the training set

One strategy for learning: *Find the **w** with least cost on this data*

Least Mean Squares (LMS) Regression

$$\min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{m} \left(y_i - \mathbf{w}^T \mathbf{x}_i\right)^2$$

Learning: minimizing mean squared error

Different strategies exist for *learning by optimization*

- Gradient descent is a popular algorithm
  - (For this minimization objective, there is also an analytical solution)

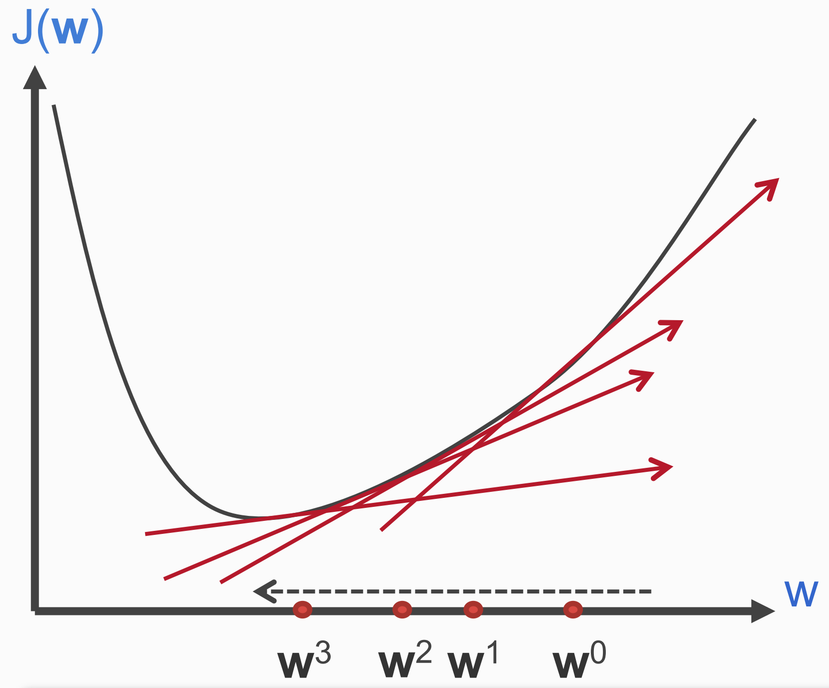
Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

General strategy for minimizing a function J(**w**)

- Start with an initial guess for **w**, say $\mathbf{w}^0$
- Iterate till convergence:
  - Compute the gradient of the gradient of J at $\mathbf{w}^t$
  - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a step in the opposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

1. Initialize $\mathbf{w}^0$
2. For t = 0, 1, 2, ….
   1. Compute gradient of J(**w**) at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$
   2. Update w as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

r: Called the learning rate
(For now, a small constant. We will get to this later)

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

1. Initialize $\mathbf{w}^0$
2. For t = 0, 1, 2, ….

   What is the gradient of J?

   1. $\boxed{\text{Compute gradient of } J(\mathbf{w}) \text{ at } \mathbf{w}^t\text{. Call it } \nabla J(\mathbf{w}^t)}$
   2. Update w as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

r: Called the learning rate
(For now, a small constant. We will get to this later)

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

Gradient of the cost

- The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

- Remember that **w** is a vector with d elements
  - **w** = $[w_1, w_2, w_3, \cdots w_j, \cdots, w_d]$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

Gradient of the cost

- The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

Gradient of the cost

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

- The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2\end{aligned}$$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

Gradient of the cost

- The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}\left(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots\right)
\end{aligned}$$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

Gradient of the cost

- The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}\left(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots\right) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij})
\end{aligned}$$

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

Gradient of the cost

- The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}\left(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots\right) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij}) \\
&= -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}
\end{aligned}$$

Gradient of the cost

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$$

- The gradient is of the form $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right]$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_{i=1}^{m}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial w_j}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial}{\partial w_j}(y_i - w_1x_{i1} - \cdots w_jx_{ij} - \cdots) \\
&= \frac{1}{2}\sum_{i=1}^{m}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij}) \\
&= \boxed{-\underbrace{\sum_{i=1}^{m}}_{\text{Sum of}}\underbrace{(y_i - \mathbf{w}^T\mathbf{x}_i)}_{\text{Error}}\times\underbrace{x_{ij}}_{\text{Input}}}
\end{aligned}$$

One element of the gradient vector

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$
2. For t = 0, 1, 2, …. *(until total error is below a threshold)*
   1. Compute gradient of J(**w**) at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

      Evaluate the function for ***each*** training example to compute the error and construct the gradient vector

      $$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

      One element of ∇ J

   2. Update **w** as follows: $\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$

      r: Called the learning rate

      (For now, a small constant. We will get to this later)

This algorithm is guaranteed to converge to the minimum of J if r is small enough. Why? The objective J is a ***convex*** function

Gradient descent for LMS

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

1. Initialize $\mathbf{w}^0$
2. For t = 0, 1, 2, …. *(until total error is below a threshold)*
    1. Compute gradient of J(**w**) at $\mathbf{w}^t$. Call it $\nabla J(\mathbf{w}^t)$

       Evaluate the function for ***each*** training example to compute the error and construct the gradient vector

       $$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{m}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$
    2. Update w as follows: $\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$

The weight vector is not updated until ***all*** *errors are calculated*

Why not make early updates to the weight vector as soon as we encounter errors instead of waiting for a full pass over the data?

Incremental/Stochastic gradient descent

- Repeat for each example ($\mathbf{x}_i$, y_i)
  - Pretend that the entire training set is represented by this single example
  - Use this example to calculate the gradient and update the model

- Contrast with *batch gradient descent* which makes one update to the weight vector for every pass over the data

Incremental/Stochastic gradient descent

1. Initialize $\mathbf{w}$
2. For t = 0, 1, 2, …. (until error below some threshold)
   - For each training example ($\mathbf{x}_i$, y_i):
     - Update **w**. For each element of the weight vector (w_j):

$$w_j^{t+1} = w_j^t + r(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

Contrast with the previous method, where the weights are updated only after all examples are processed once

Incremental/Stochastic gradient descent

1. Initialize $\mathbf{w}$
2. For t = 0, 1, 2, …. (until error below some threshold)
   – For each training example $(\mathbf{x}_i, y_i)$:
     - Update **w**. For each element of the weight vector (w_j):

$$w_j^{t+1} = w_j^t + r(y_i - \mathbf{w}^T \mathbf{x}_i) x_{ij}$$

This update rule is also called the Widrow-Hoff rule in the neural networks literature

Online/Incremental algorithms are often preferred when the training set is very large

May get close to optimum much faster than the batch version

Learning Rates and Convergence

- In the general (non-separable) case the learning rate *r* must decrease to zero to guarantee convergence

- The learning rate is called the *step size.*
  - More sophisticated algorithms choose the step size automatically and converge faster

- Choosing a better starting point can also have impact

- Gradient descent and its stochastic version are very simple algorithms
  - Yet, almost all the algorithms we will learn in the class can be traced back to gradient decent algorithms for different loss functions and different hypotheses spaces

Linear regression: Summary

- What we want: Predict a real valued output using a feature representation of the input
- Assumption: Output is a linear function of the inputs
- Learning by minimizing total cost
  - Gradient descent and stochastic gradient descent to find the *best* weight vector
  - This particular optimization can be computed directly by framing the problem as a matrix problem

Exercises

1. Use the gradient descent algorithms to solve the mileage problem (on paper, or write a small program)

2. LMS regression can be solved analytically. Given a dataset D = { (x_1, y_1), (x_2, y_2), $\cdots$, (x_m, y_m)}, define matrix X and vector Y as follows:

$$X = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots \mathbf{x}_m \end{bmatrix}_{d \times m} \qquad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_{m \times 1}$$

Show that the optimization problem we saw earlier is equivalent to

$$\min_{\mathbf{w}} \left(X^T\mathbf{w} - Y\right)^T \left(X^T\mathbf{w} - Y\right)$$

This can be solved analytically. Show that the solution w* is

$$\mathbf{w}^* = \left(XX^T\right)^{-1} XY$$

Hint: You have to take the derivative of the objective with respect to the vector **w** and set it to zero.

Where are we?

- Linear classifiers
- Expressivity of linear classifiers
- Least Squares Method
- Learning linear classifiers: The lay of the land

Regression vs. Classification

- Linear regression is about predicting real valued outputs

- Linear classification is about predicting a discrete class label
  - `+1` or `-1`
  - `SPAM` or `NOT-SPAM`
  - Or more than two categories

Coming up (next several weeks): Linear *classification*

- **Perceptron** and **Winnow**: Similar to the "incremental" method – Error driven learning, updates the hypothesis if there is an error
- **Support Vector Machines**: Define a different cost function that includes an error term *and* a term that targets future performance
- **Naïve Bayes classifier**: A simple linear classifier with a probabilistic interpretation
- **Logistic regression**: Another probabilistic linear classifier, bears similarity to linear regression and support vector machines

In all cases, the prediction will be done with the same rule:

$\mathbf{w}^T\mathbf{x} + b \geq 0 \Rightarrow$ Predict y = 1

$\mathbf{w}^T\mathbf{x} + b < 0 \Rightarrow$ Predict y = -1