Do not distribute course material

Linear Regression Continued

Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Local Optimizer: “Batch” Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation.
- Extensions
- Removing features

This is a very brief introduction to the topic. We will transform before training.

Why do we transform features

Warning - all transformations needs to also be performed on any test/validation/new data.

2nd

- Necessary transformations
  - Some algorithms require numeric data
  - Some algorithms expect the input to be of a specific size
- Optional transformations
  - Normalizing numeric features
  - Creating non-linearities in the feature space - thus allowing us to still use linear models (next topic)

Necessary transformations

Many algorithms require numerical data

How would we handle Categorical data?

Categorical data contains labels: {BMW, VW, Ford, GM} or {low, medium, high}, etc

59	Female	'<1H OCEAN'	Cold
48	Male	‘ <1H OCEAN’	Warm
72	Female	'NEAR OCEAN'	Cold
24	Male	‘INLAND’	Hot
50	Male	'<1H OCEAN’	Warm
23	Male	'NEAR BAY’	Warm
36	Female	`NEAR OCEAN’	Cold

categorical data

Some categories contain a *natural ordering*: low, medium, high

Other categories *don't*: BMW, VW, Ford, GM

Many machine learning algorithms cannot work with categorical data.

How can we convert them to numerical data?

Two common approaches:

- Ordinal encoding
- One-hot encoding

ordinal encoding

If the $x_j \in \{$ cold, warm, hot$\}$

Cold	0
Warm	1
Cold	0
Hot	2
Warm	1

Suppose x_i is a categorical variable

- One of a finite number of choices
- Example: “place” (gold silver bronze), (cold, warm, hot), etc

“**Ordinal encoding** should be used for *ordinal variables* (where order matters, like `cold`, `warm`, `hot`)”

- Assigns an integer to encode each value

Warning! Sklearn OrdinalEncoder class assigns integers to categories based on alphabetic ordering
If you want the “right” encoding - assign the order “manually” by using the *categories* argument.

https://feature-engine.readthedocs.io/en/latest/encoding/OrdinalEncoder.html

https://datascience.stackexchange.com/questions/39317/difference-between-ordinalencoder-and-labelencoder

One Hot Coding

```
<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'
```

Suppose x_i is a categorical variable

- One of a finite number of choices
- Example: male or female, or model of a car, location of a house, etc

```
                      <1H OCEAN'  'INLAND'  'ISLAND'  'NEAR BAY'  'NEAR OCEAN'
[['<1H OCEAN']        [1.,        0.,       0.,       0.,         0.],
 ['<1H OCEAN']        [1.,        0.,       0.,       0.,         0.],
 ['NEAR OCEAN']       [0.,        0.,       0.,       0.,         1.],
 ...                  ...,
 ['INLAND']           [0.,        1.,       0.,       0.,         0.],
 ['<1H OCEAN']        [1.,        0.,       0.,       0.,         0.],
 ['NEAR BAY']]        [0.,        0.,       0.,       1.,         0.]
```

Dummy variable encoding

If the $x_i \in \{$ Ford, BMW, GM, VW $\}$

Model	u_1	u_2	u_3
Ford	0	0	0
BMW	1	0	0
GM	0	1	0
VW	0	0	1

Dummy variable encoding is the preferred method for linear regression.
This method avoids creating collinearity

One Hot Coding

```
<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'
```

Suppose x_i is a categorical variable

- One of a finite number of choices
- Example: male or female, or model of a car, location of a house, etc

1 - if value is equal
0 - otherwise

```
[ 'NEAR OCEAN' ]
...
[ 'INLAND' ]
[ '<1H OCEAN' ]
[ 'NEAR BAY' ]]
```

```
'<1H OCEAN'  'INLAND'  'ISLAND'  'NEAR BAY'  'NEAR OCEAN'
[1., 0., 0., 0., 0.],
[1., 0., 0., 0., 0.],
[0., 0., 0., 0., 1.],
...,
[0., 1., 0., 0., 0.],
[1., 0., 0., 0., 0.],
[0., 0., 0., 1., 0.]
```

Dummy variable encoding

If the $x_i \in \{$ Ford, BMW, GM, VW $\}$

If the number of categories is 2, then create a new variable x_i that takes takes two values. E.g. if we have a gender variable create

$$x_i^{(j)} = \begin{cases} 0 \text{ if jth person is female} \\ 1 \text{ if jth person is male} \end{cases}$$

Dummy variable encoding is the preferred method for linear regression.
This method avoids creating collinearity

Toy Example

Preprocessing step: feature transformation

	Standardization	Dummy Variable Encoding	One Hot Encoding	Ordinal Encoding
Data	59	Female	'<1H OCEAN'	Cold
	48	Male	' <1H OCEAN'	Warm
	72	Female	'NEAR OCEAN'	Cold
	24	Male	'INLAND'	Hot
	50	Male	'<1H OCEAN'	Warm
	23	Male	'NEAR BAY'	Warm
	36	Female	'NEAR OCEAN'	Cold

$$
\begin{array}{cc}
 & \begin{array}{cccccccc} & & \text{'<1H OCEAN'} & \text{'INLAND'} & \text{'ISLAND'} & \text{'NEAR BAY'} & \text{'NEAR OCEAN'} & \end{array} \\
X = & \begin{bmatrix}
0.79 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0.19 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1.51 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
-1.14 & 1 & 0 & 1 & 0 & 0 & 0 & 2 \\
0.30 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
-1.19 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
-0.47 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

Standardization for the first feature:

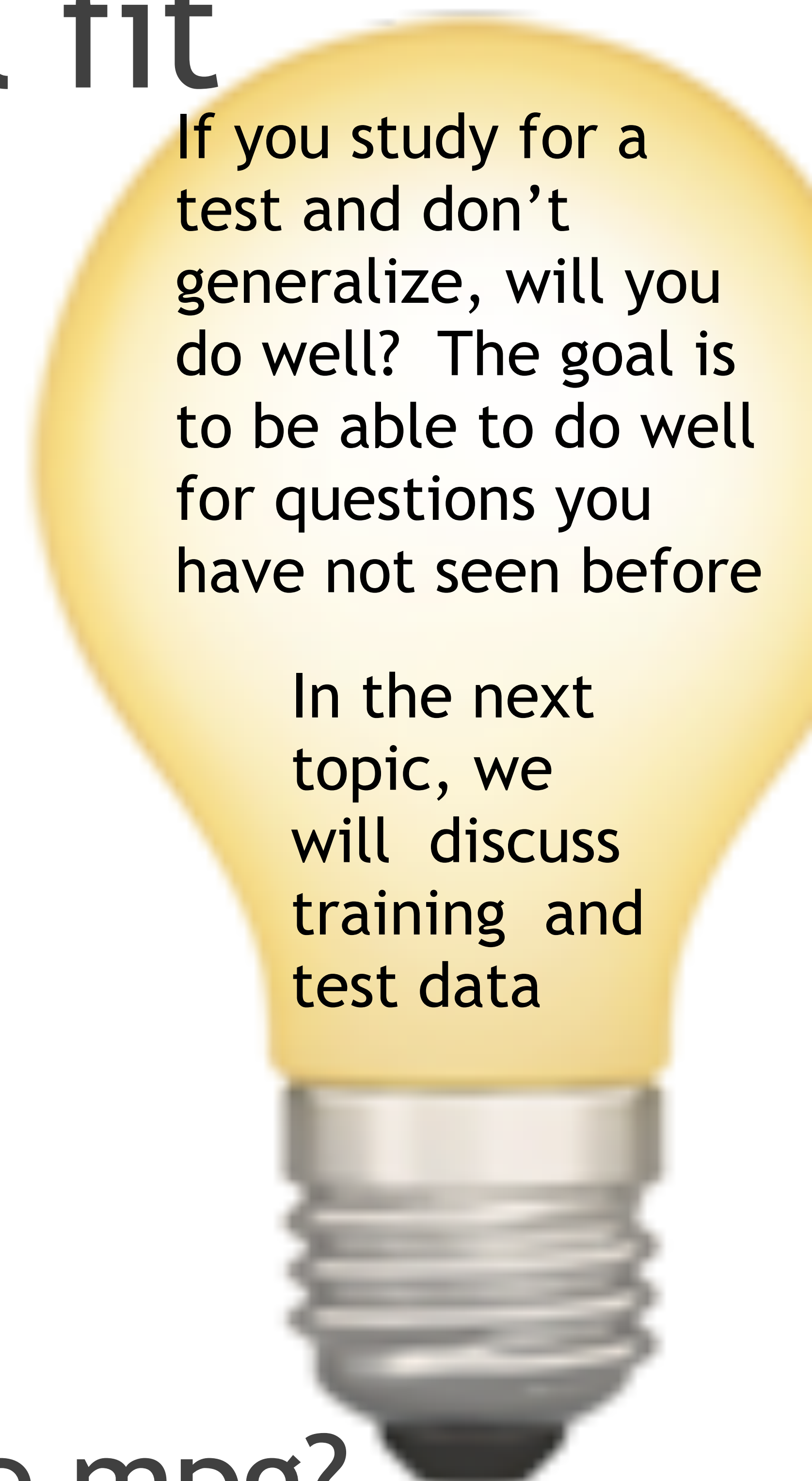
- Mean = 44.57
- Standard deviation = 18.09

(59 - 44.57)/18.09		0.79	Female		0
(48 - 44.57)/18.09		0.19	Male		1
(72 - 44.57)/18.09		1.51	Female		0
(24 - 44.57)/18.09		-1.14	Male		1
(50 - 44.57)/18.09		0.30	Male		1
(23 - 44.57)/18.09		-1.19	Male		1
(36 - 44.57)/18.09		-0.47	Female		0

Outline

- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Local Optimizer: "Batch" Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation.
- Extensions
- Removing features

How well does our model fit the training data?

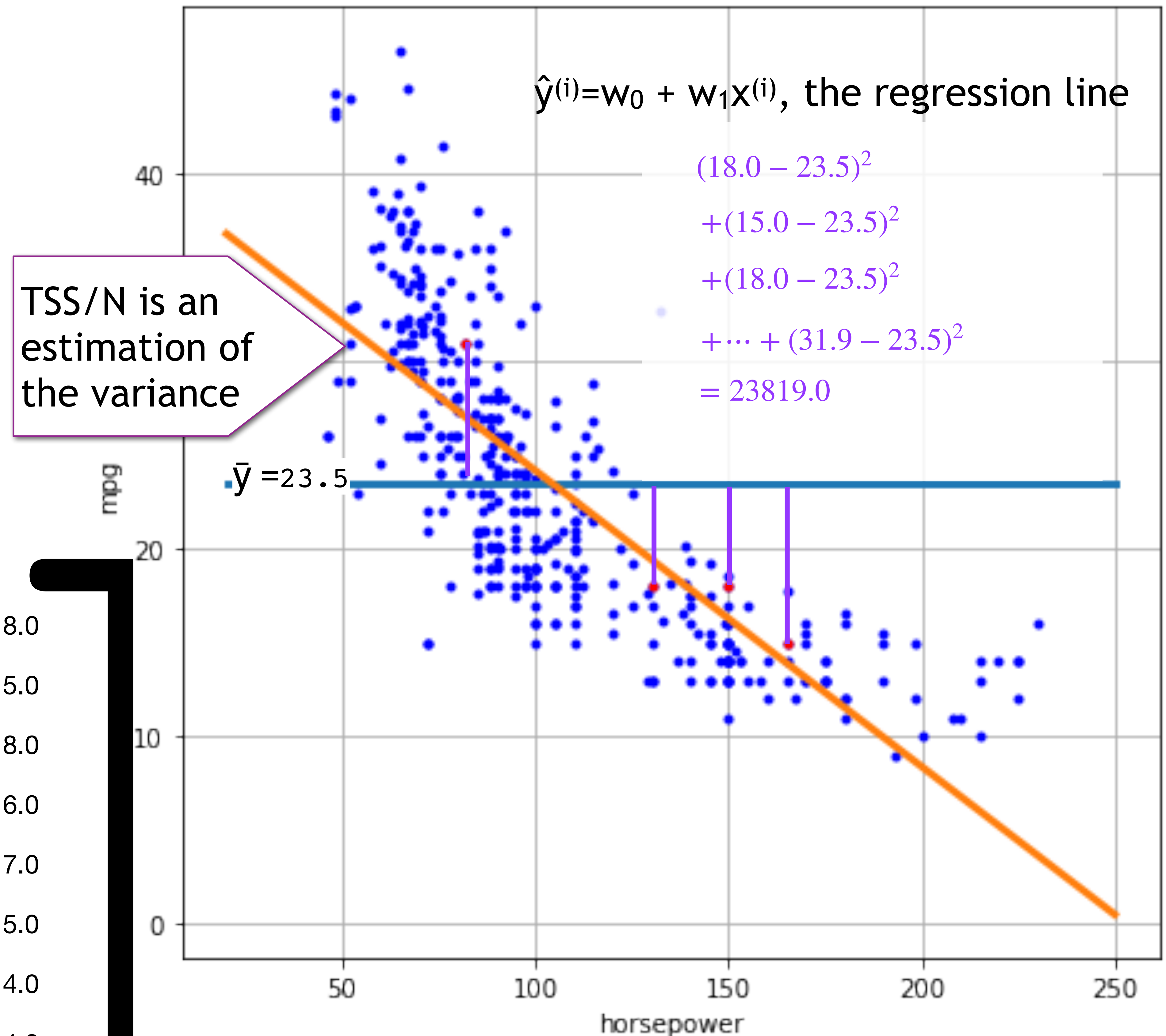


i.e. How well did you predict the mpg?

Variability of y

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

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

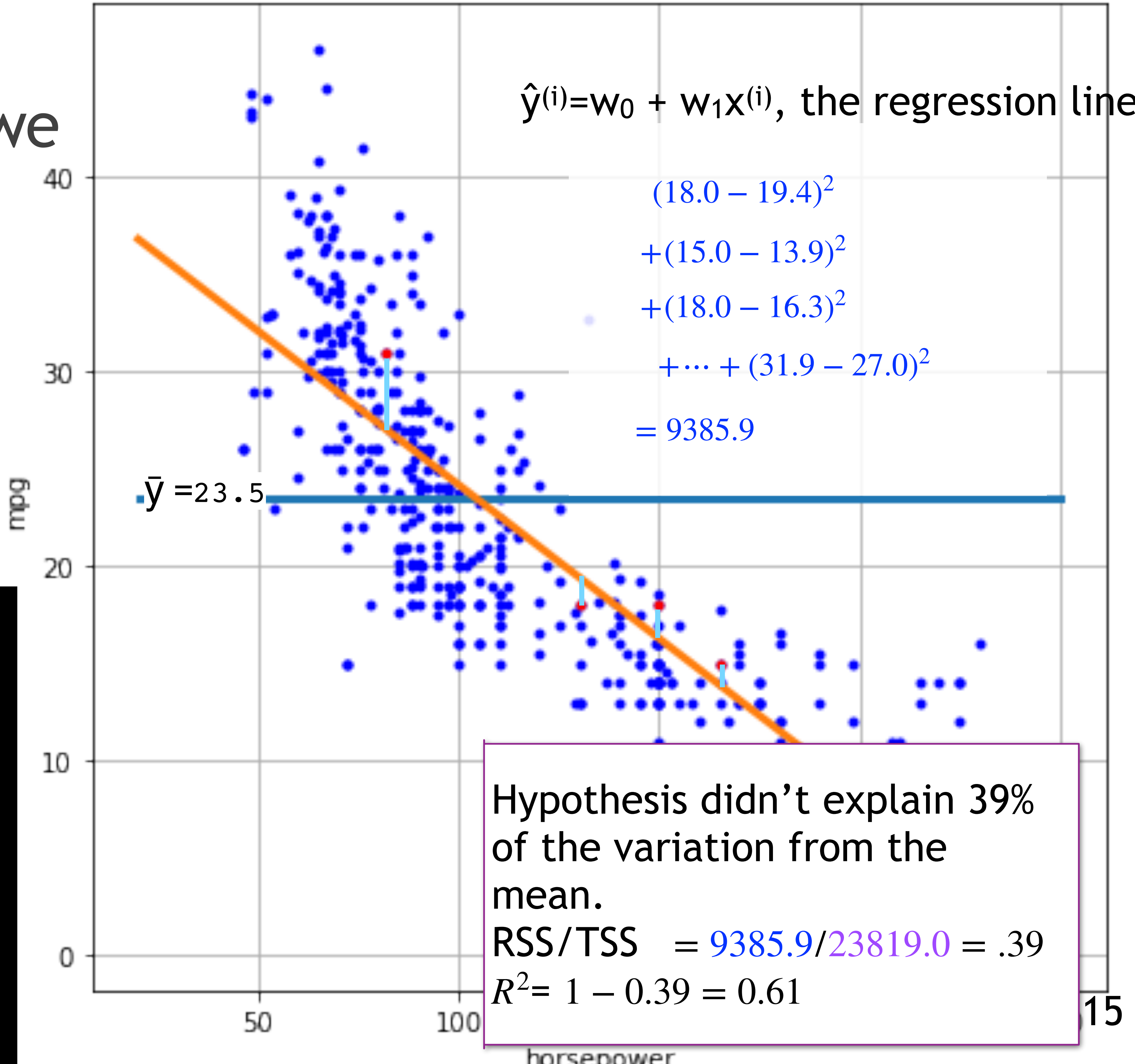


Variability of y when we take into account $\mathbf{x}$

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

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

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



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.  …]
```

```
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.]
  …           …
      ]           ]
```

```
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
```

Learn more about 1D and 2D numpy arrays: https://numpy.org/devdocs/user/absolute_beginners.html

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']
```

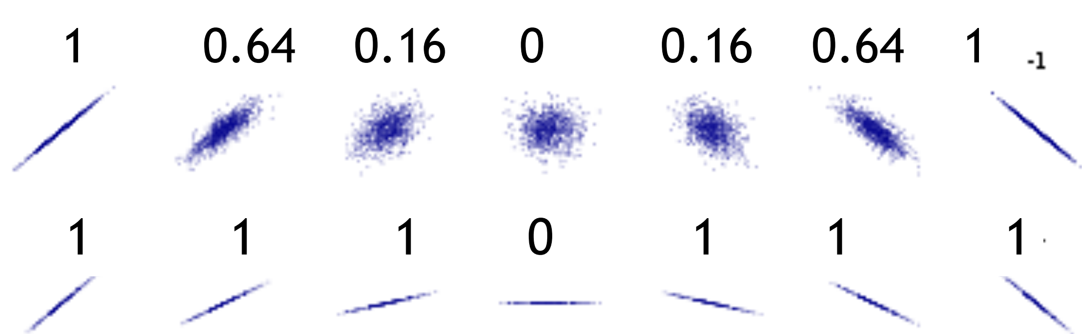
```
for k in range(2,len(names)-1,1):
    df1 = df[['mpg',names[k]]]
    df2 = df1.dropna()
    X=np.array(df2[names[k]])
    y=np.array(df2['mpg'])
    w0,w1,rsq=fit_linear(X,y) # returns w0, w1, rsq
    print(names[k],": ", np.around(rsq,2))
```

```
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

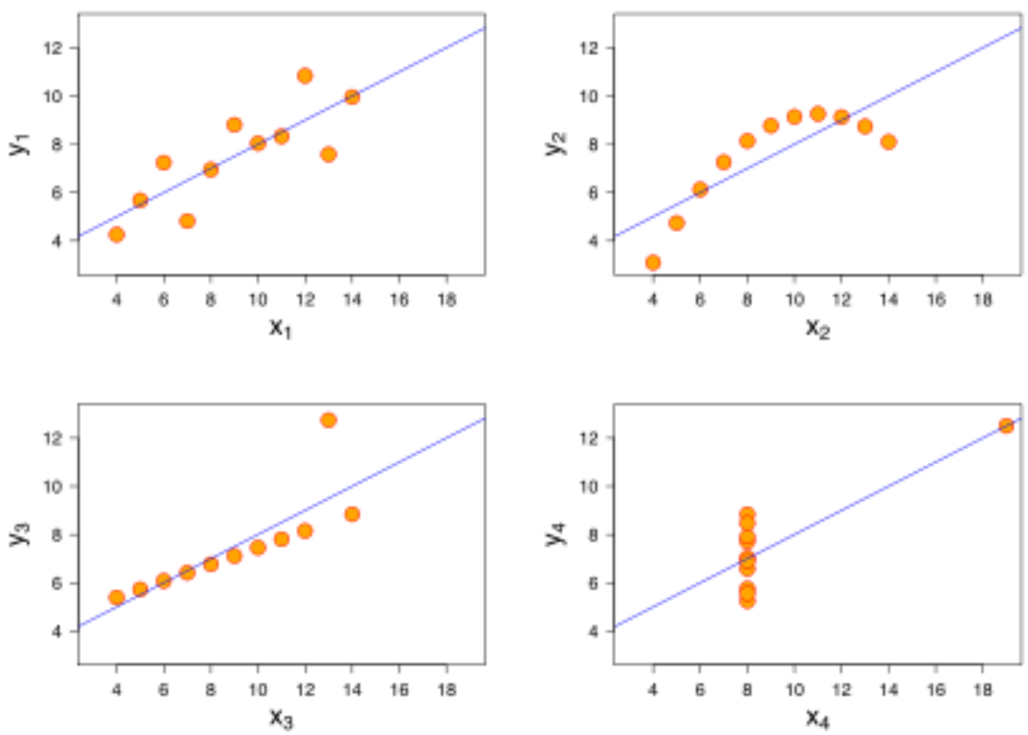
Visually seeing correlation

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



Very good fit

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 + 0.500x
  - But, errors and their reasons are different
- How would you describe these errors?
- All 4 graphs have a R^2 of 0.67

Outline

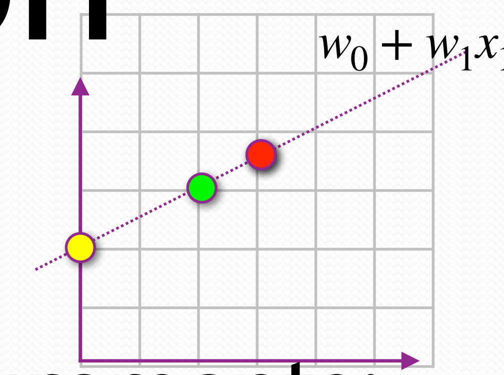
- Motivating Example: Predicting the mpg of a car
- Model: Linear Model
- Objective function: Least Squares Fit Problem
- Local Optimizer: “Batch” Gradient Descent
    - When $d = 1$ (simple linear regression)
    - What can go wrong?
    - General case, any $d > 0$ (multiple variable linear regression)
        - Example
        - Algorithm
        - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation.
- Extensions
- Removing features

Another way of looking at our cost function/algorithm

Another approach for determining which $\mathbf{w}$ is best

…it ends of reducing to the objective function we already chose

Probabilistic Interpretation



- **Errors due to** missing features or noise in our measurements:

$$y^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + \epsilon^{(i)}$$

$$(\mathbf{x}^{(1)}, y^{(1)}) = (2, 2 + \epsilon^{(1)})$$
$$(\mathbf{x}^{(2)}, y^{(2)}) = (3, 2.5 + \epsilon^{(2)})$$
$$(\mathbf{x}^{(3)}, y^{(3)}) = (0, 1 + \epsilon^{(3)})$$
$$(\mathbf{x}^{(4)}, y^{(4)}) = (3, 2.5 + \epsilon^{(4)})$$

- $p(\epsilon^{(1)}) \cdot p(\epsilon^{(2)}) \cdot p(\epsilon^{(3)}) \cdot p(\epsilon^{(4)})$

- What is $p(\epsilon^{(i)})$?

- One approach is to formally model $\epsilon^{(i)} \sim \mathcal{N}(0,1)$ independent of x (IID, independently and identically distributed)

$$\frac{1}{\sqrt{2\pi}} \frac{1}{e^{(0.5)^2/2}} \frac{1}{\sqrt{2\pi}} \frac{1}{e^{(-1)^2/2}} \frac{1}{\sqrt{2\pi}} \frac{1}{e^{(-0.5)^2/2}} \frac{1}{\sqrt{2\pi}} \frac{1}{e^{(0.75)^2/2}} \qquad \frac{1}{\sqrt{2\pi}} \frac{1}{e^{(1.5)^2/2}} \frac{1}{\sqrt{2\pi}} \frac{1}{e^{(0)^2/2}} \frac{1}{\sqrt{2\pi}} \frac{1}{e^{(0.5)^2/2}} \frac{1}{\sqrt{2\pi}} \frac{1}{e^{(1.75)^2/2}}$$

$\left(\epsilon^{(i)}\right)^2 = (y^{(i)} - \hat{y}^{(i)})^2$
$= \left(y^{(i)} - (w_0 + w_1\mathbf{x}^{(i)})\right)^2$

Maximum Likelihood estimation (cont.)

Which parameter **w** is best?

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

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

By performing a series of algebraic simplifications can be see to be the same as minimizing

$$\sum_{i=1}^{N} \left(y^{(i)} - (\mathbf{w}^T\mathbf{x}^{(i)})\right)^2$$

The measurements we have do not have infinite precision (i.e. $\mathbf{x} \in [\mathbf{x}_0 - \Delta/2, \mathbf{x}_0 + \Delta/2]$). Thus we can use the probability density function to compute the probability per unit area

$$\int_{\mathbf{x}_0-\Delta/2}^{\mathbf{x}_0+\Delta/2} f(\mathbf{x}_0; \theta) dx \approx f(\mathbf{x}_0; \theta)\Delta$$

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

Algebraic Simplifications

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

We are using ℓ for the log likelihood function

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

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

Just algebraic manipulation

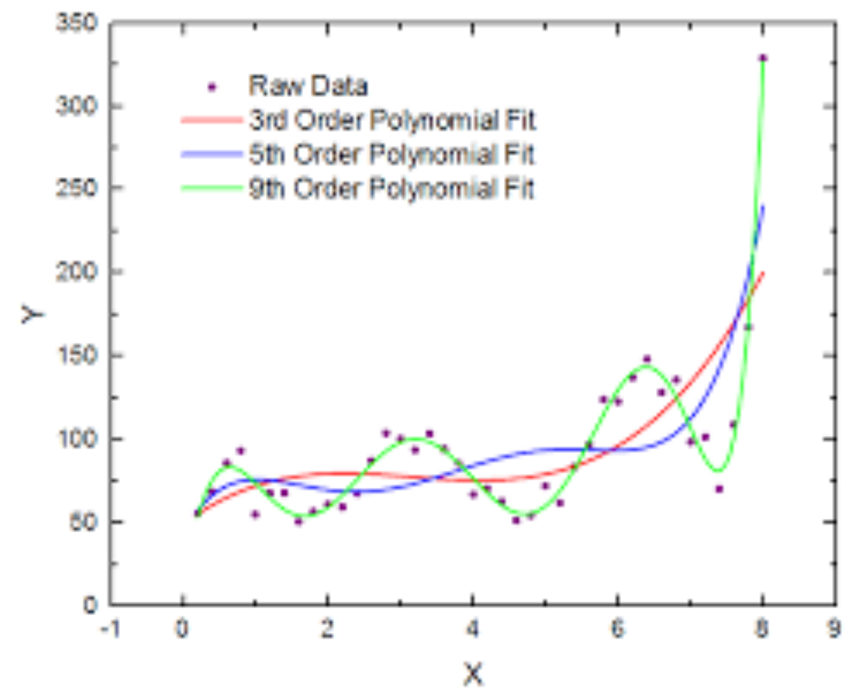
This … is the same as minimizing $\sum_{i=1}^{N} \left(y^{(i)} - (w_0 + w_1 \mathbf{x}^{(i)})\right)^2$

Outline

- ❑ Motivating Example: Predicting the mpg of a car
- ❑ Model: Linear Model
- ❑ Objective function: Least Squares Fit Problem
- ❑ Local Optimizer: “Batch” Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- ❑ Global Optimization: Normal Equations
- ❑ Feature transformation
- ❑ Assessing Goodness of Fit
- ❑ Objective function revisited: Probabilistic interpretation.
- ❑ Extensions
- ❑ Removing features

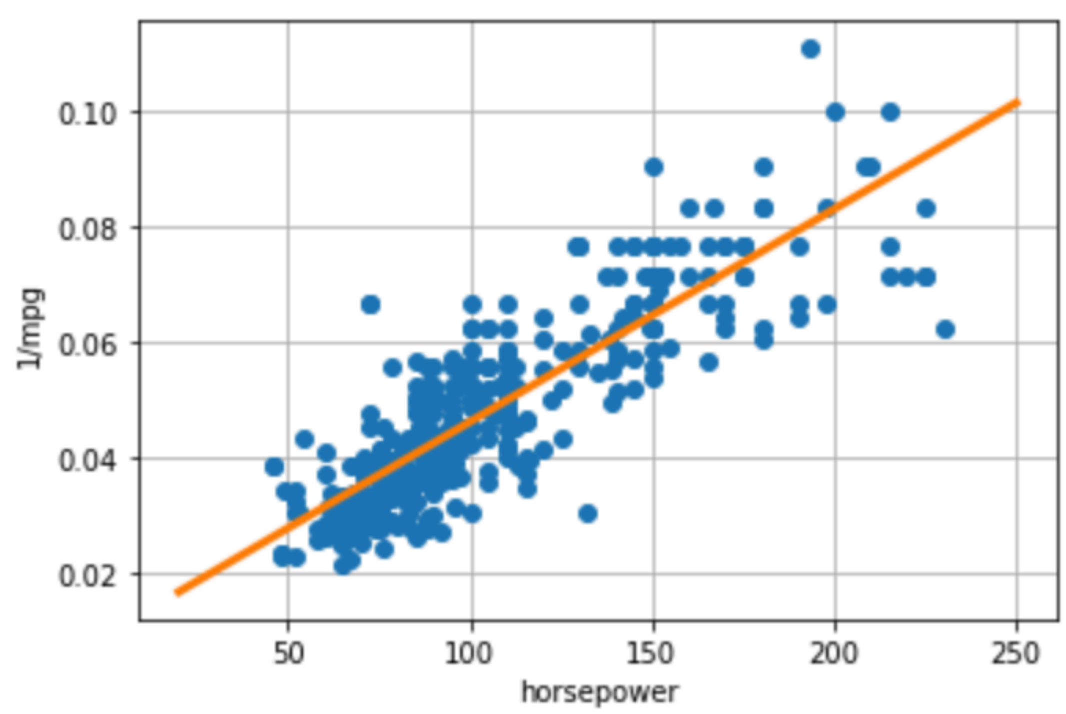
Polynomial Fitting

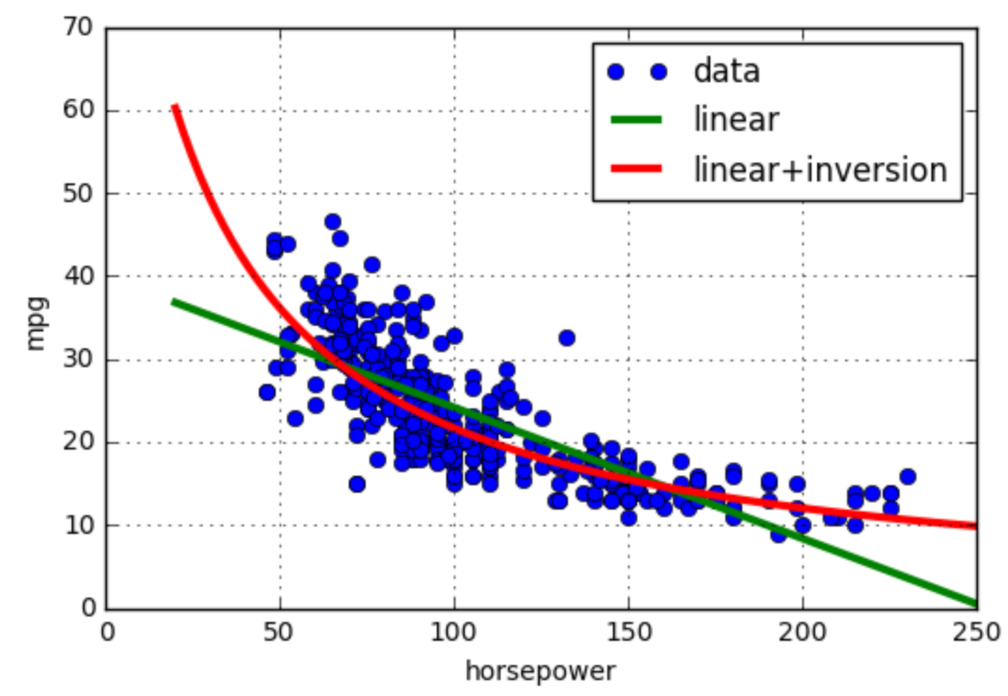
- Learn a polynomial model $\hat{y}^{(i)} = w_0 + w_1 \cdot x_1^{(i)} + w_2 \cdot \left(x_1^{(i)}\right)^2 + \cdots + w_d \cdot (x_1^{(i)})^d$
- Given data $(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \ldots, n$



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
- Local Optimizer: “Batch” Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- Global Optimization: Normal Equations
- Feature transformation
- Assessing Goodness of Fit
- Objective function revisited: Probabilistic interpretation.
- Extensions
- Removing features

In the next topic, we will talk about how to create **new** features from our current set of features.

How do we "prune" away non useful features?

We will have to train more than one model to find what works best

Which features to select

Here we are using k instead d for the number of features

- Subset selection: Identify a subset of the k predictors we believe are associated with the response.
- Then the least squares solution can be fit on the reduced set of variables
- We will use adjusted R^2 statistics to compare models when the number of features varies. (R^2 can increase when the number of features increases even if the features do not help predict the outcome!)

$$R^2_{adj} = 1 - \frac{RSS/(n-k-1)}{TSS/(n-1)}$$

Subset Selection

$$R^2_{adj} = 1 - \frac{RSS/(n-q-1)}{TSS/(n-1)}$$

- Subset selection: Identify a subset of the k predictors we believe are associated with the response. Then the least squares solution can be fit on the reduced set of variables

Algorithm 6.1 *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani

Forward Selection

$$R_{adj}^2 = 1 - \frac{RSS/(n-q-1)}{TSS/(n-1)}$$

- Forward selections starts with no predictors predictors in the model
- Then repeatedly adds the most significant predictor

Algorithm 6.2 *Forward stepwise selection*

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   (a) Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Backward Selection $R^2_{adj} = 1 - \frac{RSS/(n-q-1)}{TSS/(n-1)}$

- Backward elimination starts with all k predictors in the model
- Then repeatedly deletes the least significant predictor

Algorithm 6.3 *Backward stepwise selection*

---

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   (b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

There are many ways to clean up the data

WE WILL NOT FOCUS ON MANY OF THE ISSUES

THIS IS AN INTRODUCTION TO THE TOPIC

Outline

- ❑ Motivating Example: Predicting the mpg of a car
- ❑ Model: Linear Model
- ❑ Objective function: Least Squares Fit Problem
- ❑ Local Optimizer: "Batch" Gradient Descent
  - When $d = 1$ (simple linear regression)
  - What can go wrong?
  - General case, any $d > 0$ (multiple variable linear regression)
    - Example
    - Algorithm
    - Vectorized Algorithm
- ❑ Global Optimization: Normal Equations
- ❑ Feature transformation
- ❑ Assessing Goodness of Fit
- ❑ Objective function revisited: Probabilistic interpretation.
- ❑ Extensions
- ❑ Removing features
- Sklearn

Using Sklearns linear regression algorithm

Linear Regression using Scikit-learn (Sklearn)

- Can we predict diabetes patients' condition a year after taking 10 baseline measurements?
- The 10 baseline measurements are: age, sex, body mass index. average blood pressure, and 6 blood serum measurements
- Steps for using Scikit-Learn to make predictions
  1. Import packages
  2. Get the data and preprocess the data
  3. Create a model, fit model with data
  4. Evaluate how well your model performs
  5. Use model to predict

Step 1) Import packages

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

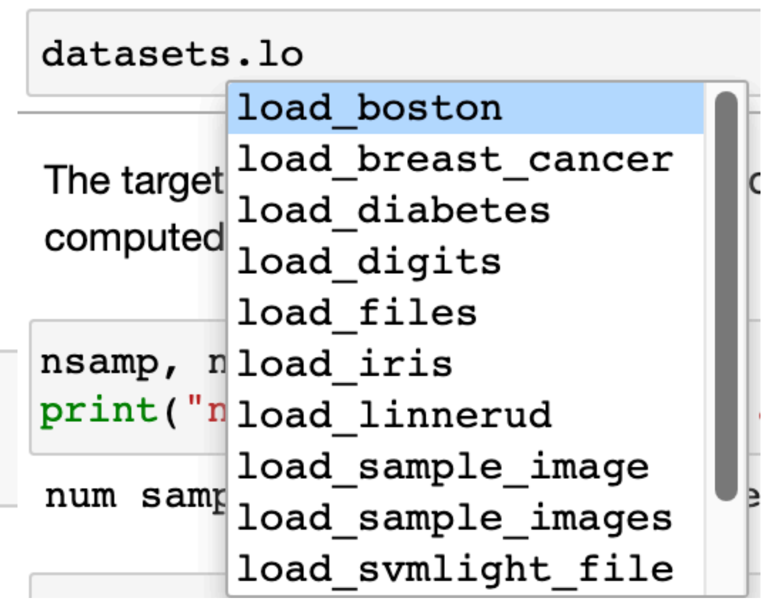
```
from sklearn import datasets, linear_model, preprocessing
```

Step 2) Loading the Data

```
# Load the diabetes dataset
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
```

```
nsamp, natt = X.shape
print("num samples={0:d}  num attributes={1:d}".format(nsamp,natt))
```

```
num samples=442  num attributes=10
```



- Sklearn package:
  - Many methods for machine learning
  - Datasets
  - Will use throughout this class
- Diabetes dataset is one example
- All code in demo

Step 2) dividing the dataset

```
ns_train = 300
ns_test = nsamp - ns_train
X_tr = X[:ns_train,:]      # Gets the first ns_train rows of X
y_tr = y[:ns_train]        # Gets the correspoinding rows of y
```

```
[[ 0.038  0.051  0.062  0.022 -0.044 -0.035 -0.043 -0.0
 [-0.002 -0.045 -0.051 -0.026 -0.008 -0.019  0.074 -0.0
 [ 0.085  0.051  0.044 -0.006 -0.046 -0.034 -0.032 -0.003  0.003 -0.026]
 [-0.089 -0.045 -0.012 -0.037  0.012  0.025 -0.036  0.034  0.023 -0.009]
 [ 0.005 -0.045 -0.036  0.022  0.004  0.016  0.008 -0.003 -0.032 -0.047]
 [-0.093 -0.045 -0.041 -0.019 -0.069 -0.079  0.041 -0.076 -0.041 -0.096]
 [-0.045  0.051 -0.047 -0.016 -0.04  -0.025  0.001 -0.039 -0.063 -0.038]
 [ 0.064  0.051 -0.002  0.067  0.091  0.109  0.023  0.018 -0.036  0.003]
 [ 0.042  0.051  0.062 -0.04  -0.014  0.006 -0.029 -0.003 -0.015  0.011]
 [-0.071 -0.045  0.039 -0.033 -0.013 -0.035 -0.025 -0.003  0.068 -0.014]
 …
]
```

(300, 10)

```
[151.  75. 141. 206. 135.  97. 138.  63. 110. 310., …,  83]
```

(300,)

We should randomly permute the data first!

- Divide data into two portions:
  - Training data: First 300 samples
  - Test data: Remaining 142 samples
- Train model on training data.
- Test model (i.e. measure RSS) on test data
- Reason for splitting data will be discussed in the next topic.

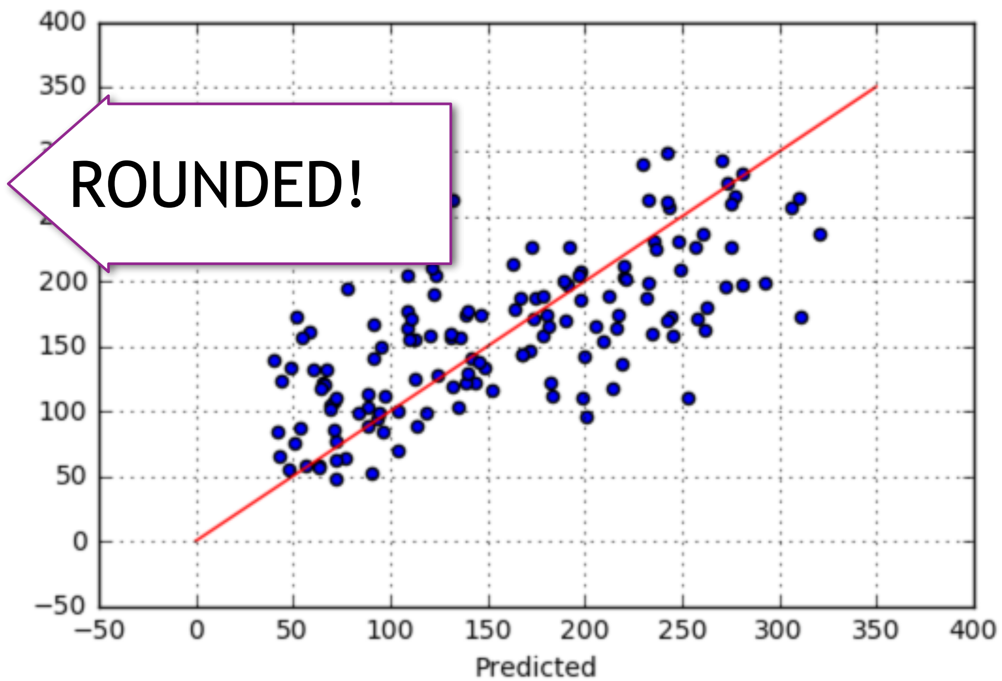
Step 3) Calling the sklearn method

- Construct a linear regression object
- Run it on the training data
- Find the parameters of the model

```
regr = linear_model.LinearRegression()
regr.fit(X_tr,y_tr)
```

```
print('The intercept w0 = ', regr.intercept_)
print('The coefficients w[1..d]=', regr.coef_)
```

```
The intercept w0 =  152.35
The coefficients w[1..d]= [-16.58 -254.67 560.99
278.92 -393.41 97.05 -19. 169.46 632.95 114.22]
```



$$\hat{y}^{(i)} = 152.35 - 16.58x_1^{(i)} - 254.67x_2^{(i)} + 560.99x_3^{(i)} + 278.92x_4^{(i)} - 393.41x_4^{(i)} + 97.05x_6^{(i)} - 19x_7^{(i)} + 169.46x_8^{(i)} + 632.95x_9^{(i)} + 114.22x_{10}^{(i)}$$

Learn more at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
https://machine-learning-apps.github.io/hands-on-ml2/04_training_linear_models

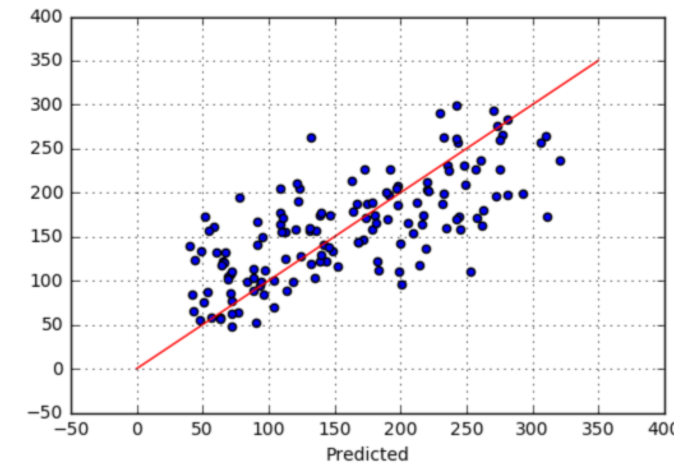
$$\hat{y}^{(i)} = 152.35 - 16.58x_1^{(i)} - 254.67x_2^{(i)} + 560.99x_3^{(i)} + 278.92x_4^{(i)} - 393.41x_4^{(i)} + 97.05x_6^{(i)} - 19x_7^{(i)} + 169.46x_8^{(i)} + 632.95x_9^{(i)} + 114.22x_{10}^{(i)}$$

Step 4 & 5) Evaluating the model and predicting

- ❑Predict values on the training data
- ❑Compute the R^2 score
- ❑Predict values on the test data

```
y_tr_pred = regr.predict(X_tr)
RSS = np.sum((y_tr_pred-y_tr)**2)
TSS = np.sum((y_tr - np.mean(y_tr))**2)
print("RSS = {0:f}".format(RSS))
print("Ein = {0:f}".format(RSS/ns_train))
print("RMSE = {0:f}".format(np.sqrt(RSS/ns_train)))
print("R^2 = {0:f}".format(1-RSS/TSS))
```

```
RSS = 876900.060150
Ein = 2923.000201
RMSE = 54.064778
R^2 = 0.514719
```

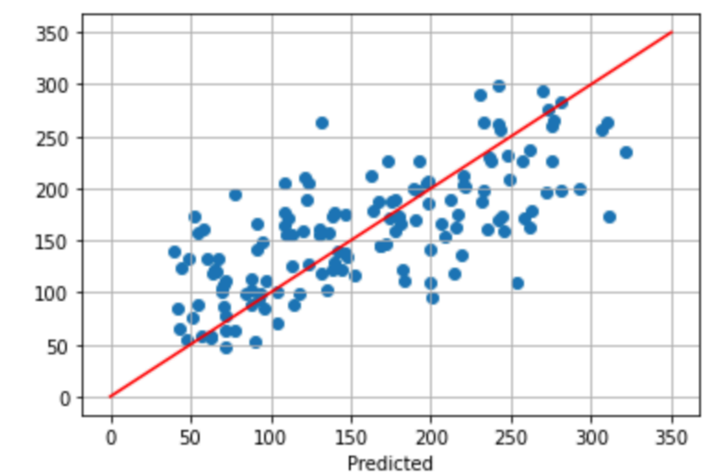



```
X_test = X[ns_train:,:]
y_test = y[ns_train:]
y_test_pred = regr.predict(X_test)

RSS = np.sum((y_test_pred-y_test)**2)

print("RSS = {0:f}".format(RSS))
print("MSE = {0:f}".format(RSS/ns_test))
print("RMSE = {0:f}".format(np.sqrt(RSS/ns_test)))
```

```
RSS = 396828.800059
MSE = 2794.569015
RMSE = 52.863683
```



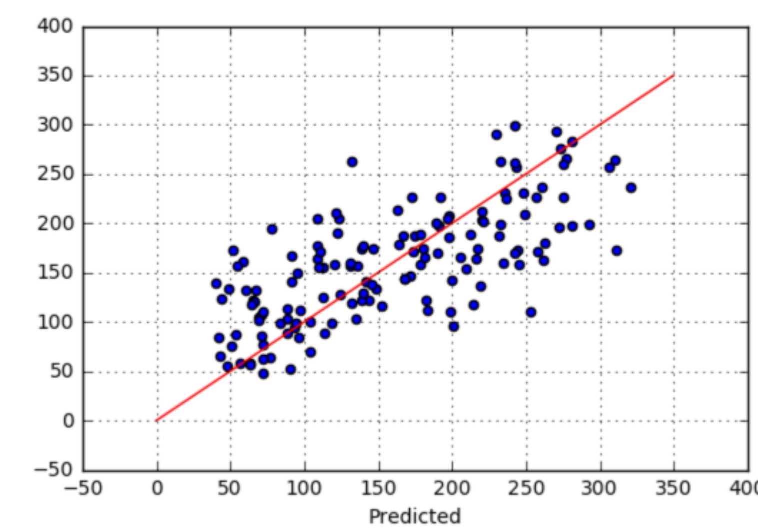
$$\hat{y}^{(i)} = 152.35 - 16.58x_1^{(i)} - 254.67x_2^{(i)} + 560.99x_3^{(i)} + 278.92x_4^{(i)} - 393.41x_4^{(i)} + 97.05x_6^{(i)} - 19x_7^{(i)} + 169.46x_8^{(i)} + 632.95x_9^{(i)} + 114.22x_{10}^{(i)}$$

Step 4) Evaluating the model

❑Compute the R^2 score

```python
# We can also use the built in score function
regr.score(X_tr,y_tr)
```

R^2 = 0.514719



$$\hat{y}^{(i)} = 152.35 - 16.58x_1^{(i)} - 254.67x_2^{(i)} + 560.99x_3^{(i)} + 278.92x_4^{(i)} - 393.41x_4^{(i)} + 97.05x_6^{(i)} - 19x_7^{(i)} + 169.46x_8^{(i)} + 632.95x_9^{(i)} + 114.22x_{10}^{(i)}$$

Step 5) predicting

[[36 2 22 90 160 99.6 50 3 3.9512 82]]

Scale any **new** data using the mean and std of the training dataset

y_hat = regr.predict(x)

[[-0.046 0.051 -0.047 -1.600e-02 -0.040 -2.480e-02 -0.001 -0.039 -0.063 -0.038]]

71.81