Do not distribute course material

Topic 1 continued

CS/EE-UY 4563: INTRODUCTION TO MACHINE LEARNING

PROF. LINDA SELLIE

Some of the slides are from Prof. Sundeep Rangan

Some approaches are taken from CMU 18-661 Introduction to Machine Learning

Learning Objectives

- How to load data from a text file
- How to visualize data via a scatter plot
- Describe a linear model for data
  - Identify the label (target variable) and feature (predictor)
- Understand the objective function for linear regression
- Understand how partial derivatives can be used in a convex optimization problem
- Optimization:
  - Compute optimal parameters for the model using a closed form solution
  - Compute the optimal parameters for the model using gradient descent
- Evaluation:
  - Able to compute R^2
  - Able to visually determine goodness of fit and identify different causes for poor fit

Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Global Optimizer: LS Fit Solution
- Local Optimizer: Gradient Descent
- Assessing Goodness of Fit
- Extra Slides: Global Optimizer for multivariate linear regression: Normal Equation

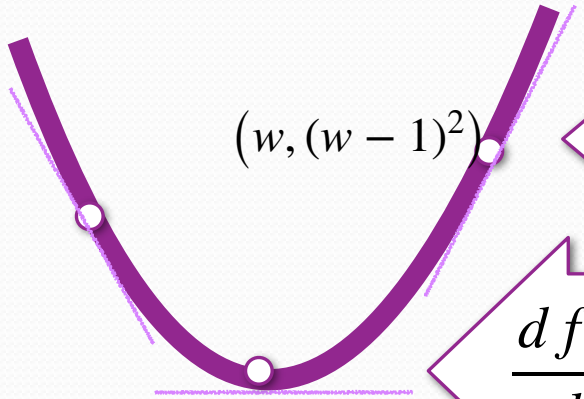
Finding Parameters via Optimization
A general ML recipe

General ML problem	Simple linear regression
☑ Get data	Data: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1,\ldots,N$
☑ Find a hypothesis class/model class with parameters	Linear model: $\hat{y} = w_0 + w_1\mathbf{x}$
☑ Pick a loss function ◦ Measures goodness of fit model to data ◦ Function of the parameters	Loss function: $RSS(w) = \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2$
☐ Find parameters that minimizes loss	Select w= $\begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ to minimize loss function

Before finding how to use a local optimization technique on RSS, lets develop some intuition on a simpler function

Calculus Review

$$f(w) = (w-1)^2$$



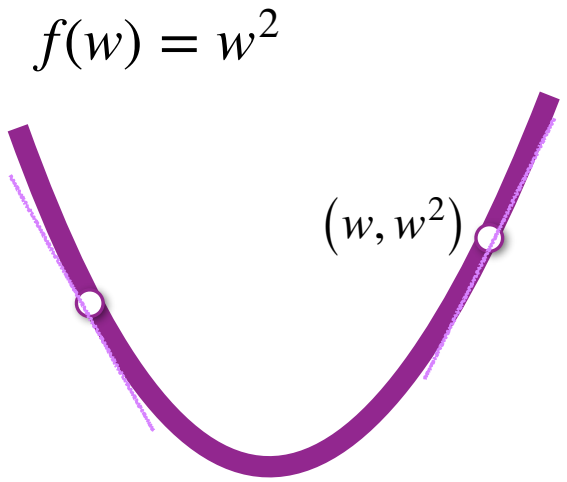
$\frac{df(w)}{dw} = 2(w-1)$ is the rate of change of f at the point w

$\frac{df(1)}{dw} = 2(1-1) = 0$, the function is at a minimal value

Global optimization: Find the minimum value of the function.

$$\frac{df(w)}{dw} = 2(w-1) = 0$$

Another approach to minimize a function



Lets use a simpler function: $f(w) = w^2$

For any w, which $w' = w + \epsilon$ would most likely cause $f(w') < f(w)$?

(You may assume $f(w) \neq 0$)

Notation alert!

w' is just another variable, not the derivative.

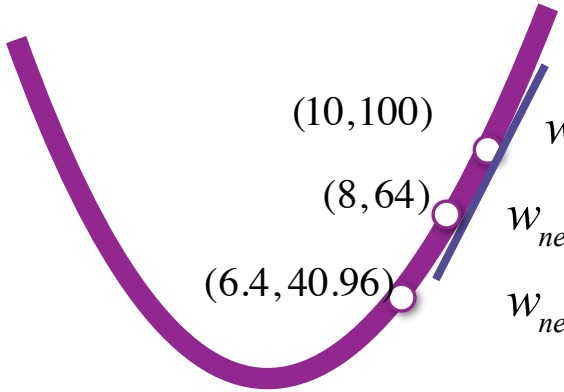
Local optimization

- Notice that the amount we move decreases as we move toward the minimum
  - If we are at w = 10 the amount we move is 2
  - If we are at 8 the amount we move is 1.6
- If we started at x = -10, the derivative is negative, so $w_{new} = (-10) - \alpha \cdot 2 \cdot (-10)$

$$= (-10) - (0.1) \cdot 2 \cdot (-10)$$
$$= (-10) + 2 = -8$$

The derivate gives the direction to move

f(w) = w² df/dw = 2w



$w_{new} = w - \alpha \cdot 2 \cdot w \qquad \alpha = 0.1$

$w_{new} = 10 - \alpha \cdot 2 \cdot 10 = 10 - (0.1) \cdot 2 \cdot 10 = 8$

$w_{new} = 8 - \alpha \cdot 2 \cdot 8 \quad = 8 - (0.1) \cdot 2 \cdot 8 = 6.4$

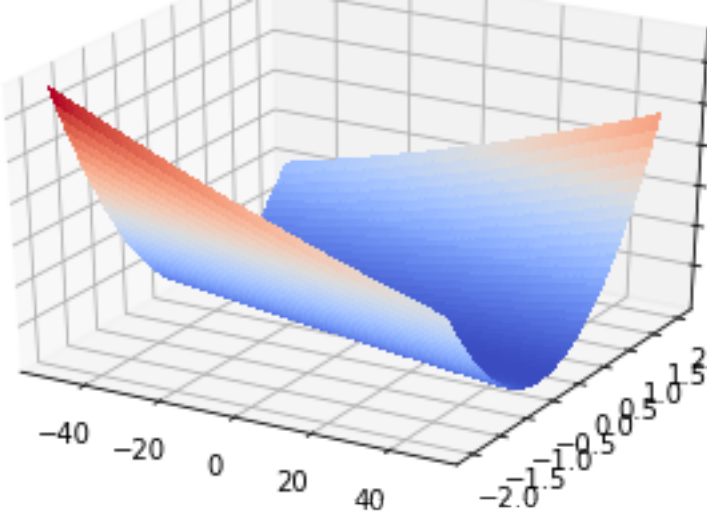
Algorithm:
For i = 1 to num_iters:

if $\frac{df(w)}{dw} > 0$ then f is increasing,
move w a little to the left

if $\frac{df(w)}{dw} < 0$ then f is decreasing,
move w a little to the right

A nice intuitive explanation is at: https://theory.stanford.edu/~tim/s16/l/l5.pdf

Local Optimization



How can we make a small improvement in our parameters?

WHAT IS THE PSEUDOCODE?

Open discussion - design an algorithm using the ideas we just discussed.

Objective function:

$$RSS(w_0, w_1) = \sum_{i=1}^{N} (y^{(i)} - (w_0 + w_1 \mathbf{x}))^2$$

New simplification: New Objective function

$$J(w_0, w_1) =$$

Partial derivatives

$$\frac{\partial J(w_0, w_1)}{\partial w_0} =$$

$$\frac{\partial J(w_0, w_1)}{\partial w_1} =$$

Cost function

- Minimizing $\text{RSS}(\mathbf{w}) = \sum_{i=1}^{N}(y^{(i)} - w_0 - w_1 x^{(i)})^2$ (our cost function for linear regression) is the same as minimizing:

$$J(w_0, w_1) = \frac{1}{2}\mathrm{E}_{\text{in}}(w_0, w_1) = \frac{1}{2N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})^2 = \frac{1}{2N}RSS$$

notice we switch the sign inside the bracket

- J, RSS and E_{in} are convex functions (as was x^2), so we can use the gradient to find the minimum by taking a sequence of steps. Here is the derivative for the E_{in} function with respect to the parameters:

$$\frac{\partial J(w_0, w_1)}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})$$

$$\frac{\partial \mathrm{J}(w_0, w_1)}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})x^{(i)}$$

By scanned by jd, photography by "frères Bisson" (Bibliothèque France (BNF)) [Public domain], via Wikimedia Commons

Gradient Descent Optimizati

$$\frac{\partial J(w_0,w_1)}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)}) \qquad \frac{\partial J(w_0,w_1)}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})x^{(i)}$$

- To decrease the cost, we update the parameters, w_0 w_1, using the update rule:

for $i = 1$ to num_iter

$$temp0 = w_0 - \alpha\frac{\partial J(w_0,w_1)}{\partial w_0} = w_0 - \frac{\alpha}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})$$

$$temp1 = w_1 - \alpha\frac{\partial J(w_0,w_1)}{\partial w_1} = w_1 - \frac{\alpha}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})x^{(i)}$$

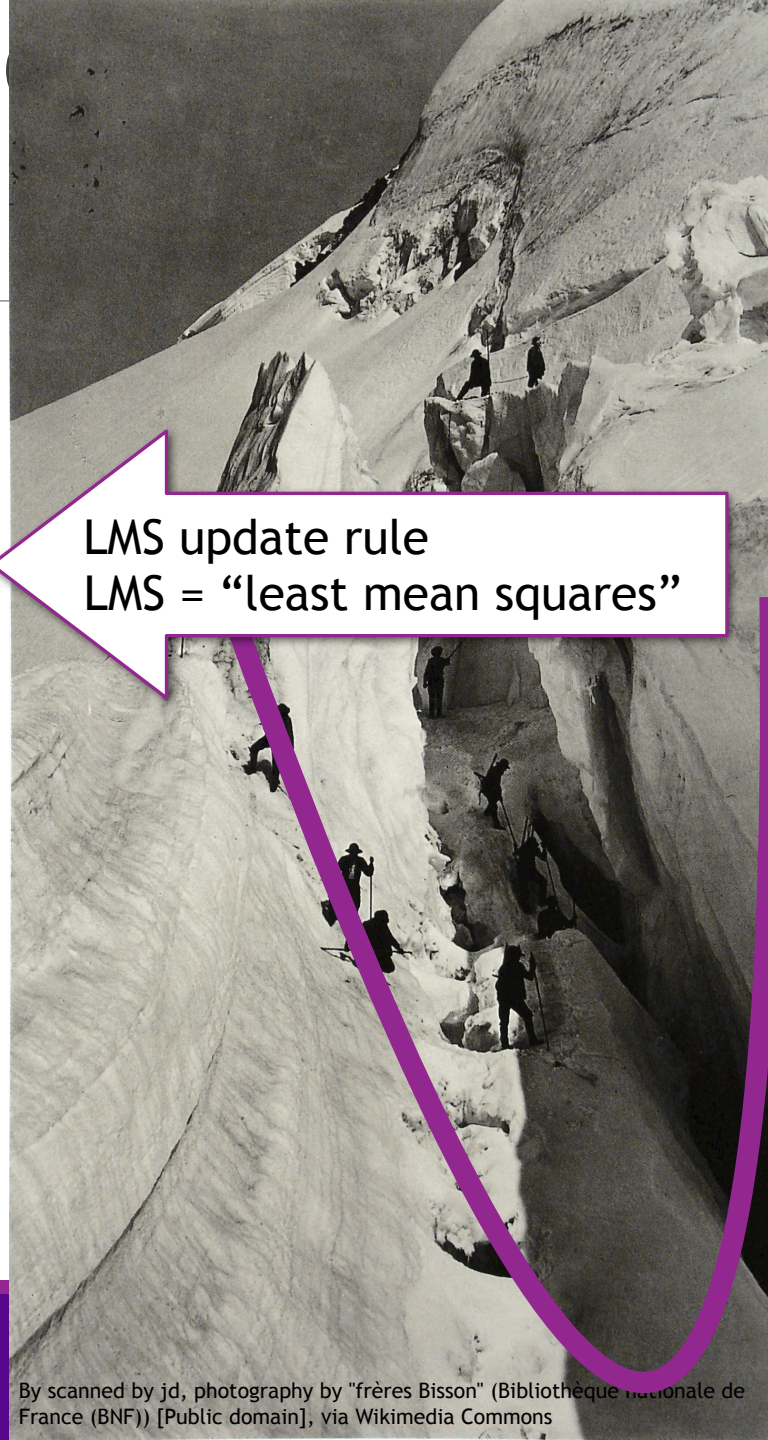
$$w_0 = temp0$$

$$w_1 = temp1$$

- While not converged (or close enough)

  take a step toward the bottom by changing

  (thus we have reduced our error!)



By scanned by jd, photography by "frères Bisson" (Bibliothèque nationale de France (BNF)) [Public domain], via Wikimedia Commons

Gradient Descent Optimizati

$$\frac{\partial J(w_0,w_1)}{\partial w_0} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)}) \qquad \frac{\partial J(w_0,w_1)}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})x^{(i)}$$

- To decrease the cost, we update the parameters, w_0, w_1, using the update rule:

for $i = 1$ to num_iter

$$temp0 = w_0 - \alpha \frac{\partial J(w_0,w_1)}{\partial w_0} = w_0 - \frac{\alpha}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})$$

$$temp1 = w_1 - \alpha \frac{\partial J(w_0,w_1)}{\partial w_1} = w_1 - \frac{\alpha}{N}\sum_{i=1}^{N}(w_0 + w_1 x^{(i)} - y^{(i)})x^{(i)}$$

$$w_0 = temp0$$

$$w_1 = temp1$$

α learning rate

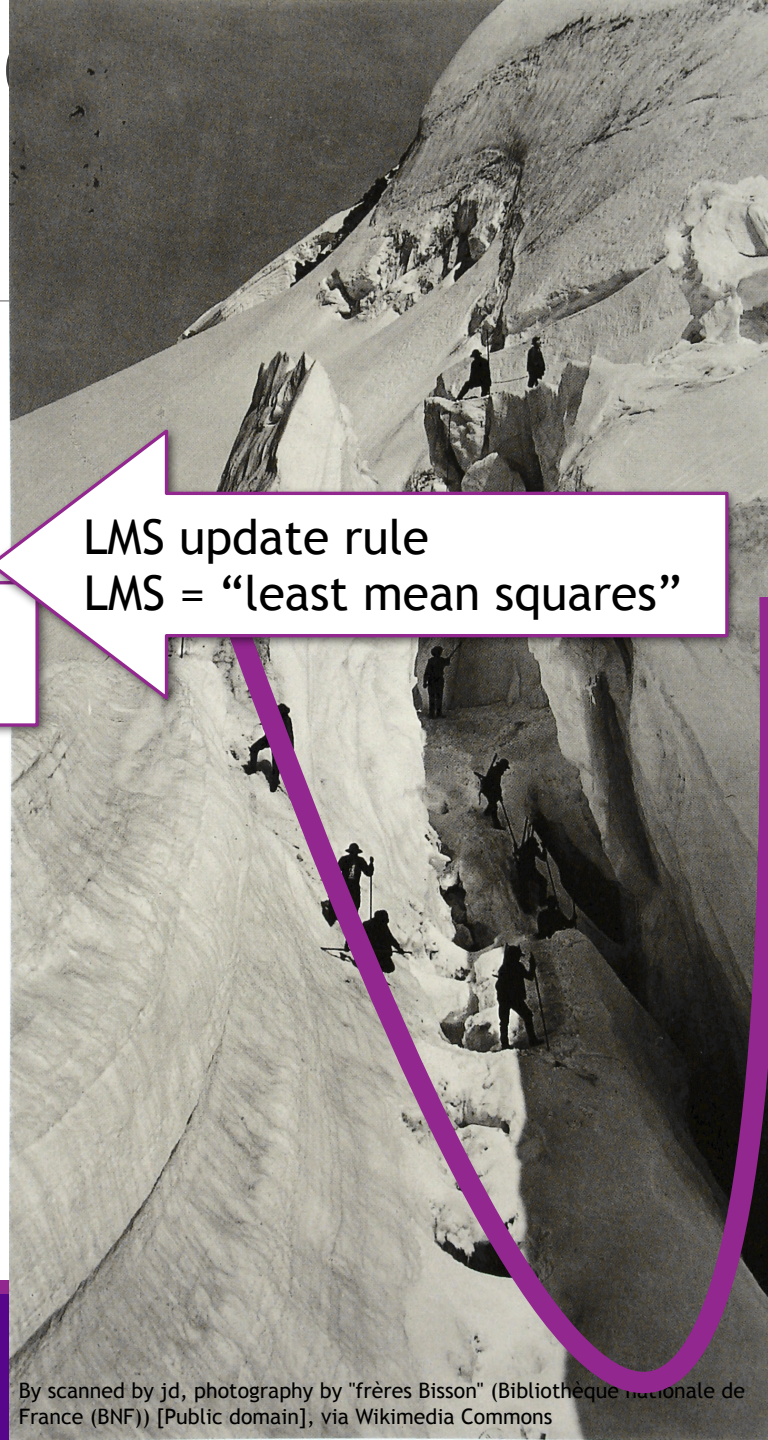
Simultaneous update

LMS update rule
LMS = "least mean squares"

- While not converged (or close enough)

  take a step toward the bottom by changing

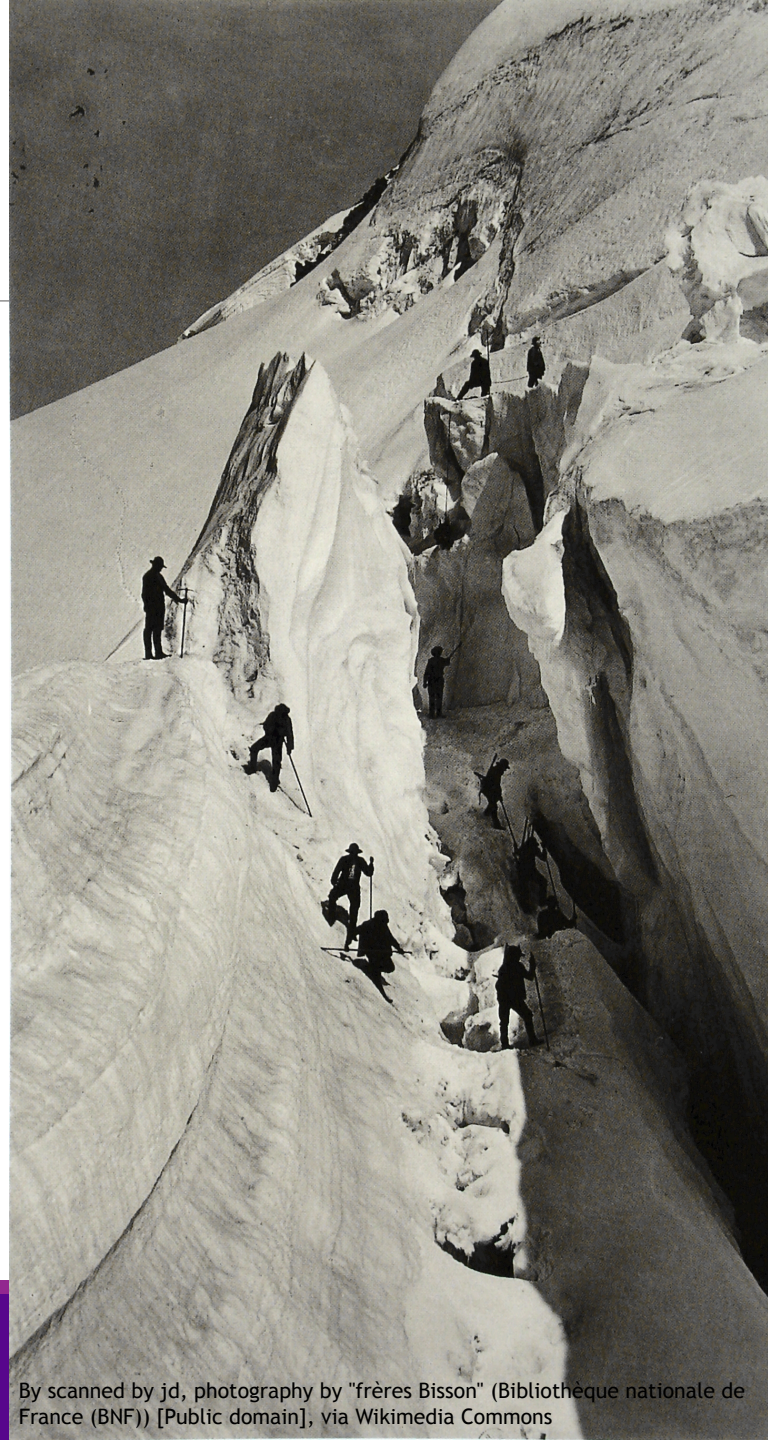
  (thus we have reduced our error!)



By scanned by jd, photography by "frères Bisson" (Bibliothèque nationale de France (BNF)) [Public domain], via Wikimedia Commons

Batch Gradient Descent

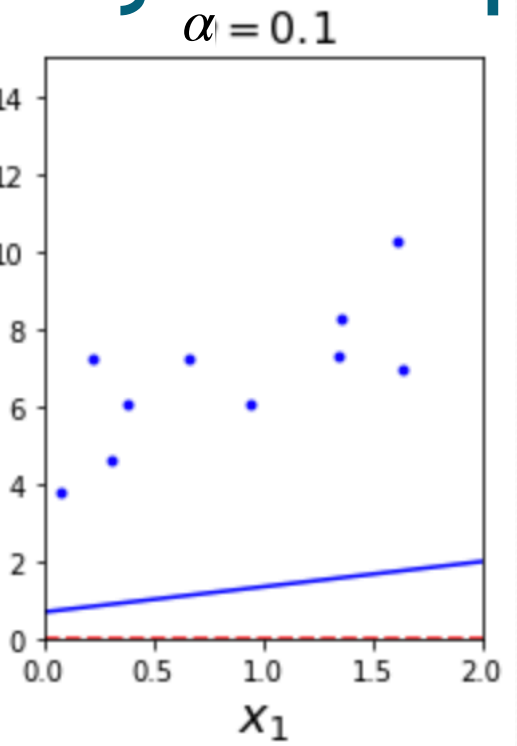
- We need some initial values for w_0, w_1. We can choose any initial values. We will choose $w_0 = 0, w_1 = 0$
- How do we choose our step size, α?
- The gradient descent applies to more general functions than RSS



By scanned by jd, photography by "frères Bisson" (Bibliothèque nationale de France (BNF)) [Public domain], via Wikimedia Commons

Toy example:

*Numbers were rounded



If $\mathbf{w} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ then $\hat{y}^{(i)} = 0$ for all training examples.

The partial derivative of J with respect to w_0 at the point $w_0 = 0,\ w_1 = 0$

$$\frac{\partial J(0,0)}{\partial w_0} = \frac{1}{10}\left((0 - 6.06) + (0 - 3.79) + \cdots + (0 - 6.96)\right)$$

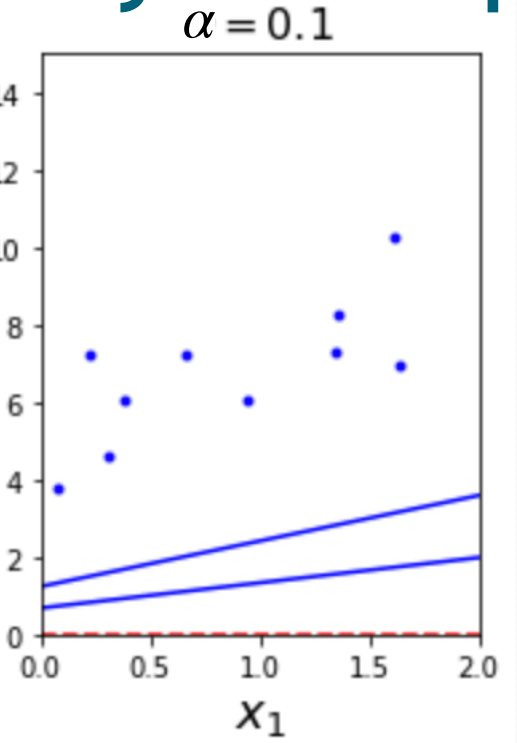
$$\frac{\partial J(0,0)}{\partial w_1} = \frac{1}{10}\left((0 - 6.06)(0.38) + (0 - 3.79)(0.08) + \cdots + (0 - 6.96)(1.64)\right) = -6.53$$

$$\mathbf{w} = \begin{bmatrix} 0.68 \\ 0.65 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 0.1 \begin{bmatrix} -6.78 \\ -6.53 \end{bmatrix}$$

The current $w_0 = w_1 = 0$,
The learning rate is $\alpha = 0.1$

$$X = \begin{bmatrix} 0.38 \\ 0.08 \\ 0.31 \\ 1.62 \\ 0.66 \\ 0.94 \\ 1.35 \\ 0.22 \\ 1.36 \\ 1.64 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 6.06 \\ 3.79 \\ 4.64 \\ 10.29 \\ 7.21 \\ 6.08 \\ 7.31 \\ 7.21 \\ 8.24 \\ 6.96 \end{bmatrix}$$

Toy example:



$$X = \begin{bmatrix} 0.38 \\ 0.08 \\ 0.31 \\ 1.62 \\ 0.66 \\ 0.94 \\ 1.35 \\ 0.22 \\ 1.36 \\ 1.64 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 6.06 \\ 3.79 \\ 4.64 \\ 10.29 \\ 7.21 \\ 6.08 \\ 7.31 \\ 7.21 \\ 8.24 \\ 6.96 \end{bmatrix}$$

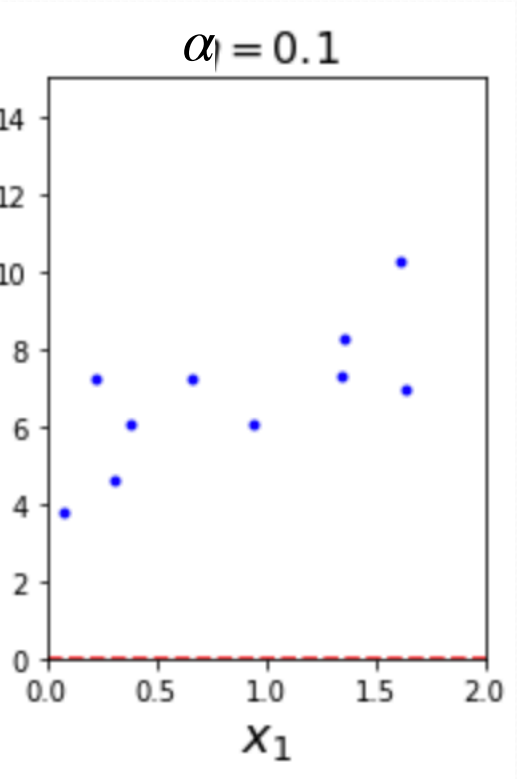
For $\mathbf{w} = \begin{bmatrix} 0.68 \\ 0.65 \end{bmatrix}$ then $\hat{\mathbf{y}} = \begin{bmatrix} 1 & 0.38 \\ 1 & 0.08 \\ \vdots & \\ 1 & 1.64 \end{bmatrix} \begin{bmatrix} 0.68 \\ 0.65 \end{bmatrix} = \begin{bmatrix} 0.93 \\ 0.73 \\ \vdots \\ 1.75 \end{bmatrix}$

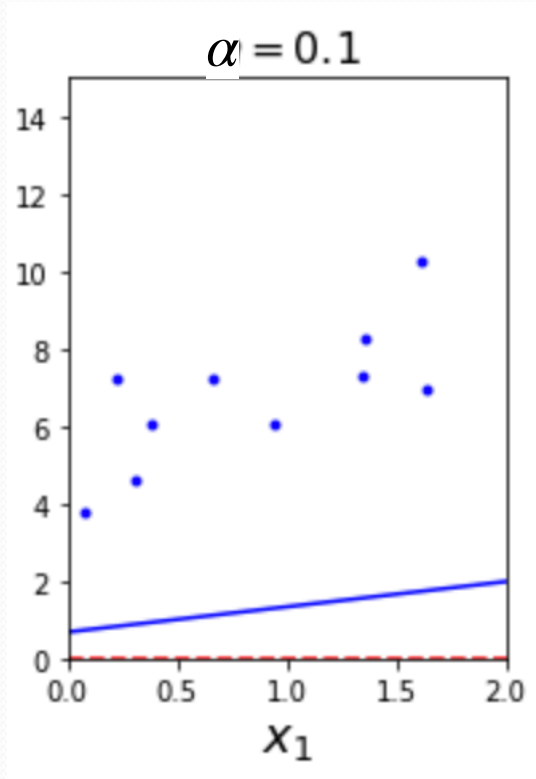
The partial derivative of J with respect to w_0 at the point $w_0 = 0.68,\ w_1 = 0.65$

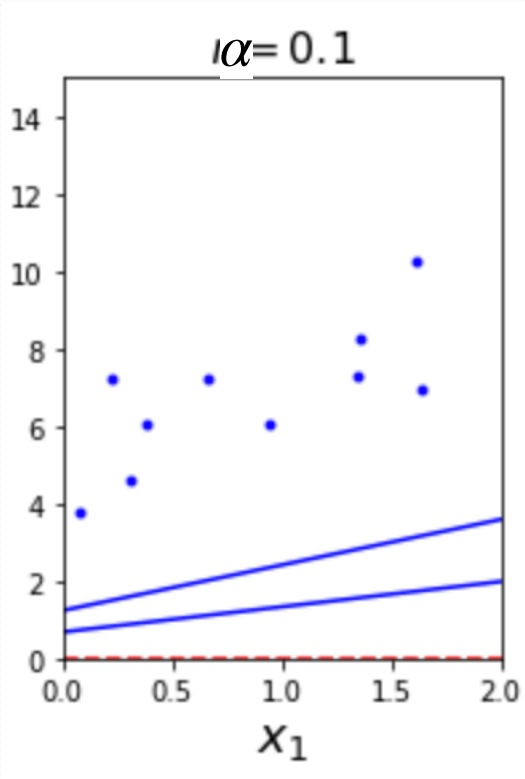
$$\frac{\partial J(0.68,0.65)}{\partial w_0} = \frac{1}{10}\left((0.93 - 6.06) + (0.73 - 3.79) + \cdots + (1.75 - 6.96)\right) = -5.54$$

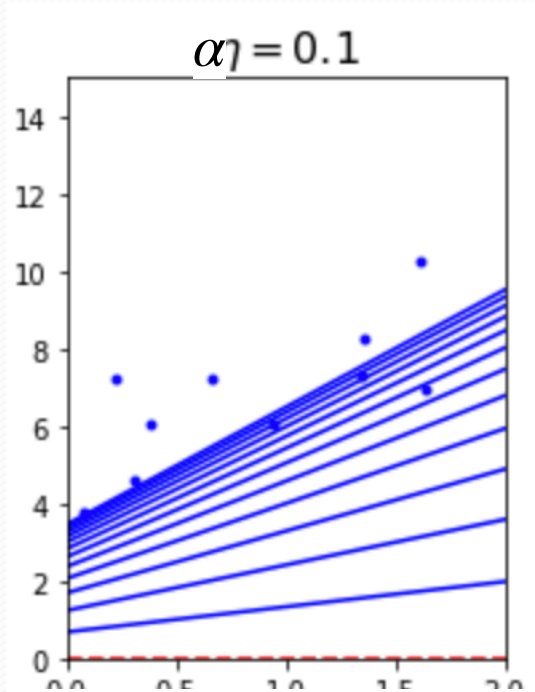
$$\frac{\partial J(0.68,0.65)}{\partial w_1} = \frac{1}{10}\left((0.93 - 6.06)(0.38) + (0.73 - 3.79)(0.08) + \cdots + (1.75 - 6.96)(1.64)\right) = -5.25$$

$$\mathbf{w} = \begin{bmatrix} 1.23 \\ 1.17 \end{bmatrix} = \begin{bmatrix} 0.68 \\ 0.65 \end{bmatrix} - 0.1 \begin{bmatrix} -5.54 \\ -5.25 \end{bmatrix}$$

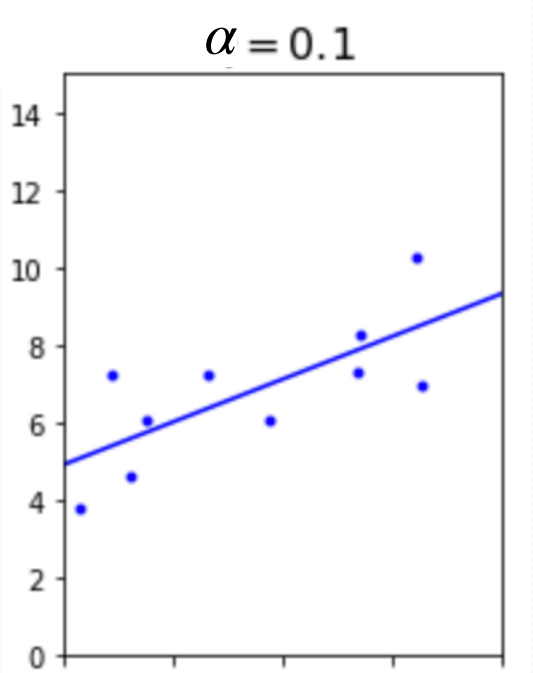




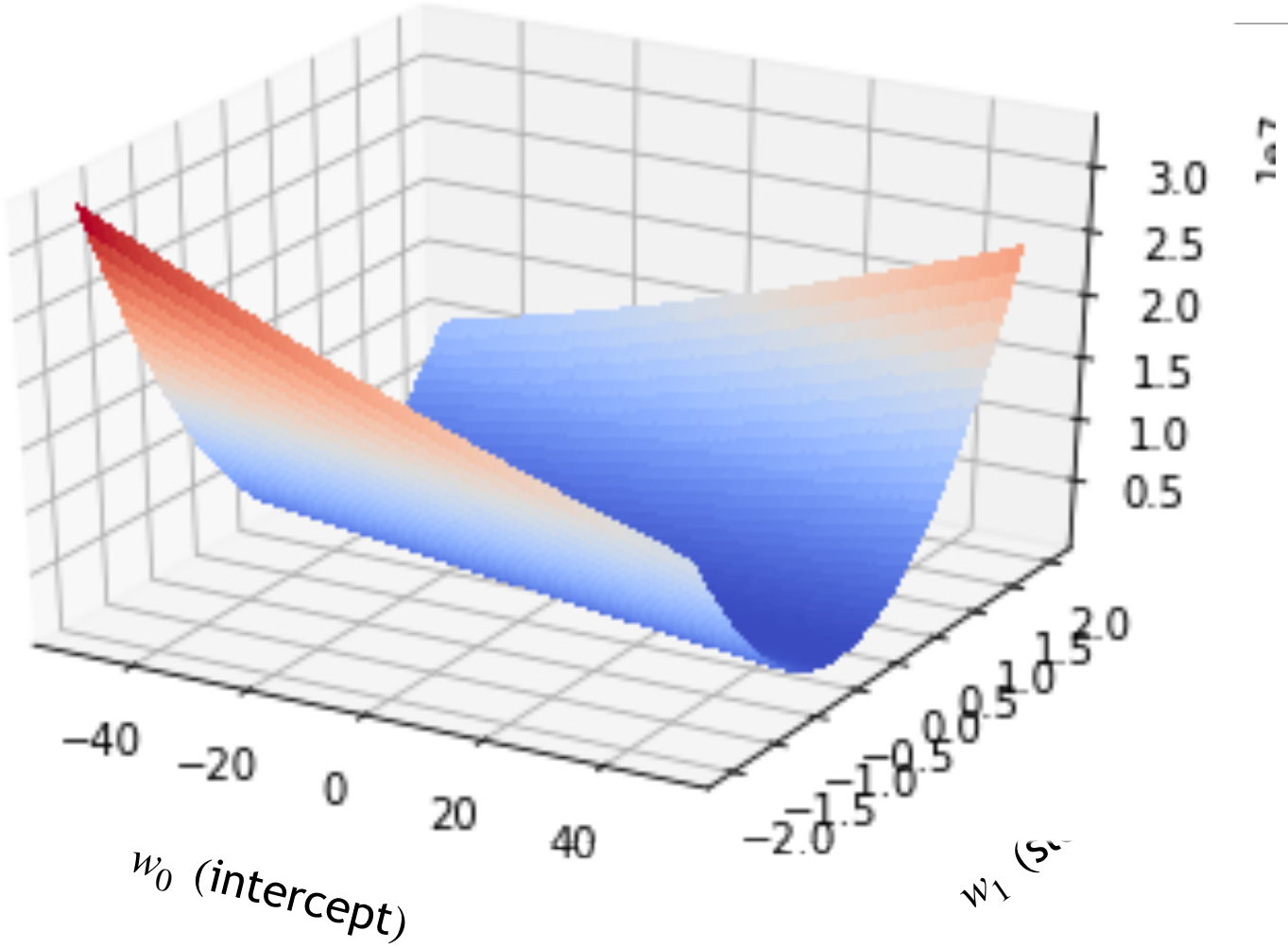


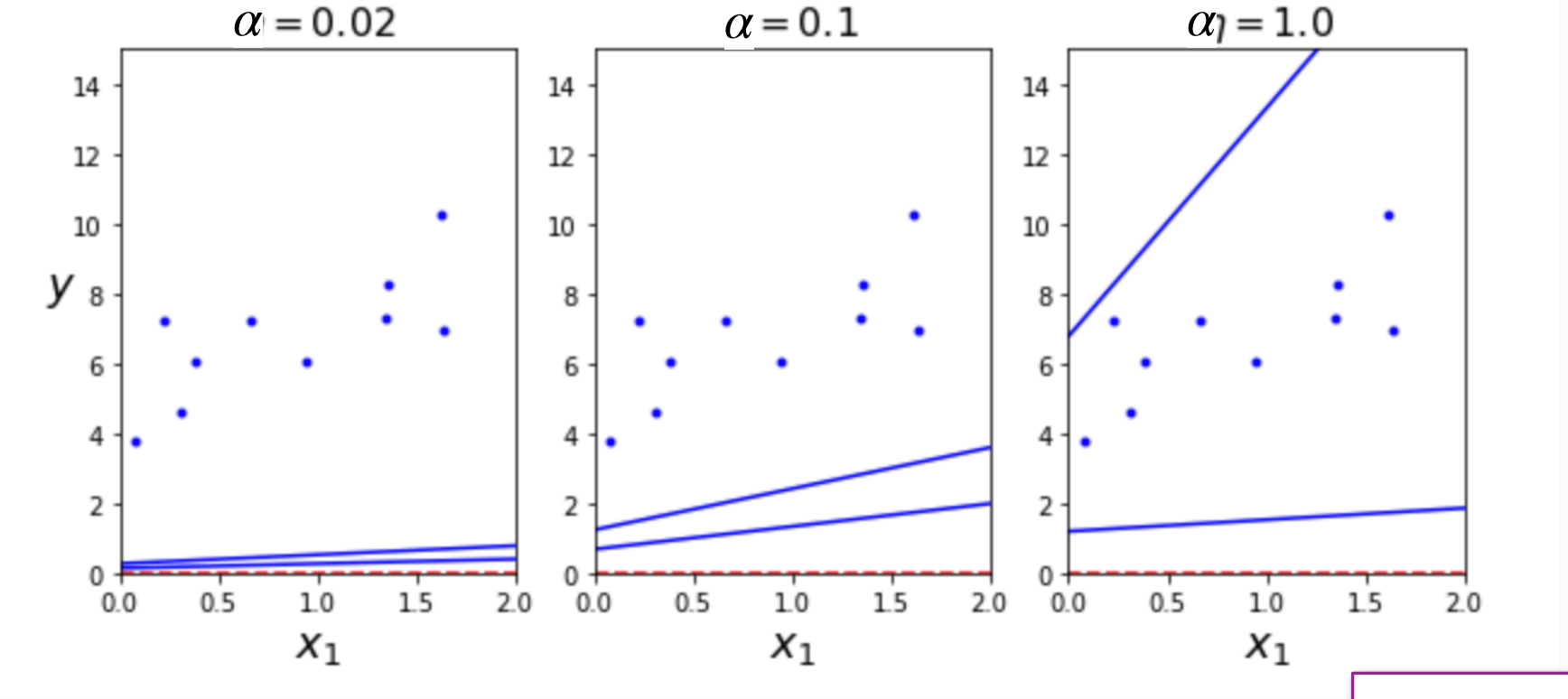


$$X = \begin{bmatrix} 0.38 \\ 0.08 \\ 0.31 \\ 1.62 \\ 0.66 \\ 0.94 \\ 1.35 \\ 0.22 \\ 1.36 \\ 1.64 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 6.06 \\ 3.79 \\ 4.64 \\ 10.29 \\ 7.21 \\ 6.08 \\ 7.31 \\ 7.21 \\ 8.24 \\ 6.96 \end{bmatrix}$$



Pair share: what could go wrong with Local optimization methods?





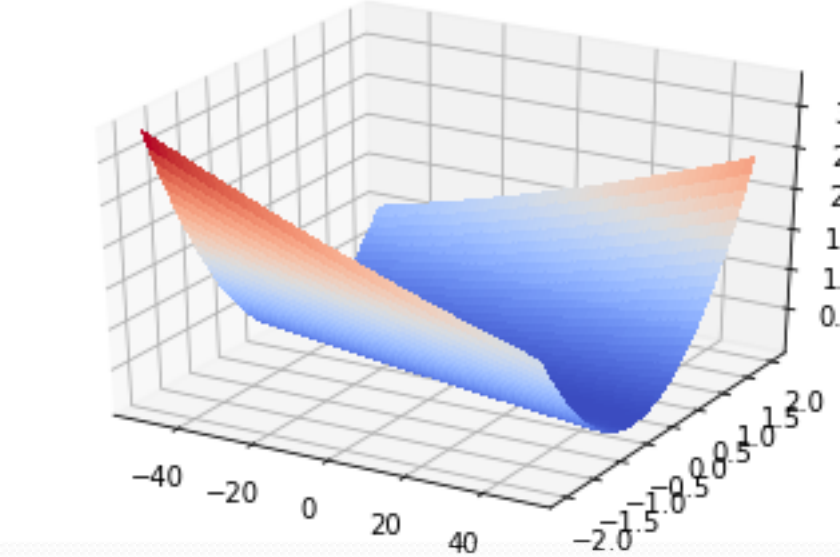
Not covered in this class for linear regression are heuristics for adaptively changing the learning rate.

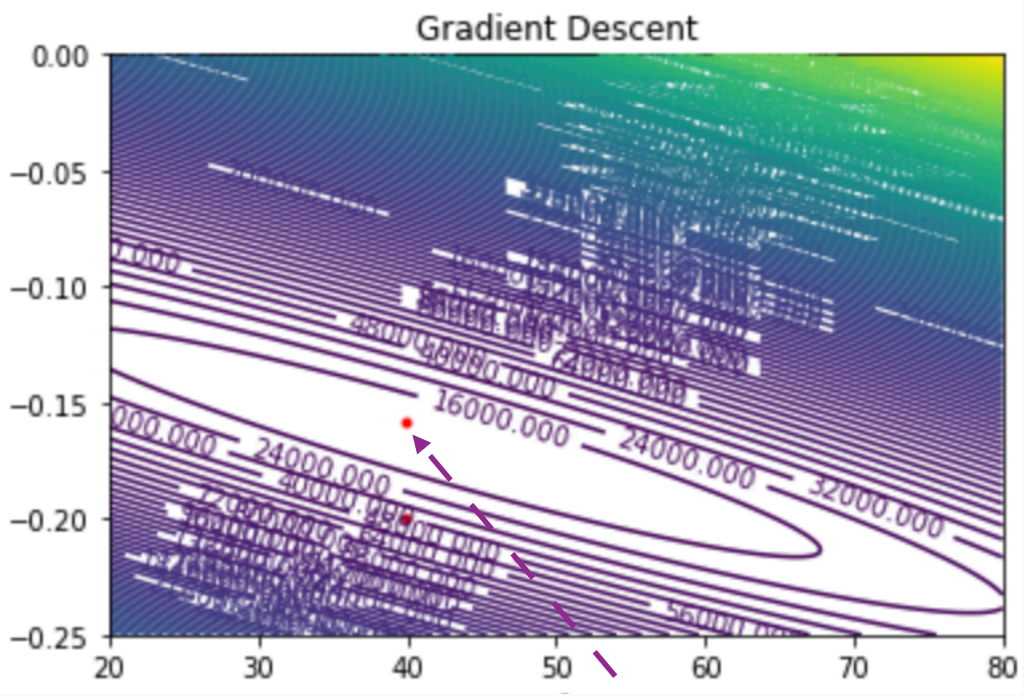
You can read on your own the choices available in Sklearn's implementation of SGD for linear regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor

Another way of looking at our cost function/algorithm

Contour Plots

Graphical technique for representing a 3-D surface in 2-D plane





$RSS(39.94, \ -0.16) = 9385.92$

- **Contour curve**: a 3-D contour curve can be formed on the 3-D surface by finding all the points which have the same Z value (e.g. RSS(w_0, w_1)=k, where k is a constant value)
- **Level curve**: are in the the 2-D plane and can be created by projecting the contour curve onto 2-D plane (e.g.the w_0, w_1 axis)
- **Contour plot**: a 2-D plot containing level curves of a function

Read more at: https://ocw.mit.edu/courses/mathematics/18-02sc-multivariable-calculus-fall-2010/2.-partial-derivatives/part-a-functions-of-two-variables-tangent-approximation-and-optimization/session-25-level-curves-and-contour-plots/MIT18_02SC_notes_15.pdf

Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Global Optimizer: LS Fit Solution
- Local Optimizer: Gradient Descent
- Objective function revisited: Probabilistic interpretation
- Assessing Goodness of Fit
- Extra Slides: Global Optimizer for multivariate linear regression: Normal Equation

How well does our model fit the training data?

If you study for a test and don't generalize, will you do well? The goal is to be able to do well for questions you have not seen before

Next model we will discuss training and test data

i.e. How well did you predict the mpg?

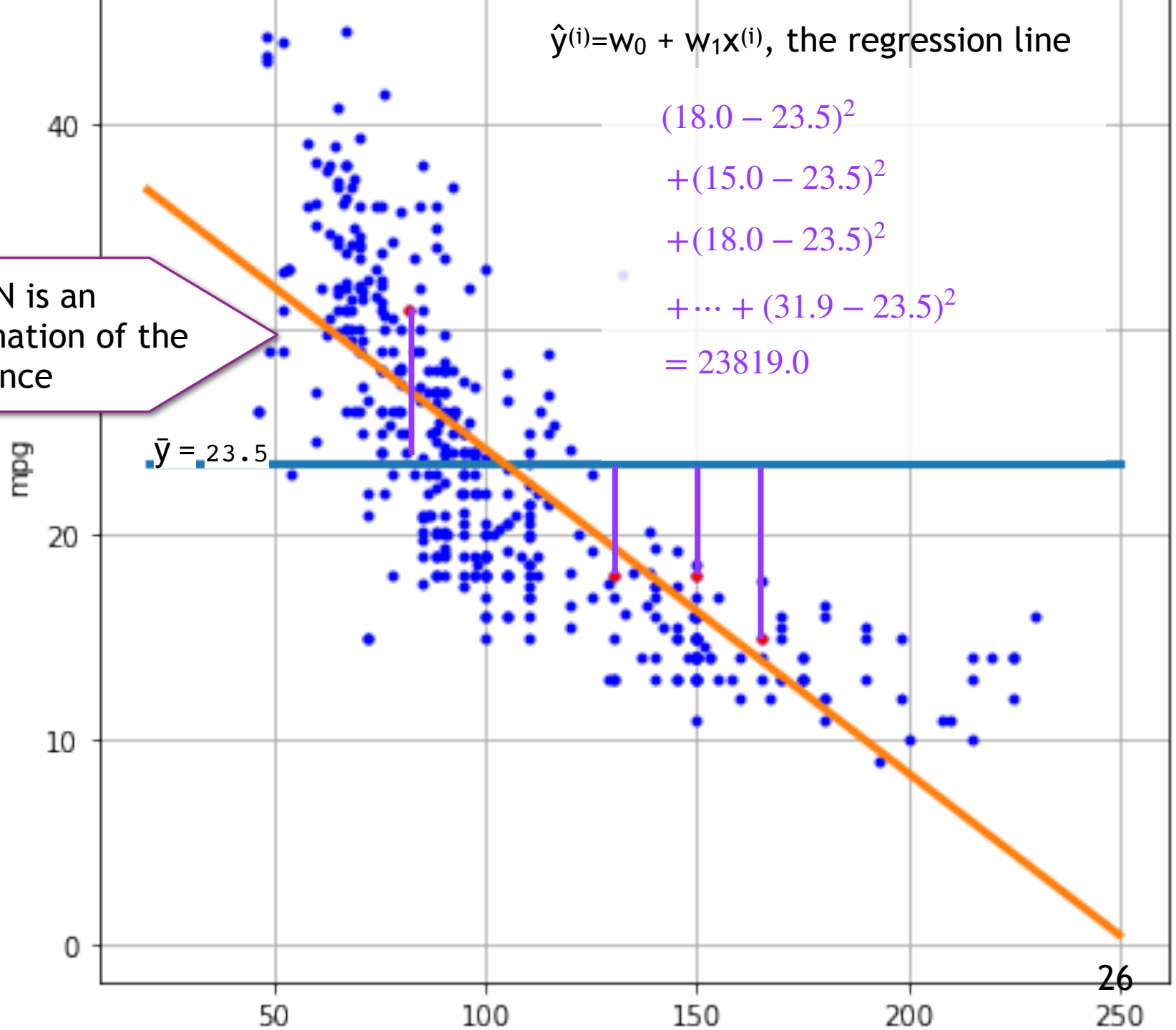
Variability of y

$$\text{TSS} = \sum_{i=1}^{N} (y^{(i)} - \bar{y})^2 = 23819.0$$

$X =$ [130.0, 165.0, 150.0, 150.0, 140.0, 198.0, 220.0, 215.0, ..]

y= [18.0, 15.0, 18.0, 16.0, 17.0, 15.0, 14.0, 14.0, ...]

TSS/N is an estimation of the variance



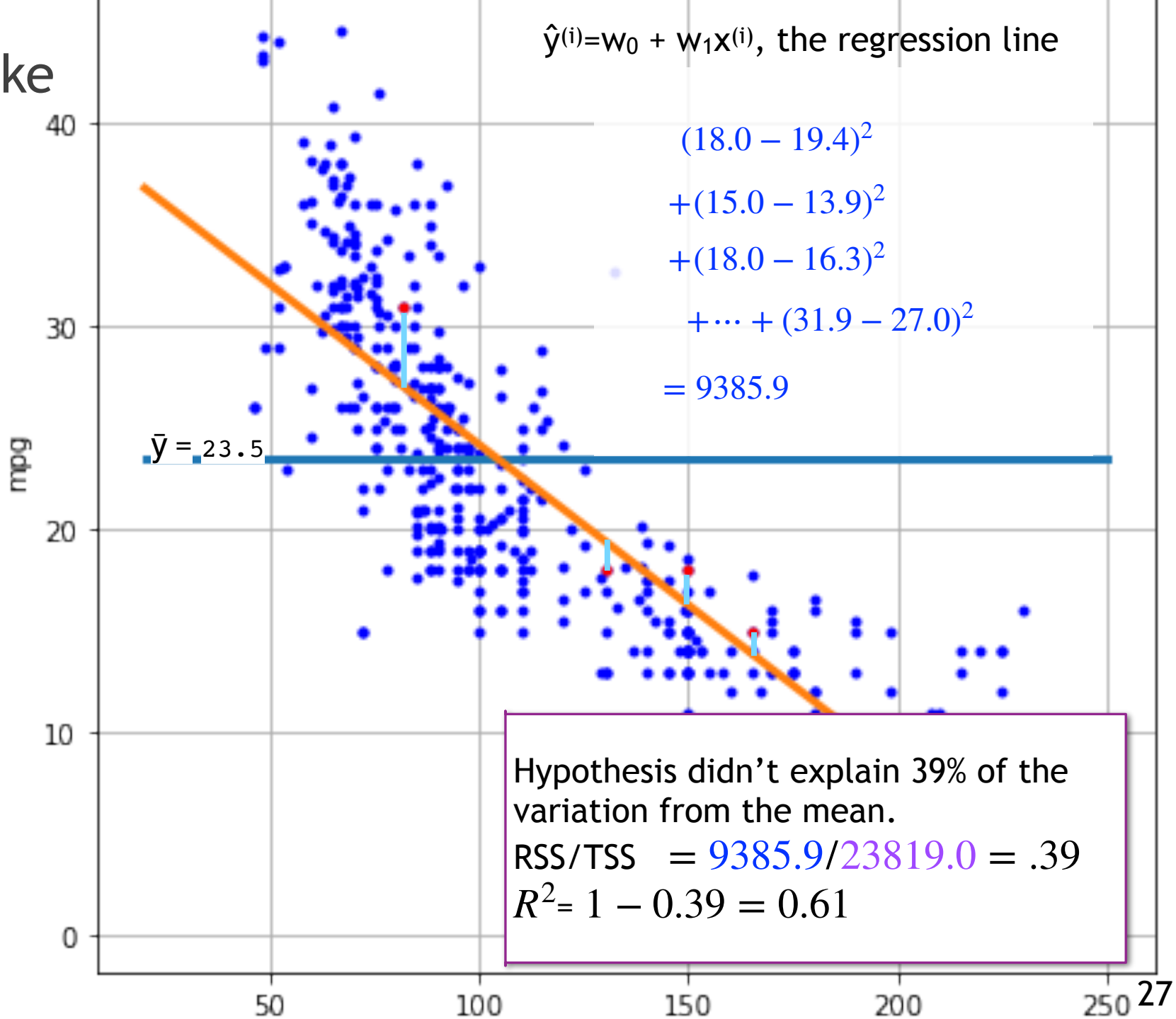
Variability of y when we take into account $\mathbf{x}$

$$\text{TSS} = \sum_{i=1}^{N} (y^{(i)} - \bar{y})^2 = 23819.0$$

$$\text{RSS} = \sum_{i=1}^{N} (y^{(i)} - \hat{y}^{(i)})^2 = 9385.9$$

$X =$ [130.0, 165.0, 150.0, 150.0, 140.0, 198.0, 220.0, 215.0, ...]

y= [18.0, 15.0, 18.0, 16.0, 17.0, 15.0, 14.0, 14.0, ...]



Hypothesis didn't explain 39% of the variation from the mean.

RSS/TSS $= 9385.9/23819.0 = .39$

$R^2 = 1 - 0.39 = 0.61$

Computing R^2 in Python

```
xstr = 'horsepower'
X = np.array(df[xstr])
y = np.array(df['mpg'])
```

```
[130. 165. 150. 150. 140. 198. 220. 215. 225. 190. …]
```

```
[18. 15. 18. 16. 17. 15. 14. 14. 14. 15.  …]
```

```
ym = np.mean(y)
```

```
 23.45
```

```
yhat=w0+w1*X
```

```
[19.42 13.89 16.26 16.26 17.84  8.68  5.21  6.    4.42  9.95, …]
```

```
RSS = np.sum((y-yhat)**2)
TSS = np.sum( (y-ym)**2)
R_sqrd = 1-RSS/TSS
print("R^2 = {0:7.2f}".format(R_sqrd))
```

```
R^2 =    0.61
```

```
N= df.shape[0]
xstr = 'horsepower'
X = np.array(df[xstr]).reshape((N,1))
y = np.array(df['mpg']).reshape((N,1))
```

```
[[130.]          [[18.]
 [165.]           [15.]
 [150.]           [18.]
 [150.]           [16.]
 [140.]           [17.]
 [198.]           [15.]
 [220.]           [14.]
 [215.]           [14.]
 [225.]           [14.]
 [190.]           [15.]
  …                 …
      ]                 ]
```

Coding cont.

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
5	15.0	8	429.0	198.0	4341.0	10.0	70	1	ford galaxie 500

```
names = ['mpg', 'cylinders','displacement', 'horsepower',
         'weight', 'acceleration', 'model year', 'origin', 'car name']
```

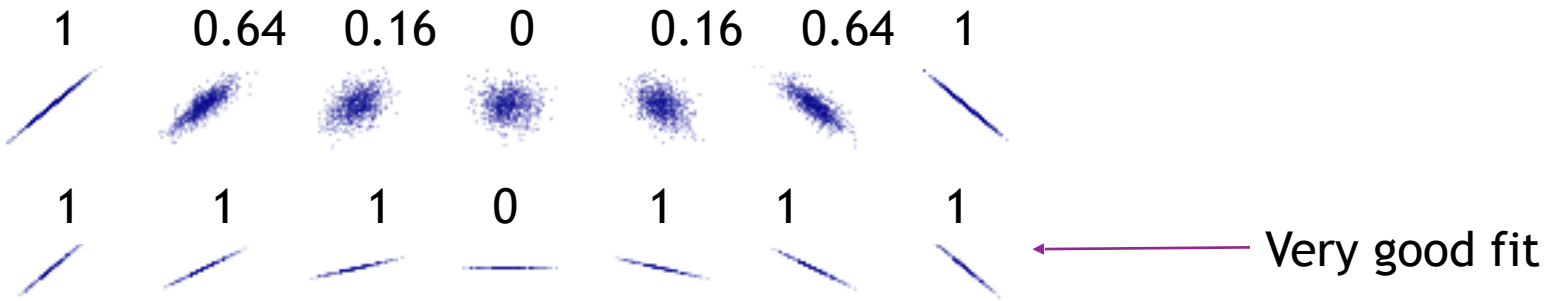
```
for k in range(2,len(names)-1,1):
    df1 = df[['mpg',names[k]]]
    df2 = df1.dropna()
    X=np.array(df2[names[k]])
    y=np.array(df2['mpg'])
    w0,w1,rsq=fit_linear(X,y) # returns w0, w1, rsq
    print(names[k],": ", np.around(rsq,2))
```

```
displacement :  0.65
horsepower :  0.61
weight :  0.69
acceleration :  0.18
model year :  0.34
origin :  0.32
```

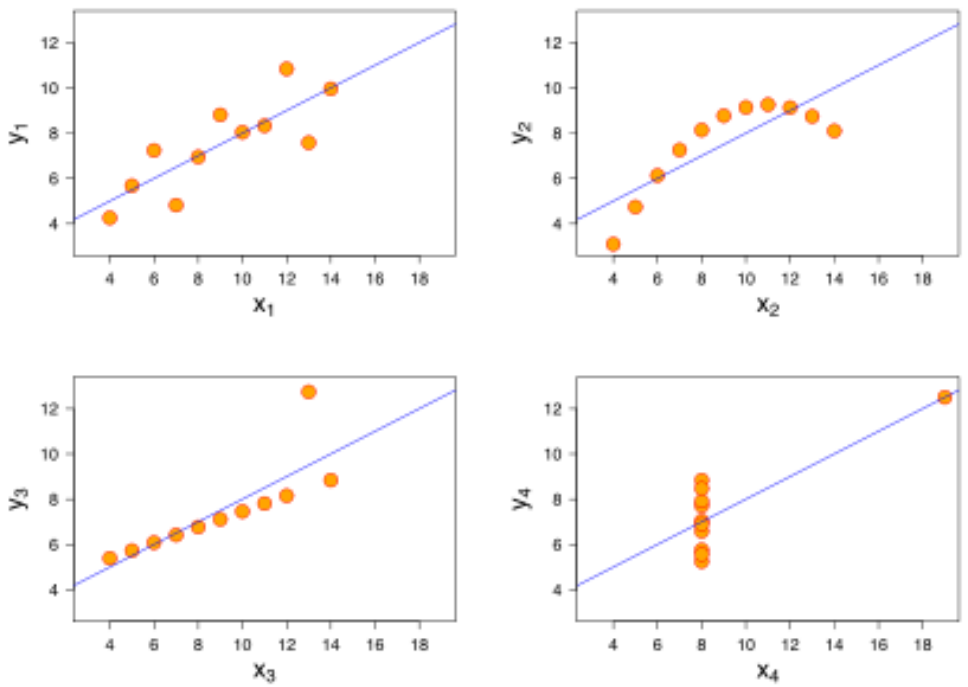
Yikes! This shows we should try all the features to determine the one that works best

Visually seeing correlation

- $R^2 \approx 1$ Linear model is a very good fit
- $R^2 \approx 0$ Linear model isn't better than just predicting the mean



Errors …

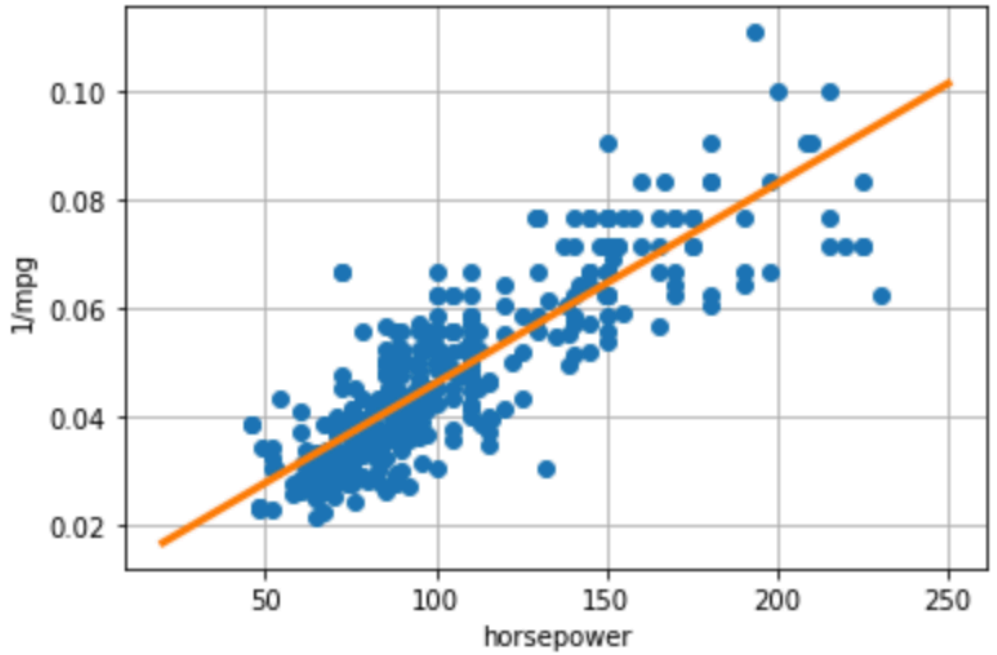


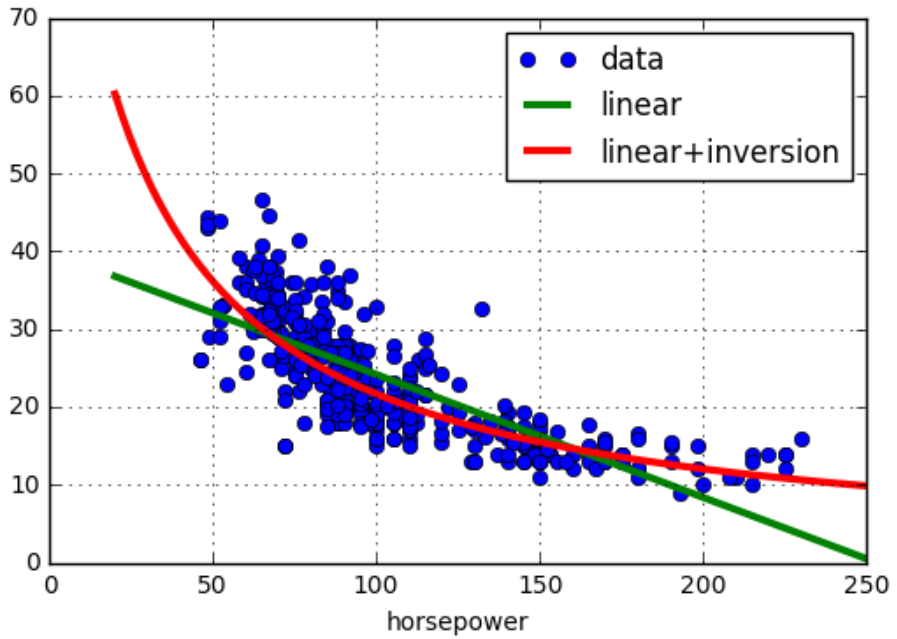
- Many sources of error for a linear model
- Example to the left
  - All four data sets have same regression line

    $\hat{y}$= 3.00 + 0.500*x*
  - But, errors and their reasons are different
- How would you describe these errors?
- All 4 graphs have a R^2 of 0.67

A Better Model for the Auto Example

- Fit the inverse: $\frac{1}{\text{mpg}} = w_0 + w_1 \text{ horsepower}$
- Uses a nonlinear transformation
- *Will cover transforming the data later*





Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Global Optimizer: LS Fit Solution
- Local Optimizer: Gradient Descent
- Objective function revisited: Probabilistic interpretation
- Assessing Goodness of Fit
- Extra Slides: Global Optimizer for multivariate linear regression: Normal Equation

Extra slides not discussed in class

Normal Equation Method AKA Least Squares

$$X = \begin{bmatrix} 1 & 130.0 \\ 1 & 165.0 \\ 1 & 150.0 \\ 1 & 150.0 \\ 1 & 140.0 \\ 1 & 198.0 \\ 1 & 220.0 \\ 1 & 215.0 \\ \dots & \end{bmatrix} \quad y = \begin{bmatrix} 18.0 \\ 15.0 \\ 18.0 \\ 16.0 \\ 17.0 \\ 15.0 \\ 14.0 \\ 14.0 \\ \dots \end{bmatrix}$$

- The optimal value of parameters w_0, w_1 for linear regression can be directly found in a single equation
- Before implementing the equation, append a column of ones in the data matrix, X
- The formula is: $\mathbf{w} = (X^T X)^{-1} X^T y$
- For *simple* linear regression:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \qquad X = \begin{bmatrix} 1 & \mathbf{x}^{(1)} \\ 1 & \mathbf{x}^{(2)} \\ \vdots & \vdots \\ 1 & \mathbf{x}^{(N-1)} \\ 1 & \mathbf{x}^{(N)} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N-1)} \\ y^{(N)} \end{bmatrix}$$

Example:

$$X = \begin{bmatrix} 1 & 130 \\ 1 & 165 \\ 1 & 150 \\ 1 & 150 \\ 1 & 140 \\ 1 & 198 \\ 1 & 220 \\ 1 & 215 \\ 1 & \dots \end{bmatrix} \quad y = \begin{bmatrix} 18.0 \\ 15.0 \\ 18.0 \\ 16.0 \\ 17.0 \\ 15.0 \\ 14.0 \\ 14.0 \\ \dots \end{bmatrix}$$

```
N = X.shape[0]
a = np.ones((N, 1))
X = np.hstack((a, X))
```

```
X.T
```

$$\mathbf{w} = (X^T X)^{-1} X^T y$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 130 & 165 & 150 & 150 & 140 & 198 & 220 & 215 & \dots \end{bmatrix} \begin{bmatrix} 1 & 130 \\ 1 & 165 \\ 1 & 150 \\ 1 & 150 \\ 1 & 140 \\ 1 & 198 \\ 1 & 220 \\ 1 & 215 \\ & \dots \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ 130 & 165 & 150 & 150 & 140 & 198 & 220 & 215 & \dots \end{bmatrix} \begin{bmatrix} 18 \\ 15 \\ 16 \\ 17 \\ 15 \\ 14 \\ 14 \\ \dots \end{bmatrix}$$

Normal Equation Method AKA Least Squares

- The normal equation method $\mathbf{w} = \left(A^T A\right)^{-1} \cdot \left(A^T \mathbf{y}\right)$ can be used to find the parameters for the multiple linear regression problem.
- Before implementing the equation, append a column of ones in the data matrix, X
- Now **w**, A, and **y** are:

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{d-1} \\ w_d \end{bmatrix} \quad A = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(N)} & \dots & x_d^{(N)} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N-1)} \\ y^{(N)} \end{bmatrix}$$

R^2 intuition, "goodness of fit"

- We used linear regression to find the best estimate for our coefficients
- Computing the RSS (residual sum of squares) gives an intuition on how much error our model has. A better model has a smaller RSS
- We can represent how well our models is doing as a percentage of reduction of the original prediction error.

$$R^2 = \text{Explained variation/total variation} = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS} \qquad TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2 \qquad RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- R^2 is called the *coefficient of determination*
- R^2 is always between 0 and 1 (0 meaning the models does not explain any of the variation and 1 meaning the model explains all of the variation
- The higher the R^2 score, the better the model fits the data
- Limitations:
  - R^2 cannot determine if the coefficient estimates and predictions are biased - you must look at the residual plots
  - A low score might be a good model (e.g. attempting to model human behavior typically has R^2 less than 50%)

Simple Example

- From: http://stattrek.com/regression/regression-example.aspx?Tutorial=AP
  - Very nice simple problems
- Predict aptitude on one test from an earlier test
- Draw a scatter plot and regression line

In this link b_0 is used for the intercept and b_1 is used for the slope

How to Find the Regression Equation

In the table below, the x_i column shows scores on the aptitude test. Similarly, the y_i column shows statistics grades. The last two rows show sums and mean scores that we will use to conduct the regression analysis.

	Student	x_i	y_i	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})^2$	$(y_i - \bar{y})^2$	$(x_i - \bar{x})(y_i - \bar{y})$
	1	95	85	17	8	289	64	136
	2	85	95	7	18	49	324	126
	3	80	70	2	-7	4	49	-14
	4	70	65	-8	-12	64	144	96
	5	60	70	-18	-7	324	49	126
Sum		390	385			730	630	470
Mean		78	77					

The regression equation is a linear equation of the form: $\hat{y} = b_0 + b_1 x$. To conduct a regression analysis, we need to solve for b_0 and b_1. Computations are shown below.

$b_1 = \Sigma [(x_i - \bar{x})(y_i - \bar{y})] / \Sigma [(x_i - \bar{x})^2]$ $b_1 = 470/730 = 0.644$	$b_0 = \bar{y} - b_1 * \bar{x}$ $b_0 = 77 - (0.644)(78) = 26.768$

Slide NOT part of lecture. I am only including this slide for those students who are interested.

Justification for gradient descent using the Taylor Expansion

Taylor expansion for a univariate real-valued function:

$$f(w+\Delta w) = f(w) + f'(w)\Delta w + f''(w)\frac{(\Delta w)^2}{2!} + f'''(w)\frac{(\Delta w)^3}{3!} + \cdots$$

We can approximate this function for ***small*** Δw by the first-order Taylor approximation:

$$f(w+\Delta w) \approx f(w) + f'(w)\Delta w$$

First-order Taylor expansion for a Multivariate real-valued function:

$$f(\mathbf{w}+\Delta\mathbf{w}) \approx f(\mathbf{w}) + \Delta\mathbf{w}^T \nabla f(\mathbf{w})$$

Now if we choose $\Delta\mathbf{w} = -\alpha\nabla f(\mathbf{w})$ then $f(\mathbf{w}+\Delta\mathbf{w}) \approx f(\mathbf{w}) - \alpha\nabla f(\mathbf{w})^T \nabla f(\mathbf{w}) < f(\mathbf{w})$

Positive if the gradient is not zero

Now we just need to make sure that we don't move to far so the approximation is reasonable!

Slide NOT part of lecture. I am only including this slide for those students who are interested.

From: http://cs229.stanford.edu/notes/cs229-notes1.pdf

Remember:

$\log(a \cdot b \cdot c) = \log a + \log b + \log c$

Thus

$\log \prod p(\epsilon^{(i)}) = \sum \log p(\epsilon^{(i)})$

What cost makes sense? (cont.)

Which parameter w is best?

The one that is most likely is the one that maximizes $L(\mathbf{w}) = L(\mathbf{w}; X, \mathbf{y})$

$$L(\mathbf{w}) = \prod_{i=1}^{N} p(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}) = \prod_{i=1}^{N} P(\epsilon^{(i)}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} exp \frac{-\left(y^{(i)} - (w_0 + w_1 \mathbf{x}^{(i)})\right)^2}{2\sigma^2}$$

Just algebraic manipulation

Note that maximizing this value is the same as maximizing $\ell(\mathbf{w}) = \log L(\mathbf{w})$

We are using ℓ for the log likelihood function

$$\ell(\mathbf{w}) = \log \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} exp \frac{-\left(y^{(i)} - (w_0 + w_1 \mathbf{x}^{(i)})\right)^2}{2\sigma^2}$$

$$= \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} exp \frac{-\left(y^{(i)} - (w_0 + w_1 \mathbf{x}^{(i)})\right)^2}{2\sigma^2}$$

$$= N \log \frac{1}{\sqrt{2\pi\sigma^2}} + \frac{1}{2\sigma^2} \sum_{i=1}^{N} - \left(y^{(i)} - (w_0 + w_1 \mathbf{x}^{(i)})\right)^2$$

This … is the same as minimizing $\sum_{i=1}^{N} \left(y^{(i)} - (w_0 + w_1 \mathbf{x}^{(i)})\right)^2$

Additional information can be found at: https://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/06/lecture-06.pdf
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-867-machine-learning-fall-2006/lecture-notes/lec5.pdf