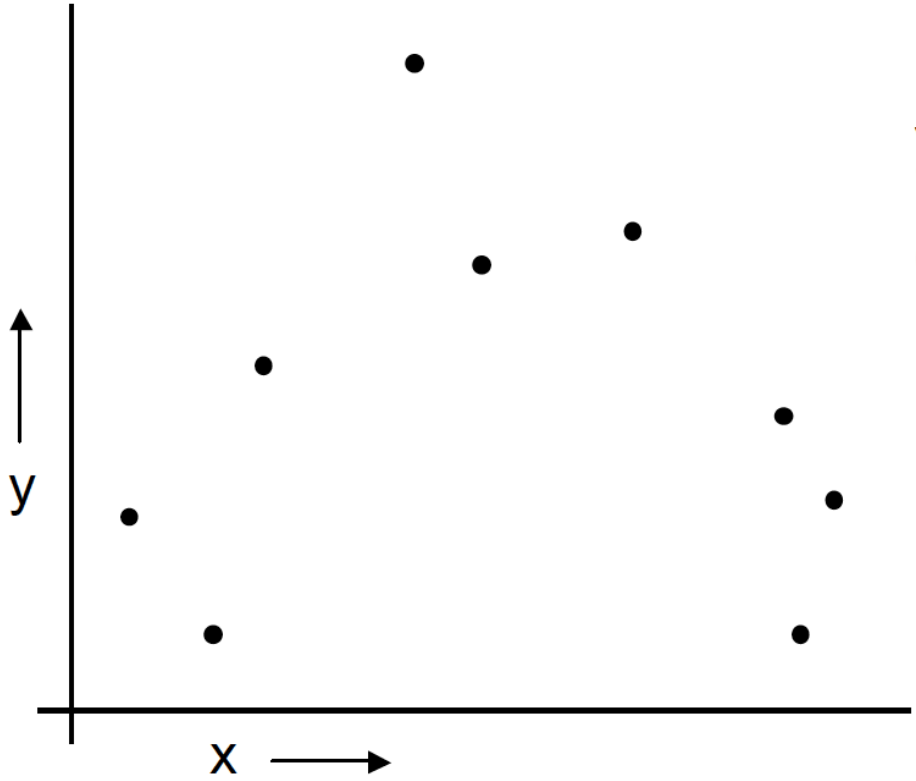
CS 440
Introduction to Artificial Intelligence

Lecture 23:

Cross-Validation – Linear Regression (continued)

May 14, 2020

A Regression Problem

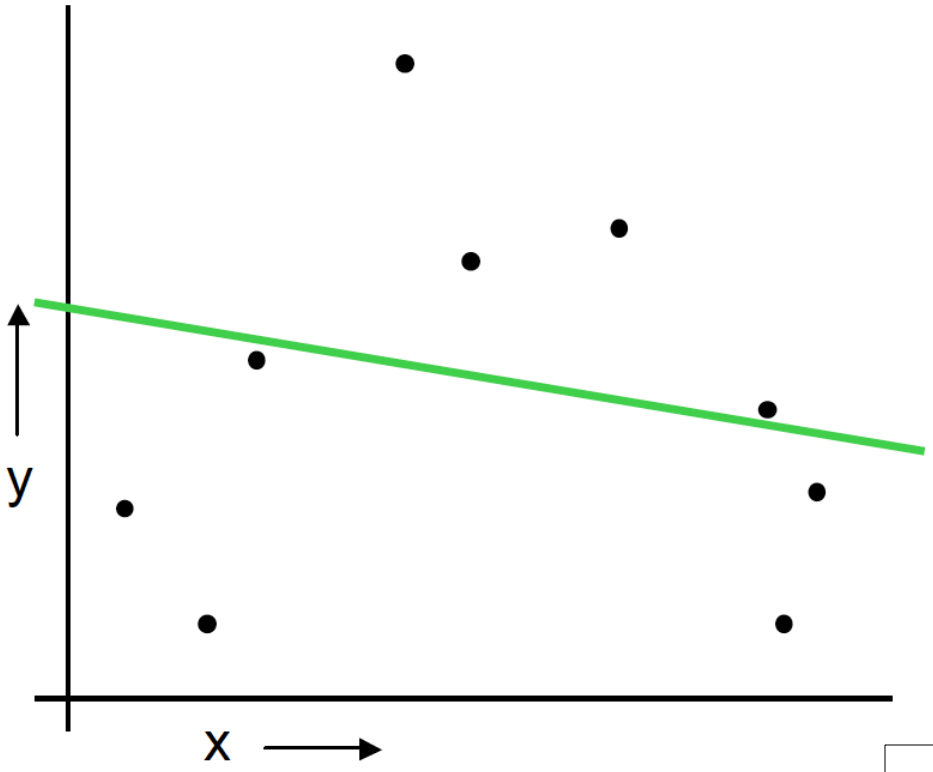


y = f(x) + noise

Can we learn f from this data?

Let's consider three methods…

Linear Regression



Objective: Minimize the

Sum of Squared Errors

i.e. sum of squared differences

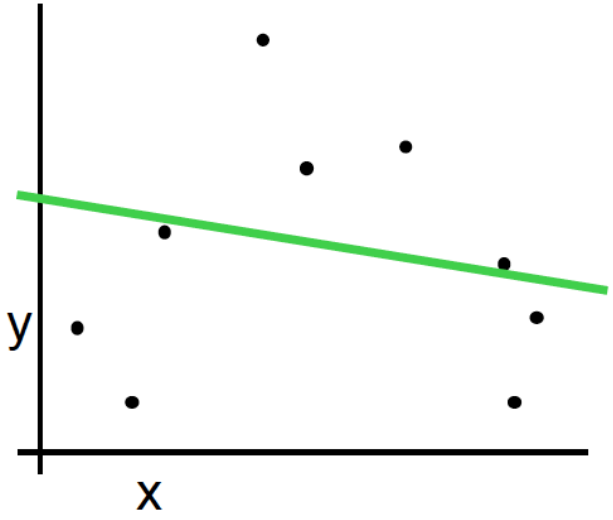
between y values

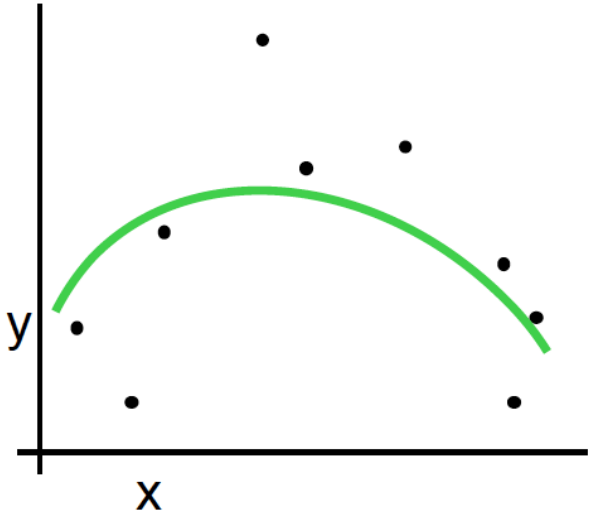
and the green line

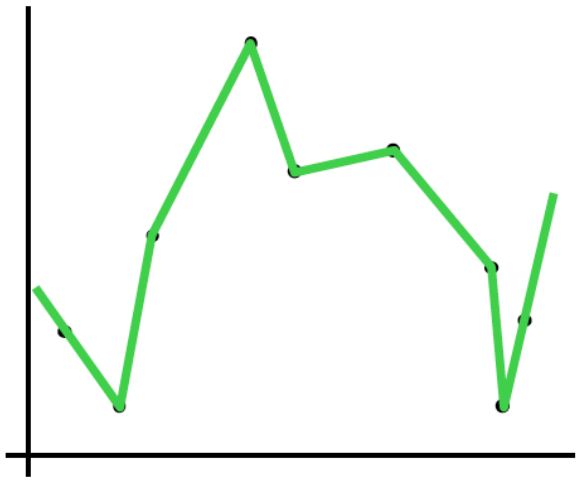
$y = w_0 + w_1 \cdot x$

- $\hat{y}i$ is the prediction of the linear model
- yi the actual value for input xi
- Then minimize: $Q = \Sigma i\ (\hat{y}i - yi)2$

Which one is the best?



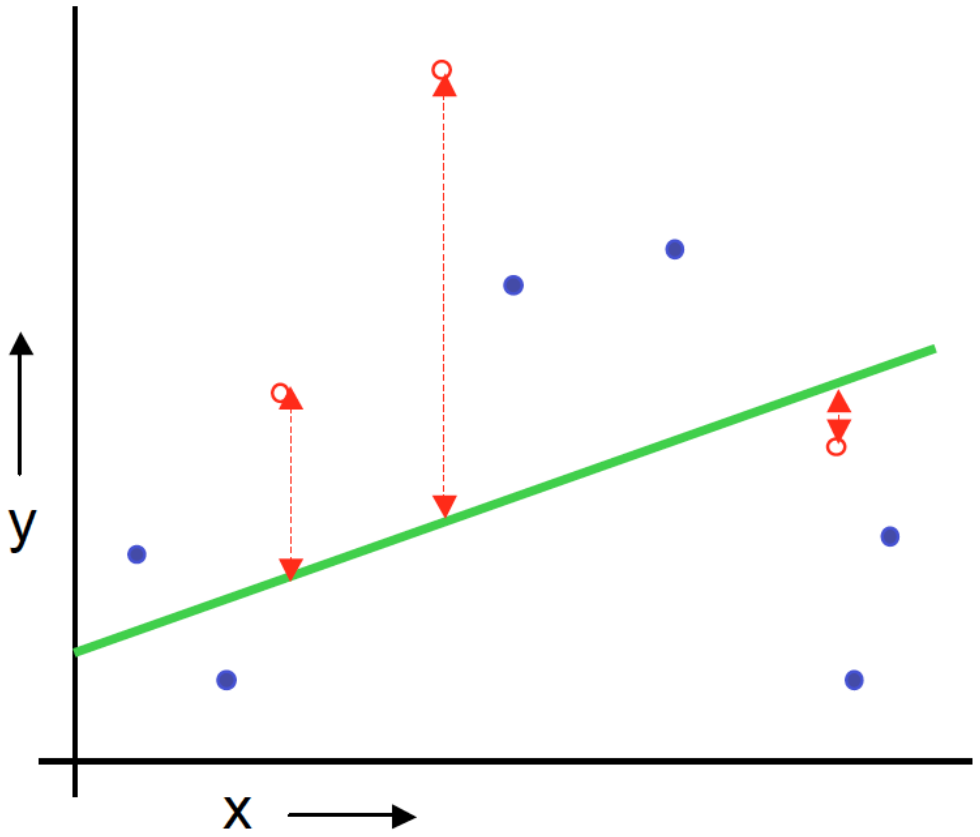




Why not choose the method with the best fit to the data?

"How well are you going to predict future data drawn from the same distribution?"

Test Set



(Linear regression example)

Mean Squared Error = 2.4

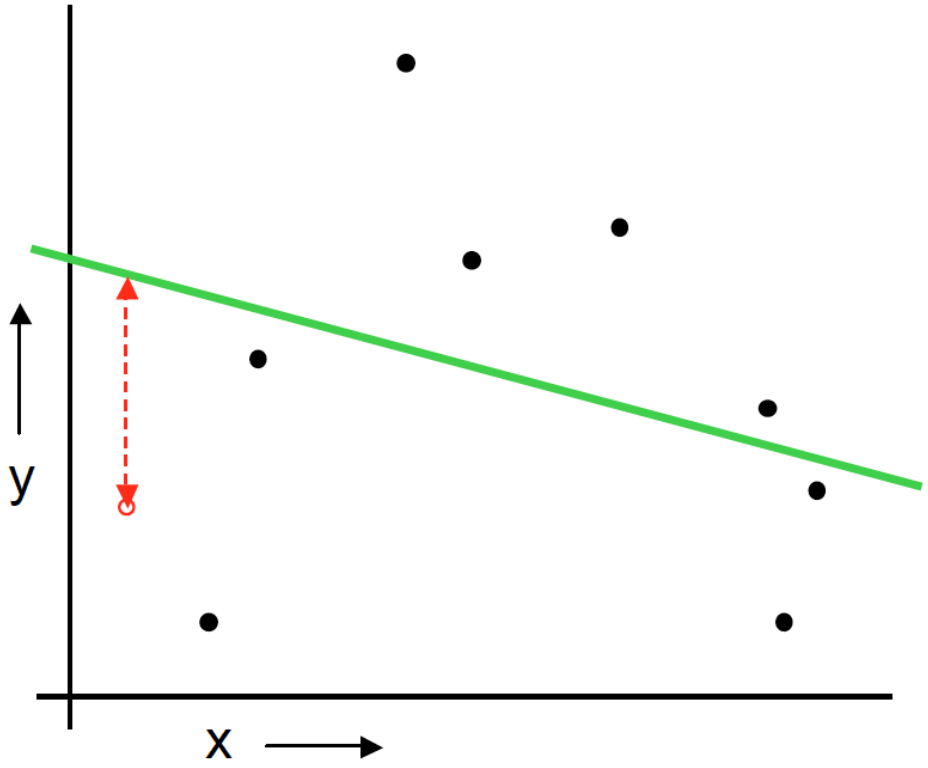
1. Randomly choose 30% of the data to be in a test set

2. The remainder is a training set

3. Perform your regression on the training set

4. Estimate your future performance with the test set

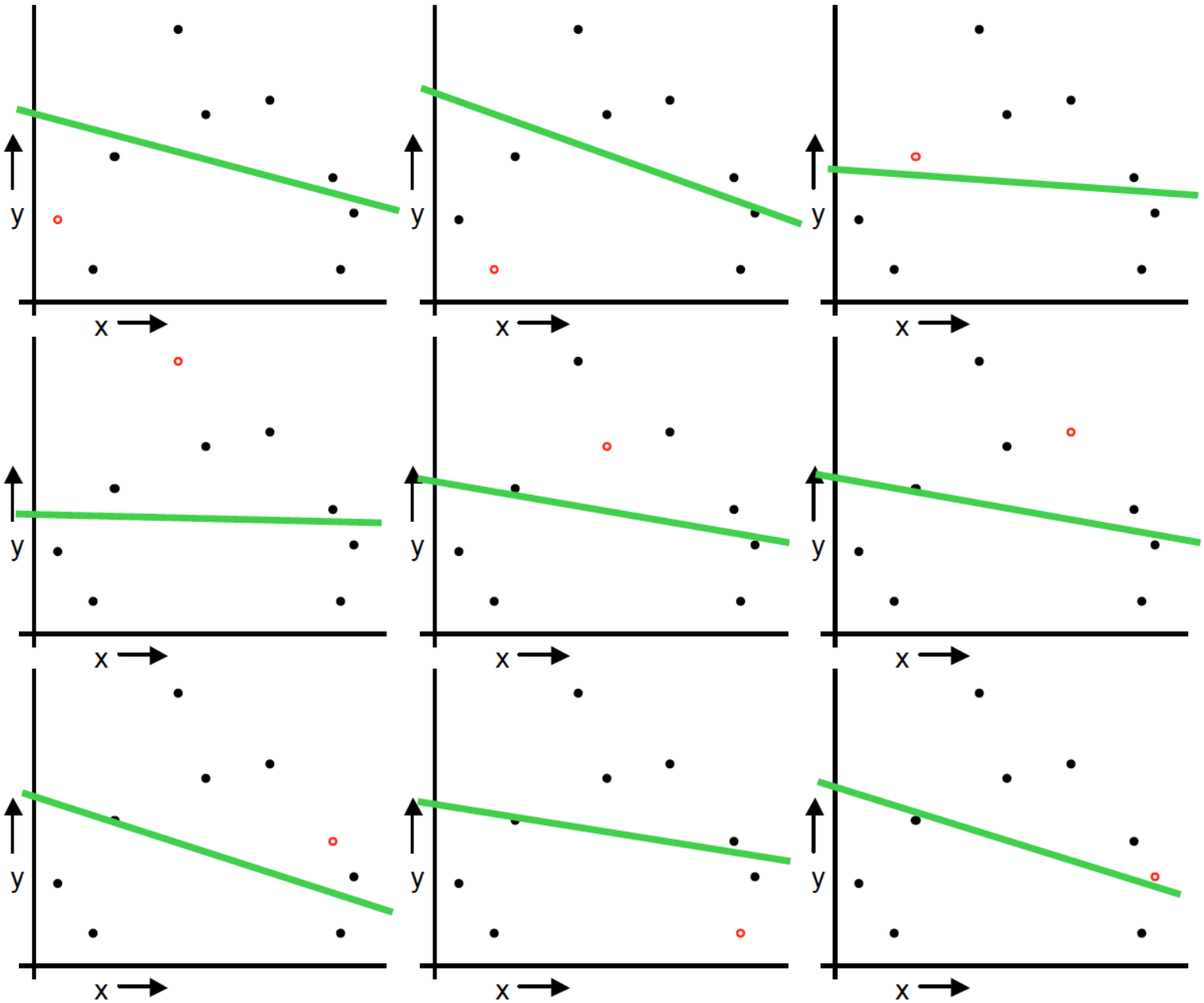
Leave One Out Cross Validation



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

Leave One Out Cross Validation



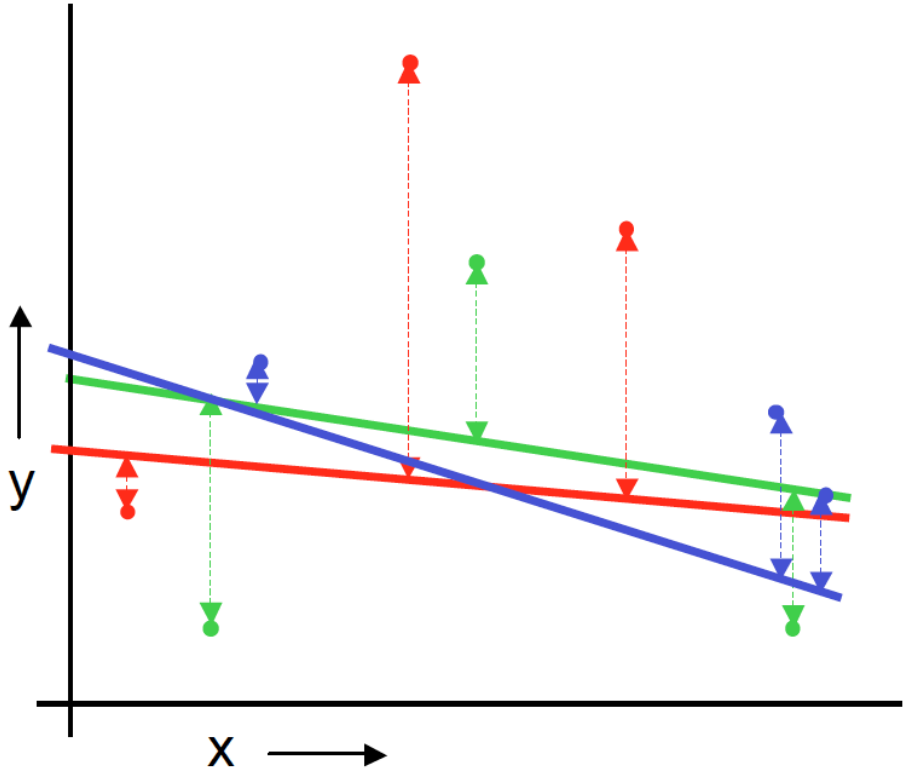
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 2.12$$

k-fold Cross Validation



Linear Regression

$MSE_{3FOLD}=2.05$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

Linear Regression

- It is traditional (going back to Gauss) to use the squared loss function L_2 summed over all the training examples

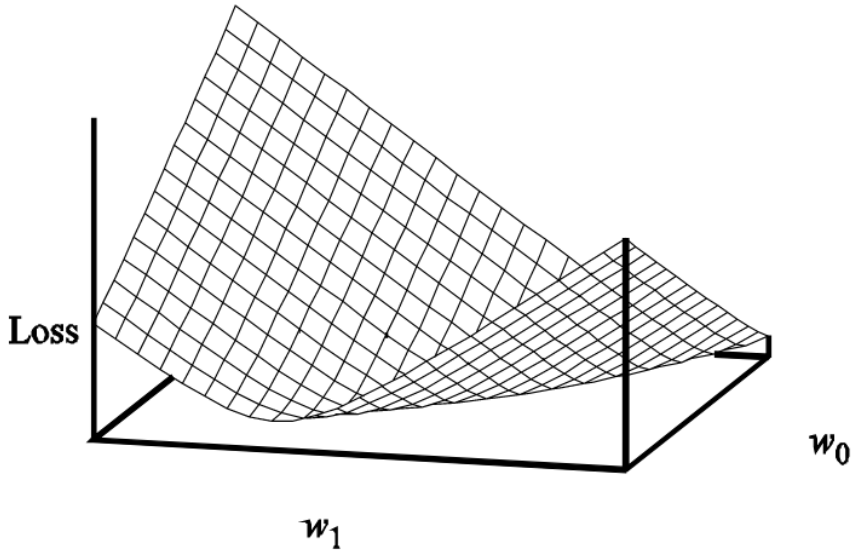
$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

- We would like to find $\mathbf{w^*} = \arg\min_w Loss(h_w)$
- The sum $\sum_j (y_j - (w_1 x + w_0))^2$ is minimized when its partial derivatives with respect to w_0 and w_1 are zero

$$\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0$$

Linear Regression



- These equations have a unique solution

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}$$

$$w_0 = \left(\sum y_j - w_1(\sum x_j)\right) / N$$

- The weight space defined by w_0 and w_1 is convex
- This is true for every linear regression problem with an L_2 loss function, and it implies that there are no local minima

Discussion

How would you minimize lost for other functions?

- Parabolas
- Third order polynomial
- ect.

Beyond Linear Regression

- To go beyond linear models, we will need to face the fact that the equation defining minimum loss will often have no closed-form solution
- Instead, we will face a general optimization search in continuous weight space
- As we already know, such problems can be addressed by a hill-climbing algorithm that follows the ***gradient*** of the function to be optimized
- Because we are trying to minimize the loss, we will use ***gradient descent***
- We choose any starting point in weight space and then move to a neighboring point that is downhill, repeating until we converge on the minimum possible loss

Learning Hypothesis Parameters

w ← any point in the parameter space

loop until convergence **do**

for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

- The step size parameter α is usually called the ***learning rate*** when we are trying to minimize loss in a learning problem
- It can be a fixed constant, or it can decay over time as the learning problem proceeds
- For univariate regression, the loss function is a quadratic function, so the partial derivative will be a linear function

One training example

- Let's consider the case of only one training example, (x, y):

$$\frac{\partial}{\partial w_i} Loss(\mathbf{w}) = \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2$$

$$= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))$$

$$= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0))$$

- Applying this to both w_0 and w_1 we get:

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

$$\frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

Many Training Examples

- Plugging these values back to the gradient descent update rule (folding the constant 2 into the learning rate α), we get the following learning rules for the weights

$$w_0 \leftarrow w_0 + \alpha(y - h_w(x))$$
$$w_1 \leftarrow w_1 + \alpha(y - h_w(x)) \cdot x$$

- Intuitively:
  - if $h_w(x) > y$ – the output of the hypothesis is too large – reduce w_0 a bit
  - Reduce w_1 if x was a positive input but increase w_1 if x was a negative input
- For N training examples the derivative of a sum is the sum of the derivatives, and we have

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j))$$
$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j)) \cdot x_j$$

Linear Classification Example