Linear Regression

From Last Time

- Perceptron Algorithm
  - linear classifier (why?)

$h(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} \cdot \mathbf{x} + b)$

Which outputs +1 or –1.

Say:
+1 corresponds to blue, and
-1 to red, or vice versa.

Perceptron Learning Algorithm

Start with random w's $\quad h(\mathbf{x}, \mathbf{w}) = \text{sign}(\mathbf{w}^T\mathbf{x})$

Training tuples
$\mathbf{x}^1, y^1$
$\mathbf{x}^2, y^2$
...
$\mathbf{x}^n, y^n$

For each misclassified training tuple,
(i.e. $\text{sign}(\mathbf{w}^T\mathbf{x}^k) \neq y^k$)

Update **w**

$\mathbf{w} = \mathbf{w} + \eta \cdot y^k \mathbf{x}^k$

Well, why is this a good rule?

It can be shown that if the data is linearly separable, and we repeat this procedure many times, we will get a line that separates the training tuples.

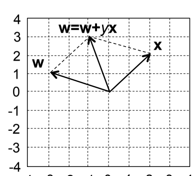
Recall the dot product

$$A \cdot B = \sum_i A_i B_i$$
$$= ||A|| * ||B|| * cos(\theta)$$

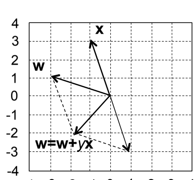
- ||A|| and ||B|| can only be positive
- cos(θ) is negative when the angle is between ½ pi and 3/2 pi
  - that is, when the angle is obtuse

Sign of dot product and **misclassification**

y=+1
w.x<0
y.**w.x**<0



y=-1
w.x>0
y.**w.x**<0

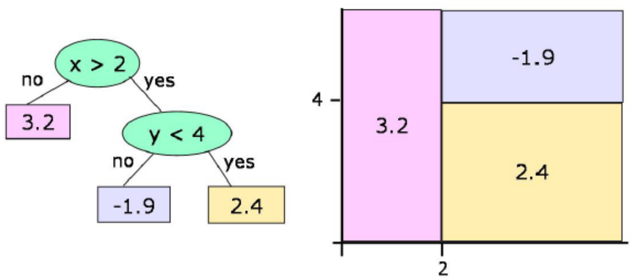


Now onto Regression

- Regression is predicting __________, whereas classification is predicting ____________.

Regression Trees

- Like decision trees, but with real-valued outputs at the leaves.



Another kind of prediction

YearsOfExperience (x_1)	Salary (y)
1	50
2	55
2	45
2.5	60
2.7	58
…	…

- Suppose we'd like to predict salary based on number of years of experience.
- Different prediction problem because class is a **continuous attribute**.
  - Called **Regression**
- Linear Regression: Build a prediction **line**.

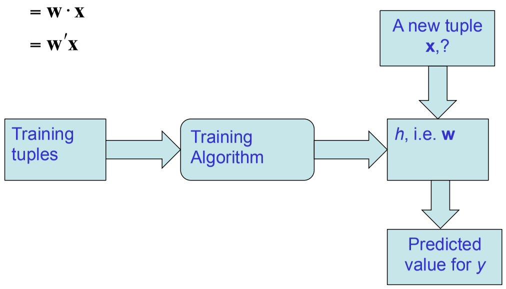
Linear regression with one variable

$h(\mathbf{w},\mathbf{x}) = w_0 + w_1 x_1$

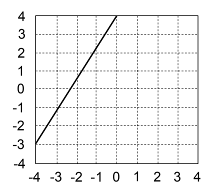
$= w_0 x_0 + w_1 x_1 \qquad x_0 = 1$

$= \mathbf{w} \cdot \mathbf{x}$

$= \mathbf{w}'\mathbf{x}$



Example of line



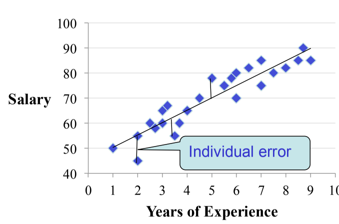
$w_0 + w_1 x_1 = y$

$w_0 + w_1(-4) = -3$

$w_0 + w_1(0) = 4$

$$\boxed{\begin{aligned} w_0 &= 4 \\ w_1 &= 7/4 \end{aligned}}$$

Cost/Error/Penalty Function



- Goal: find a line (hypothesis) $h(\mathbf{w},\mathbf{x})$ that for the training tuples gives numbers close to their y's.

n is the number of training tuples

We want to minimize E over possible $\mathbf{w}$'s. x^k are fixed, they are the training tuples.

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

Average squared error. ½ in the front is just to make the math easier.

Recap

Hypothesis form:

$h(\mathbf{w},\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = w_0 + w_1 x_1$

Weights to learn:

w_0, w_1

Error function:

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

Minimization:

$$\min_{\mathbf{w}} E(\mathbf{w}) \quad \text{i.e.} \quad \min_{w_0, w_1} E(w_0, w_1) \quad \text{i.e.} \quad E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2$$

Simplified h (for illustration)

Simplified hypothesis form (w_0=0), i.e. lines passing through the origin:

$$h(\mathbf{w},\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} = w_1 x_1$$

Weights to learn:

w_1

Error function:

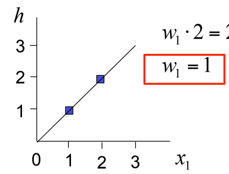
$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

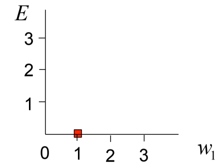
Minimization:

$\min_{\mathbf{w}} E(\mathbf{w})$ i.e. $\min_{w_1} E(w_1)$ i.e. $E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - w_1 x_1^k)^2$

h vs. E

For a fixed w_1, h is a function of x_1. For a fixed $x^k{}_1$'s, E is a function of w_1.

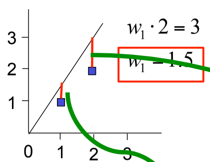


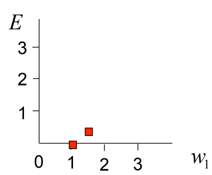


$$E(\mathbf{w}) = \frac{1}{2\cdot 2}\left[(1-1\cdot 1)^2 + (2-1\cdot 2)^2\right] = 0$$

h vs. E

For a fixed w_1, h is a function of x_1. For a fixed $x^k{}_1$'s, E is a function of w_1.

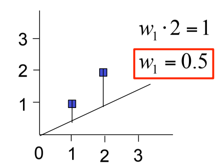


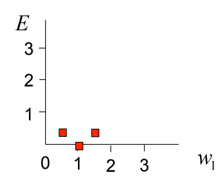


$$E(\mathbf{w}) = \frac{1}{2\cdot 2}\left[(1-1.5\cdot 1)^2 + (2-1.5\cdot 2)^2\right] = 0.3125$$

h vs. E

For a fixed w_1, h is a function of x_1. For a fixed $x^k{}_1$'s, E is a function of w_1.





$$E(\mathbf{w}) = \frac{1}{2\cdot 2}\left[(1-0.5\cdot 1)^2 + (2-0.5\cdot 2)^2\right] = 0.3125$$

$\min_{w_1} E(w_1)$? $w_1 = 1$

Recap again for w_0!=0

Hypothesis form:

$$h(\mathbf{w},\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} = w_0 + w_1 x_1$$

Weights to learn:

w_0, w_1

Error function:

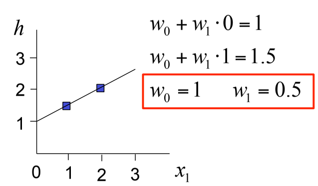
$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

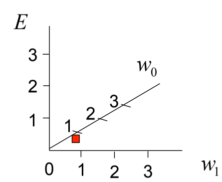
Minimization:

$\min_{\mathbf{w}} E(\mathbf{w})$ i.e. $\min_{w_0,w_1} E(w_0,w_1)$ i.e. $E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2$

h vs. E

For a fixed w_0,w_1, h is a function of x_1. For a fixed $x^k{}_1$'s, E is a function of w_1,w_0.

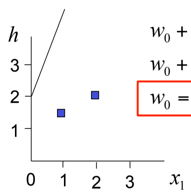




$$E(1,0.5) = \frac{1}{2\cdot 2}\left[(1.5-1-0.5\cdot 1)^2 + (2-1-0.5\cdot 2)^2\right] = 0$$

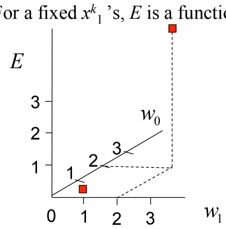
h vs. E

For a fixed w_0, w_1, h is a function of x_1.



$$w_0 + w_1 \cdot 0 = 2$$
$$w_0 + w_1 \cdot 0.5 = 3$$
$$\boxed{w_0 = 2 \quad w_1 = 2}$$

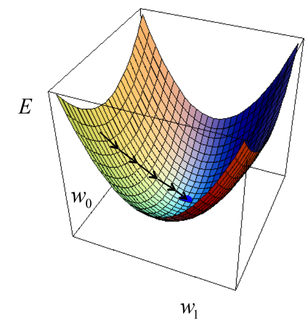
For a fixed x^k_1 's, E is a function of w_1.



$$E(2,2) = \frac{1}{2 \cdot 2}\left[(1.5 - 2 - 2 \cdot 1)^2 + (2 - 2 - 2 \cdot 2)^2\right] = 5.56$$

Minimization

- Start with some w_0, w_1,
- Nudge w_0, w_1 to lower E



Which direction to nudge?

- Use opposite of gradient direction.

$$w_0 \leftarrow w_0 - \kappa \frac{\partial}{\partial w_0} E(w_0, w_1)$$
$$= w_0 - \kappa \frac{\partial}{\partial w_0}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2\right)$$
$$= w_0 - \kappa \frac{1}{2n}\sum_{k=1}^{n}\frac{\partial}{\partial w_0}(y^k - w_0 - w_1 x_1^k)^2$$
$$= w_0 + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)$$

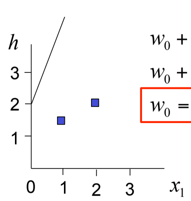
Which direction to nudge?

- Use opposite of gradient direction.

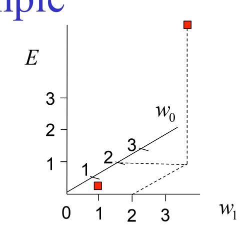
$$w_1 \leftarrow w_1 - \kappa \frac{\partial}{\partial w_1} E(w_0, w_1)$$
$$= w_1 - \kappa \frac{\partial}{\partial w_1}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2\right)$$
$$= w_1 - \kappa \frac{1}{2n}\sum_{k=1}^{n}\frac{\partial}{\partial w_1}(y^k - w_0 - w_1 x_1^k)^2$$
$$= w_1 + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)x_1^k$$

Learning rate kappa
If kappa too small, GD is slow.
If kappa too big, GD is too eager and can overshoot min.

Example



$$w_0 + w_1 \cdot 0 = 2$$
$$w_0 + w_1 \cdot 0.5 = 3$$
$$\boxed{w_0 = 2 \quad w_1 = 2}$$



$$E(2,2) = \frac{1}{2 \cdot 2}\left[(1.5 - 2 - 2 \cdot 1)^2 + (2 - 2 - 2 \cdot 2)^2\right] = 5.56$$

$$w_0 \leftarrow w_0 + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)$$
$$= 2 + 0.1\frac{1}{2}\big((1.5 - 2 - 2 \cdot 1) + (2 - 2 - 2 \cdot 2)\big)$$
$$= 1.675$$

$$w_1 \leftarrow w_1 + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)x_1$$
$$= 2 + 0.1\frac{1}{2}\big((1.5 - 2 - 2 \cdot 1) \cdot 1 + (2 - 2 - 2 \cdot 2) \cdot 2\big)$$
$$= 1.475$$

$$E(1.675, 1.475) = \frac{1}{2 \cdot 2}\left[(1.5 - 1.675 - 1.475 \cdot 1)^2 + (2 - 1.675 - 1.475 \cdot 2)^2\right] = 2.4$$

Vectorization

$$\frac{\partial}{\partial w_0} E(w_0, w_1)$$
$$= \frac{\partial}{\partial w_0}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2\right)$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 x_0^k - w_1 x_1^k)x_0^k$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_0^k$$

$$\frac{\partial}{\partial w_1} E(w_0, w_1)$$
$$= \frac{\partial}{\partial w_1}\left(\frac{1}{2n}\sum_{k=1}^{n}(y^k - w_0 - w_1 x_1^k)^2\right)$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - w_0 x_0^k - w_1 x_1^k)x_1^k$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_1^k$$

$$\nabla_E(\mathbf{w}) = \nabla_E(w_0, w_1) = \begin{bmatrix} \frac{\partial}{\partial w_0} E(w_0, w_1) \\ \frac{\partial}{\partial w_1} E(w_0, w_1) \end{bmatrix}$$
$$= \begin{bmatrix} \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_0^k \\ \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)x_1^k \end{bmatrix}$$
$$= \frac{1}{n}\sum_{k=1}^{n}\begin{bmatrix} (y^k - \mathbf{w}'\mathbf{x}^k)x_0^k \\ (y^k - \mathbf{w}'\mathbf{x}^k)x_1^k \end{bmatrix}$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\begin{bmatrix} x_0^k \\ x_1^k \end{bmatrix}$$
$$= \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

Gradient Recap

$$\nabla_E(\mathbf{w}) = \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

$$\mathbf{w} \leftarrow \mathbf{w} + \kappa \nabla_E(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

Matlab/Octave

$$E\left(\begin{bmatrix}2\\2\end{bmatrix}\right) = \frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2 + \left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right) = 5.56$$

```
E=(1/(2*2))*( (1.5-[2 2]*[1; 1])^2 + (2-[2 2]*[1; 2])^2 )
```

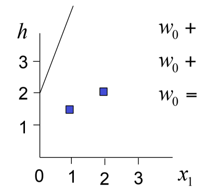
$$\mathbf{w} \leftarrow \begin{bmatrix}2\\2\end{bmatrix} + 0.1\frac{1}{2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)\begin{bmatrix}1\\1\end{bmatrix} + \left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}1.675\\1.475\end{bmatrix}$$

```
w=[2;2]+0.1*(1/2)*
( (1.5-[2 2]*[1;1])*[1;1] + (2-[2 2]*[1;2])*[1;2] )
```

$$E\left(\begin{bmatrix}1.675\\1.475\end{bmatrix}\right) = \frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2 + \left(2-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right) = 2.4$$

```
E=(1/(2*2))*
( (1.5-[1.675 1.475]*[1; 1])^2 + (2-[1.675 1.475]*[1; 2])^2 )
```

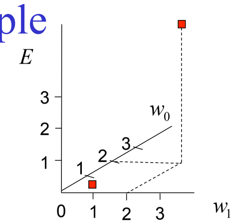

Example



$w_0 + w_1 \cdot 0 = 2$

$w_0 + w_1 \cdot 0.5 = 3$

$w_0 = 2 \quad w_1 = 2$



$$E\left(\begin{bmatrix}2\\2\end{bmatrix}\right) = \frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2 + \left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right) = 5.56$$

$$\mathbf{w} \leftarrow \mathbf{w} + \kappa \frac{1}{n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)\mathbf{x}^k$$

$$= \begin{bmatrix}2\\2\end{bmatrix} + 0.1\frac{1}{2}\left(\left(1.5-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)\begin{bmatrix}1\\1\end{bmatrix} + \left(2-\begin{bmatrix}2 & 2\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)\begin{bmatrix}1\\2\end{bmatrix}\right) = \begin{bmatrix}1.675\\1.475\end{bmatrix}$$

$$E\left(\begin{bmatrix}1.675\\1.475\end{bmatrix}\right) = \frac{1}{2\cdot 2}\left(\left(1.5-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\right)^2 + \left(2-\begin{bmatrix}1.675 & 1.475\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\right)^2\right) = 2.4$$

More than one x attribute

GPA	YearsOfExperience	Salary
90	1	50
80	3	60
90	2	55
70	8	70
…	…	…

y

Linear Approximation

$$y \approx w_1x_1 + \ldots + w_mx_m + b$$

Approximate y given the attribute values by a linear function of the attributes.

w_0 is b

For neatness, let $\mathbf{w}' = [w_0, w_1, .., w_m]$ $\quad \mathbf{x}' = [1, x_1, .., x_m]$

Then we can write the above in a neat form as

1 is an artificial, but completely harmless constant attribute we add to each training instance.

$$y \approx \mathbf{w}'\mathbf{x}$$

How to estimate the w parameters, i.e. $\mathbf{w}$?

Cost function

- Find **w** that gives the lowest approximation error given the training data.
  - Minimize the **sum of square errors**:

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

Same **w** for all the training instances.

Iterative Method

- Start at some $\mathbf{w}_0$; take a step along **steepest slope**.
  - What's the steepest slope?

- Gradient of E:

$$\nabla_E(\mathbf{w}) = -\frac{1}{n}\sum_{k=1}^{n}\left(y^k - \mathbf{w}'\mathbf{x}^k\right)\mathbf{x}^k$$

Same form as before

31

Gradient Descent Algorithm

Initialize at some $\mathbf{w}_0$

For t=0,1,2,…do

Compute the gradient $\nabla_E(\mathbf{w}_t) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$

Update the weights $\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa\nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$

Iterate with the next step until **w** doesn't change too much (or for a fixed number of iterations)

Return final **w**.

32

GD Matlab/Octave

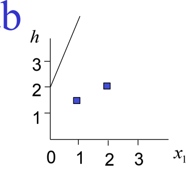
$$\mathbf{w} \leftarrow \mathbf{w} + \kappa\frac{1}{n}\sum_{k=1}^{n}\left(y^k - \mathbf{w}'\mathbf{x}^k\right)\mathbf{x}^k$$

Matlab/Octave:

```
w = w + kappa*(1/n)*(X'*(y-X*w));
```

33

GD example in Matlab




```
 % Training data
X=[1 1;
   1 2];
% These are the values we want to predict
y=[1.5; 2];
% This is the starting assignment for the weight vector.
w=[2; 2];

% Learning rate.  Play with it to see how it changes the outcome!
kappa = 0.1;
n = length(y);

fprintf('Before optimization, loss is %.4f\n',mean(1/2*(y-X*w).^2));
for t=1:20,
    w = w + kappa*(1/n)*(X'*(y-X*w));
    if rem(t,10) ==1,
        fprintf('Iteration %i, loss is %.4f\n',t,mean(1/2*(y-X*w).^2));
    end
end;
fprintf('After optimization, loss is %.4f\n',mean(1/2*(y-X*w).^2));
w
```

34