INFSCI 2915: Machine Learning

Linear Regression

Mai Abdelhakim

School of Computing and Information

610 IS Building

Spring 2018

Previous Meeting

- Performance tradeoffs
  - Bias-variance tradeoff
  - Overfitting, underfitting
- K Nearest Neighbor (KNN) Classification algorithm

Exercise Answers

A) Classify the Iris species with KNN approach using the **first two feature only** (X_train[:, : 2], X_test[:, : 2]), and check the accuracy as **K changes**. Let K takes the values [1, 5, 10, 15]. No need to scale features.

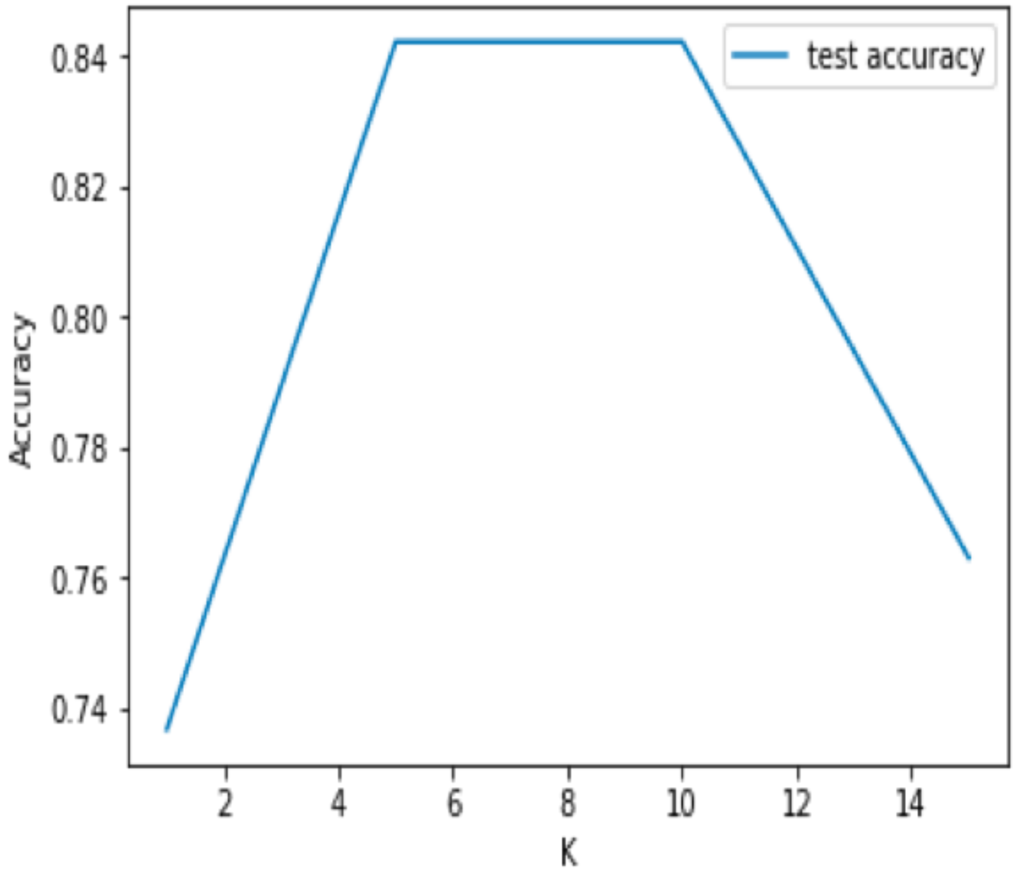
In the code, use **random_state=100** in **train_test_split**

- Plot the accuracy and comment on your result

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
test_accuracy=[ ]
neighbor_setting =[1,5,10,15]

for N_neighbor in neighbor_setting:
    knn=KNeighborsClassifier(n_neighbors=N_neighbor)
    knn.fit(X_train[:, :2], Y_train)
    test_accuracy.append(knn.score(X_test[:, :2],Y_test))

plt.plot(neighbor_setting,test_accuracy,label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("K")
plt.legend()
print (test_accuracy)
plt.legend()
print (test_accuracy)
```



<u>Comment</u> K=1 results in low accuracy due to overfitting.
Increasing K improves the results up to certain point, after which the performance degrades again due to underfitting

This unit

- Linear regression model
- Coefficients estimation
- Inference: Measure association between target and features
  - T-statistics
  - P-value
- Linear regression with multiple features
- Feature selection
- Performance metrics

Linear Regression

- Regression: **Quantitative response prediction**, **supervised** learning
- Linear regression: assumes linear relation between predictors/features $(X_1, X_2, \ldots X_p)$ and the response Y
  - **Parametric Model**

β_i : Coefficients or parameters

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$$

- Easy to **interpret** the parameters
  - β_1 is the average increase in Y when there is one unit increase in X_1 and all other features are constant

- Linear regression is extremely useful both conceptually and practically.

Linear Regression

- **Population line/Actual functions:** Y= β_0+$\beta_1 X_1$+$\beta_2 X_2$+…+$\beta_p X_p + \epsilon$

- **Estimated line:** $\hat{Y} = \hat{\beta}_0$+$\hat{\beta}_1 X_1$+ ..+ $\hat{\beta}_p X_p$

- Given training data, what are the **coefficients** that fit the available training data well
  - We have n training observations: (x_1,y_1), (x_2,y_2), …. (x_n,y_n)
  - How to estimate $\beta_i' s$?

- One method to estimate coefficients is the **least square method**

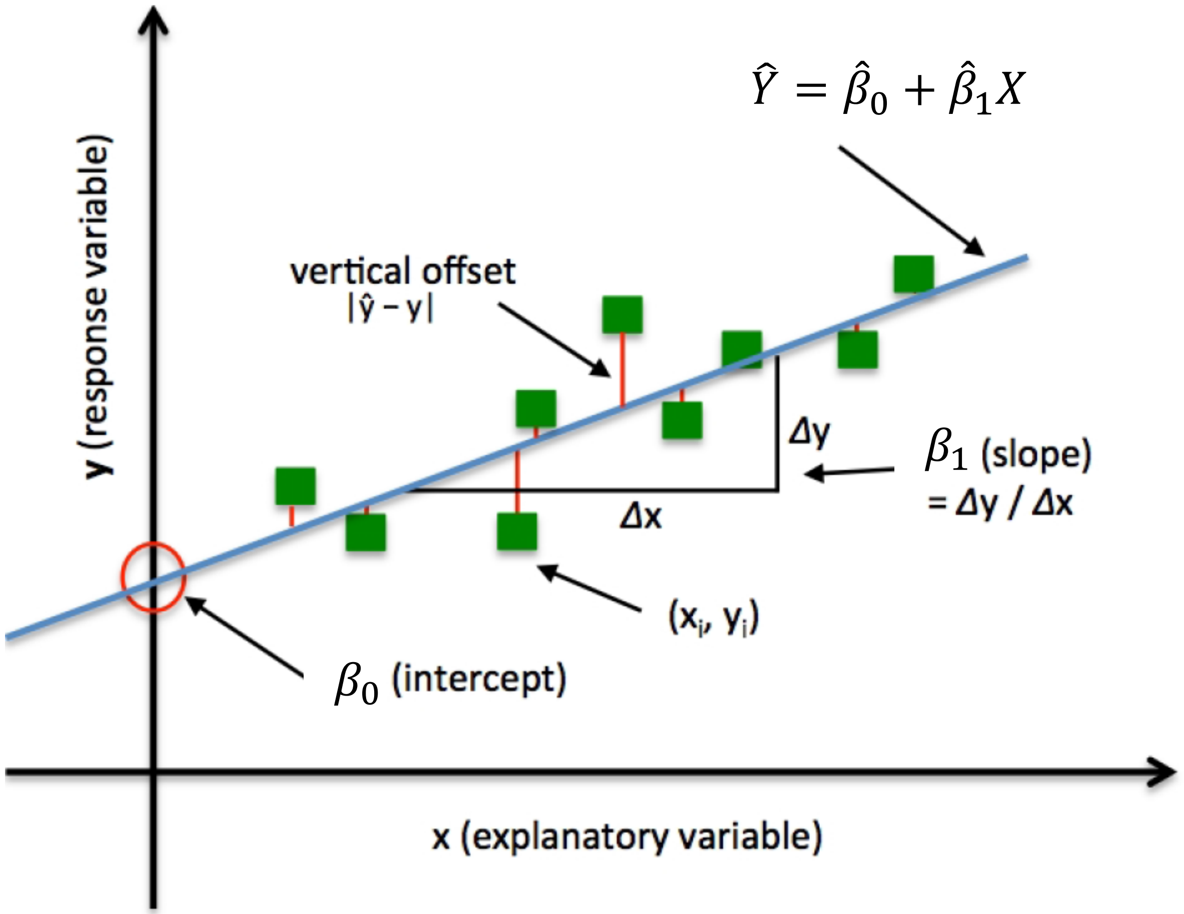
Simple Linear Regression – Single Feature

Linear Regression with one feature

$$Y \approx \beta_0 + \beta_1 X$$

- β_0 is called the intercept or bias term
- β_1 is the slope
- The coefficients (parameters) β_0 and β_1 are unknown
- Before we make predictions, coefficients must be estimated

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$



Ref: Raschka, Python Machine Learning

Example

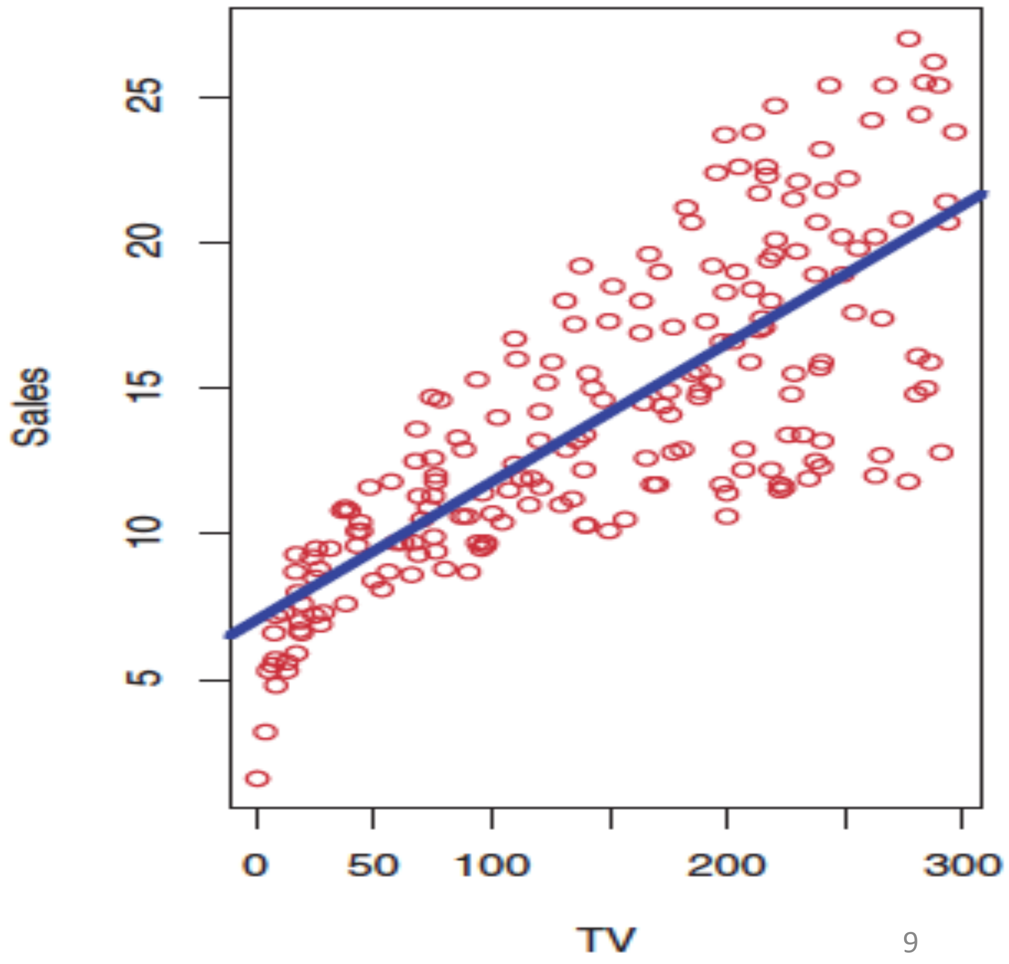
- In the Advertising example, this data set consists of the TV advertising budget and product sales in n = 200 different markets

- Linear regression model:

$$\text{sales} \approx \beta_0 + \beta_1 \text{ x TV}$$

- How would the fit look like if
  - $\beta_1 = 0$?
  - $\beta_0 = 0$?

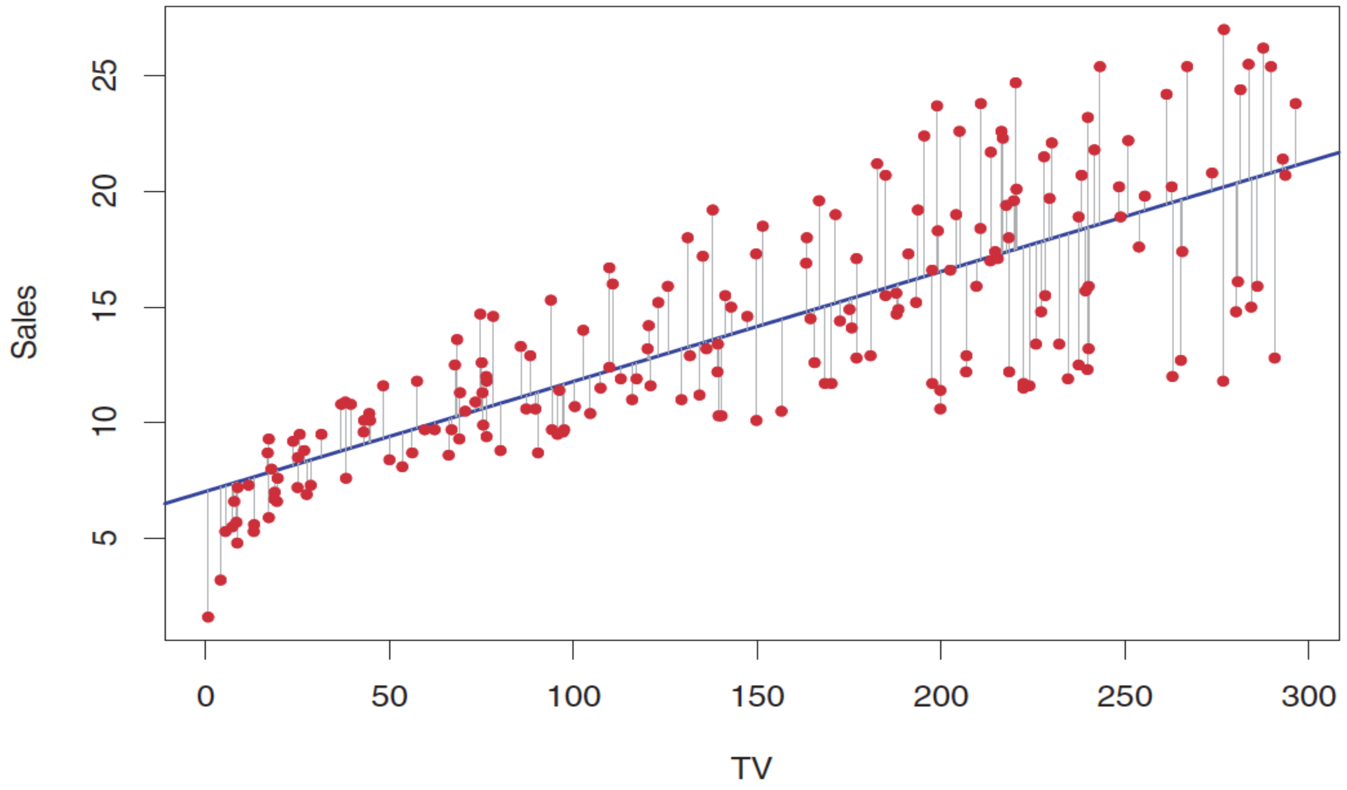
- How to get β_0 and β_1?



Least Square Method

- Error term of each sample in the training data is:

$$e_i = \hat{y}_i - y_i$$



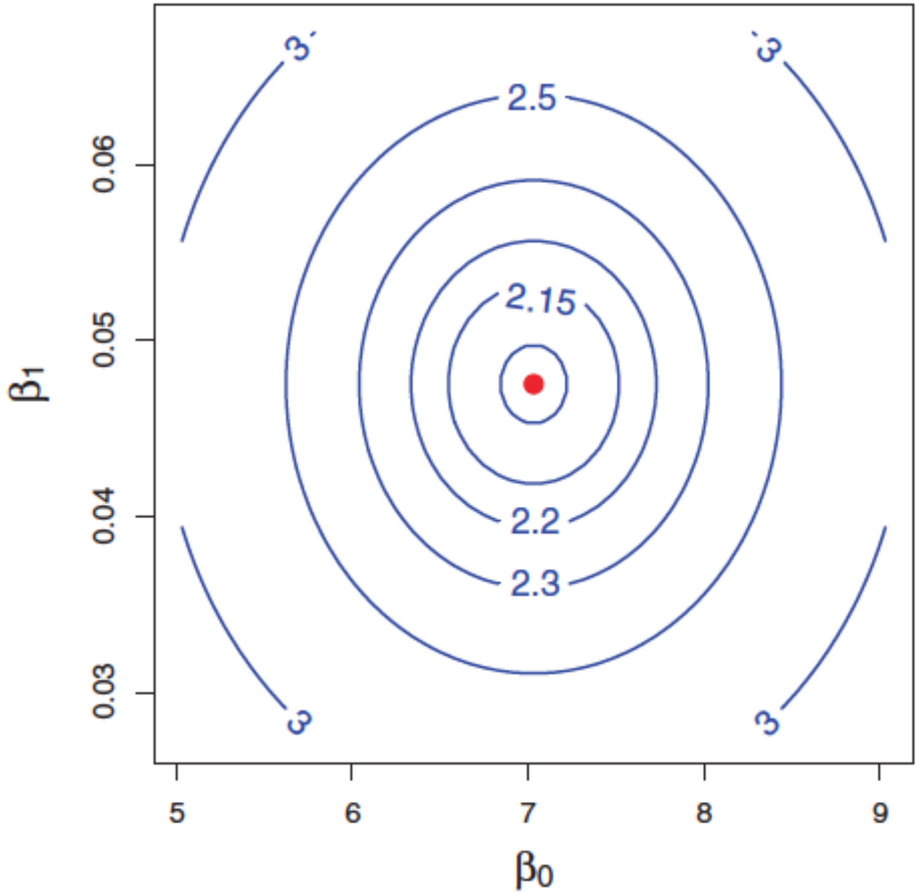
Training Set

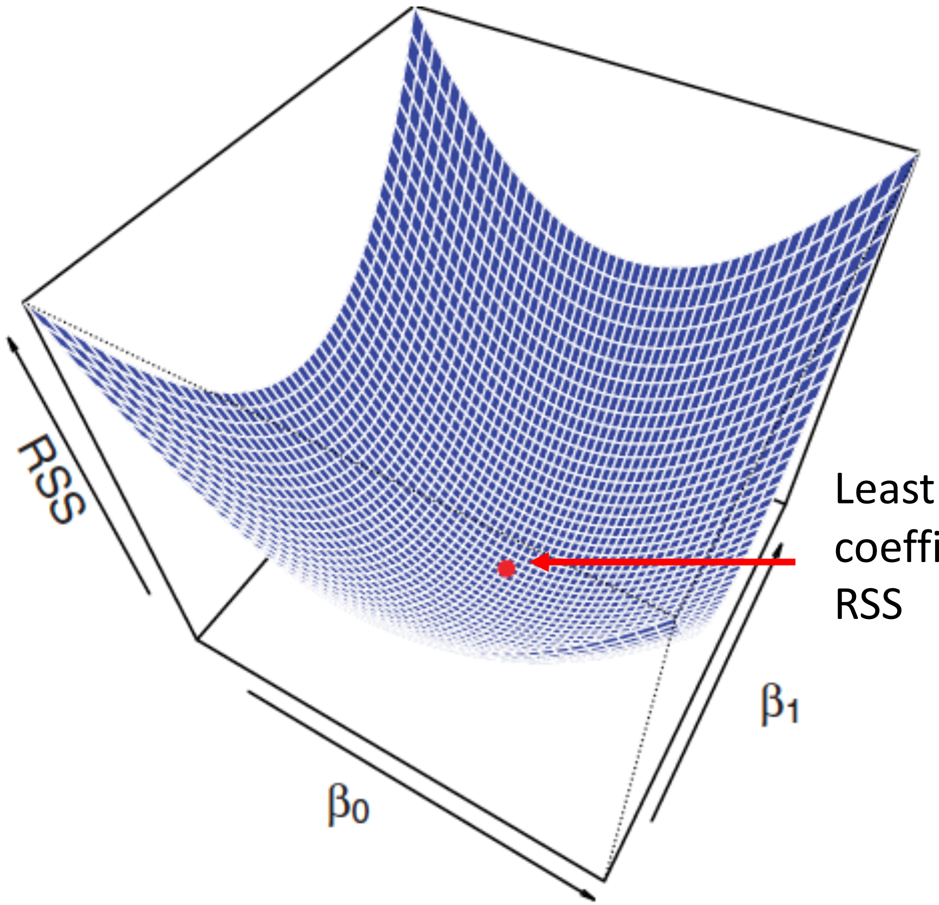
TV budget	Sales
x_1	y_1
x_2	y_2
x_N	y_N

Least Square Method

- RSS: residual sum of squares:

$$RSS = e_1^2 + \ldots + e_n^2 = \sum_{i=1}^{n}[\hat{y}_i - y_i]^2 = \sum_{i=1}^{n}[\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i]^2$$





Least square estimate are coefficients at minimum RSS

this point fit the model very well

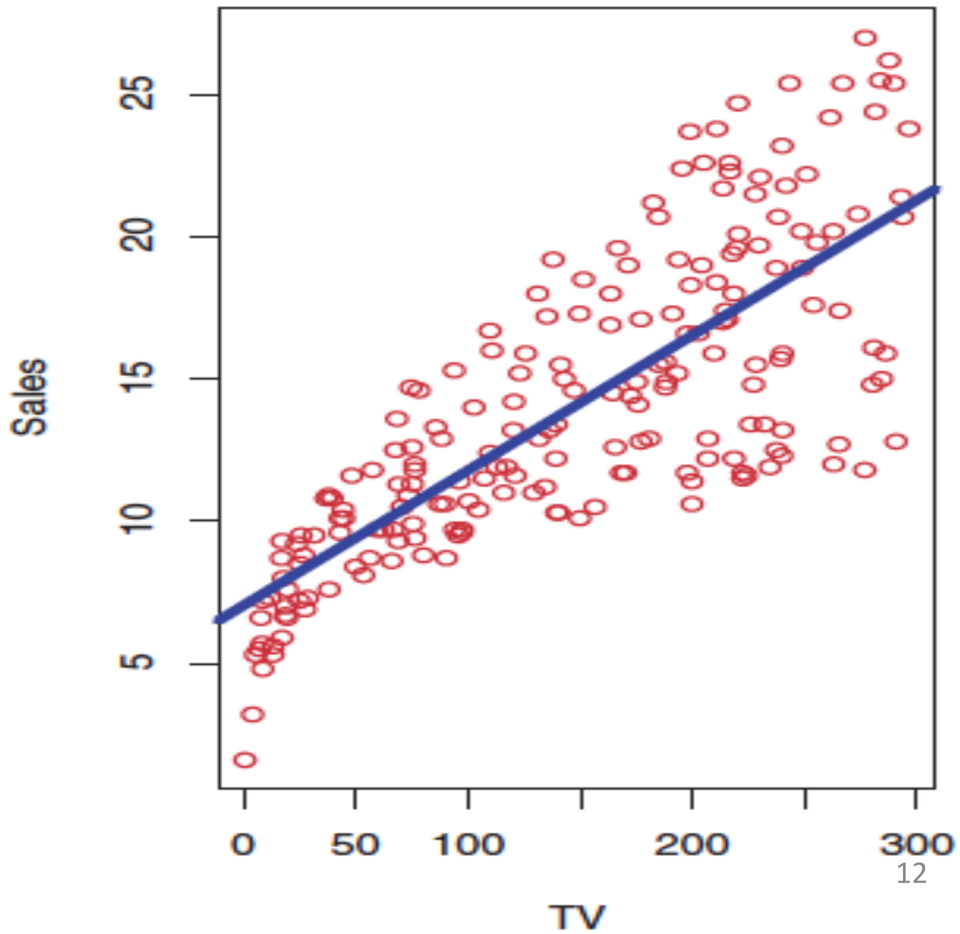
Least Square Method

- Define cost function as mean square error (MSE):

$$J(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n}\sum_{i=1}^{n}[\hat{y}_i - y_i]^2 = \frac{1}{n}\sum_{i=1}^{n}[\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i]^2 = \frac{1}{n} RSS$$

- **Least square method finds the coefficients that minimizes $\boldsymbol{J(\beta_0, \beta_1)}$**
  - This also **minimizes RSS**

$$minimize_{\beta_0, \beta_1}\ J(\beta_0, \beta_1)$$



Least Square Method

- Model: $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$

- Parameters: estimate β_0 and β_1

- Optimization/cost function:
  - $J(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{n} \sum_{i=1}^{n} [\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i]^2 = \frac{1}{n}$ *RSS*
  - *Sometimes defined as* $J(\hat{\beta}_0, \hat{\beta}_1) = \frac{1}{2n} \sum_{i=1}^{n} [\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i]^2 = \frac{1}{2n}$ *RSS*
  - The cost function can also be defined to be just equal *RSS:* $J(\hat{\beta}_0, \hat{\beta}_1) = RSS$

- Goal: fine coefficients that minimize the cost functions

$$minimize_{\beta_0, \beta_1} \; J(\beta_0, \beta_1)$$

Methods for Finding the Optimal Coefficients

- Get derivative of the cost function then set to zero –> closed form method
- Gradient descent

Ordinary Least Square (OLS) – Closed Form Solution

- We get the partial derivative of cost function (or RSS) with respect to β_i, then equate to zero to get the coefficients

$$\frac{\partial RSS}{\partial \beta_0} = 0, \frac{\partial RSS}{\partial \beta_1} = 0$$

- Using some calculus, show that β_0 and β_1 that minimize the RSS are

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} y_i x_i - \bar{y}\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^2 - \bar{x}\sum_{i=1}^{n} x_i} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where $\bar{y} = \frac{\sum_{i=1}^{y} y_i}{n}$ $\quad \bar{x} = \frac{\sum_{i=1}^{y} x_i}{n}$

R^2 square value 0-->1
R^2=(TSS-RSS)/TSS
TSS=(y平均-yi)from 1 to n
RSS=(y^-yi) from 1 to n

get by train set,
use to test dataset

Least Square Method
Optimization – Gradient Descent

- Another method to find optimal parameters
- Used when finding closed form solution is difficult or computationally more expensive
- Iterative method to find the optimal values of coefficients
  - Start with some values for the parameters (β_0 and β_1)
  - Calculate the gradient and then update all parameters simultaneously

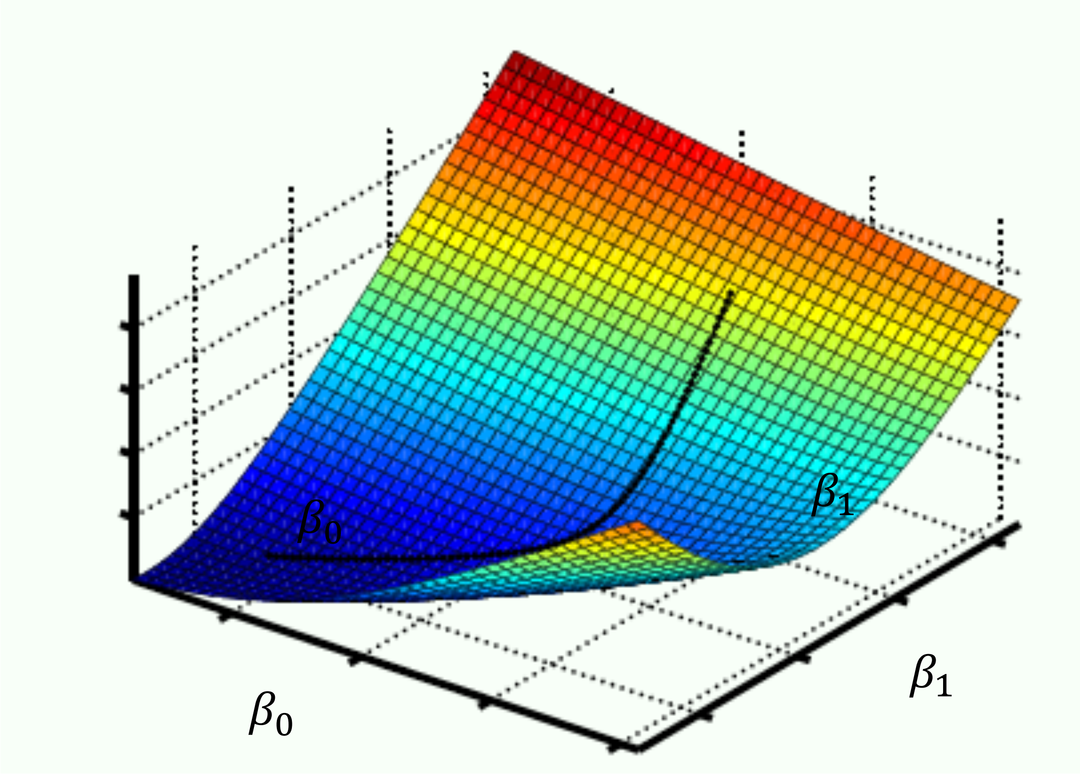
$$\beta_i := \beta_i - \alpha \frac{\partial}{\partial \beta_i} \mathrm{J}(\beta), \qquad \alpha \text{ is the learning rate}$$

  - Keep changing the parameters to reduce the objective function $J\left(\hat{\beta}_0, \hat{\beta}_1\right)$ until the minimum value of the objective function is obtained or a predefined stopping condition is met.

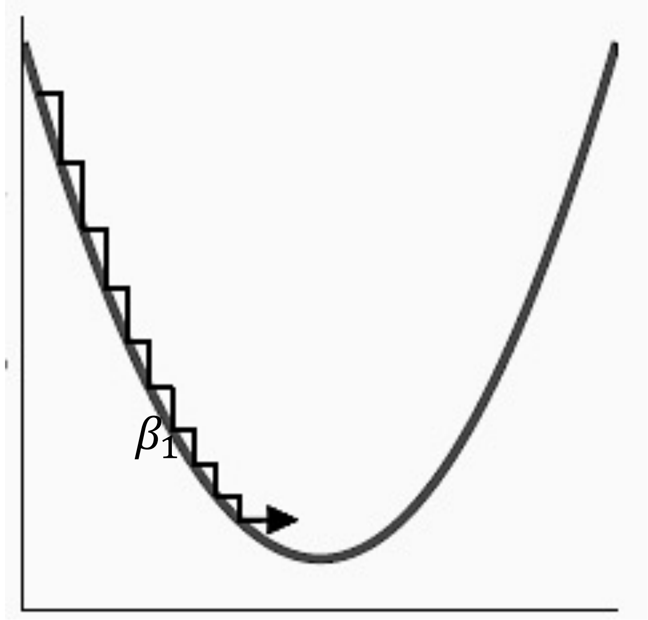
Impact of Learning Rate

- With a small learning rate, e.g. α=0.1, convergence is slow

RSS



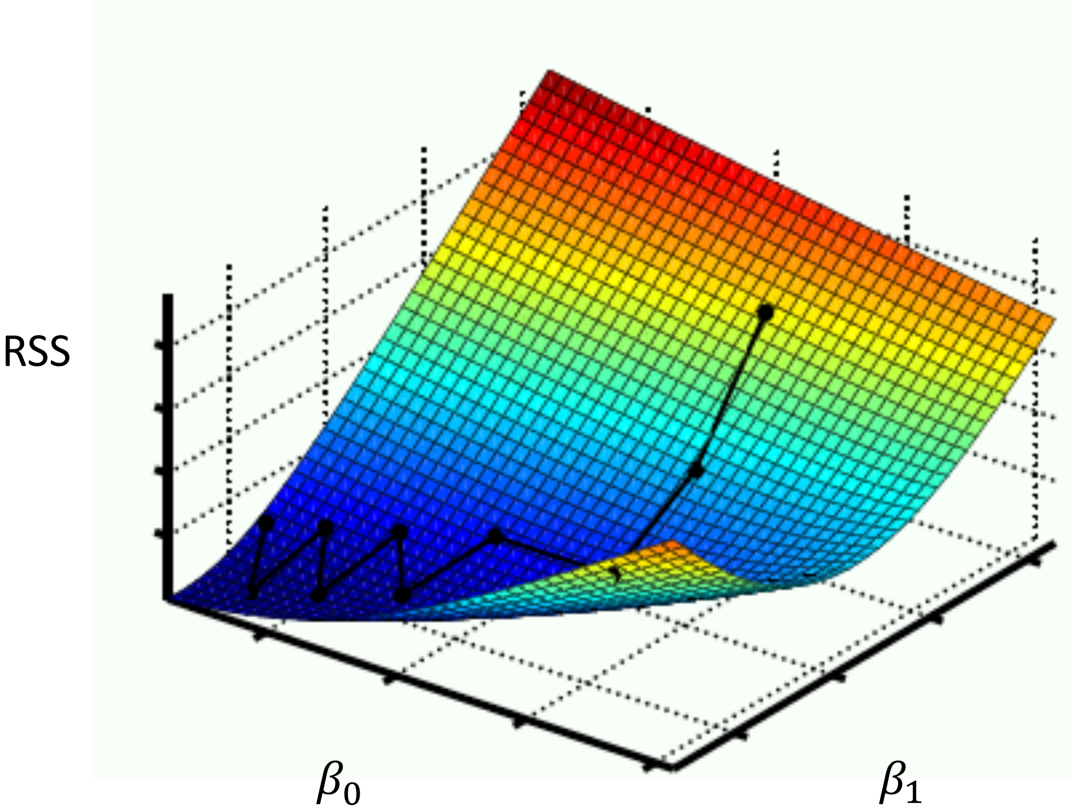
RSS

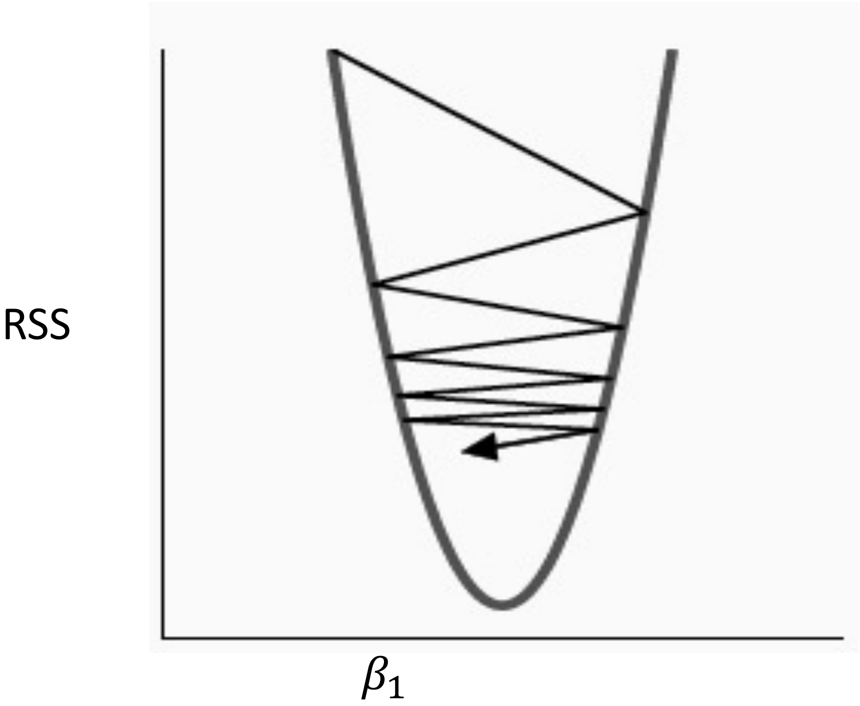


β_1

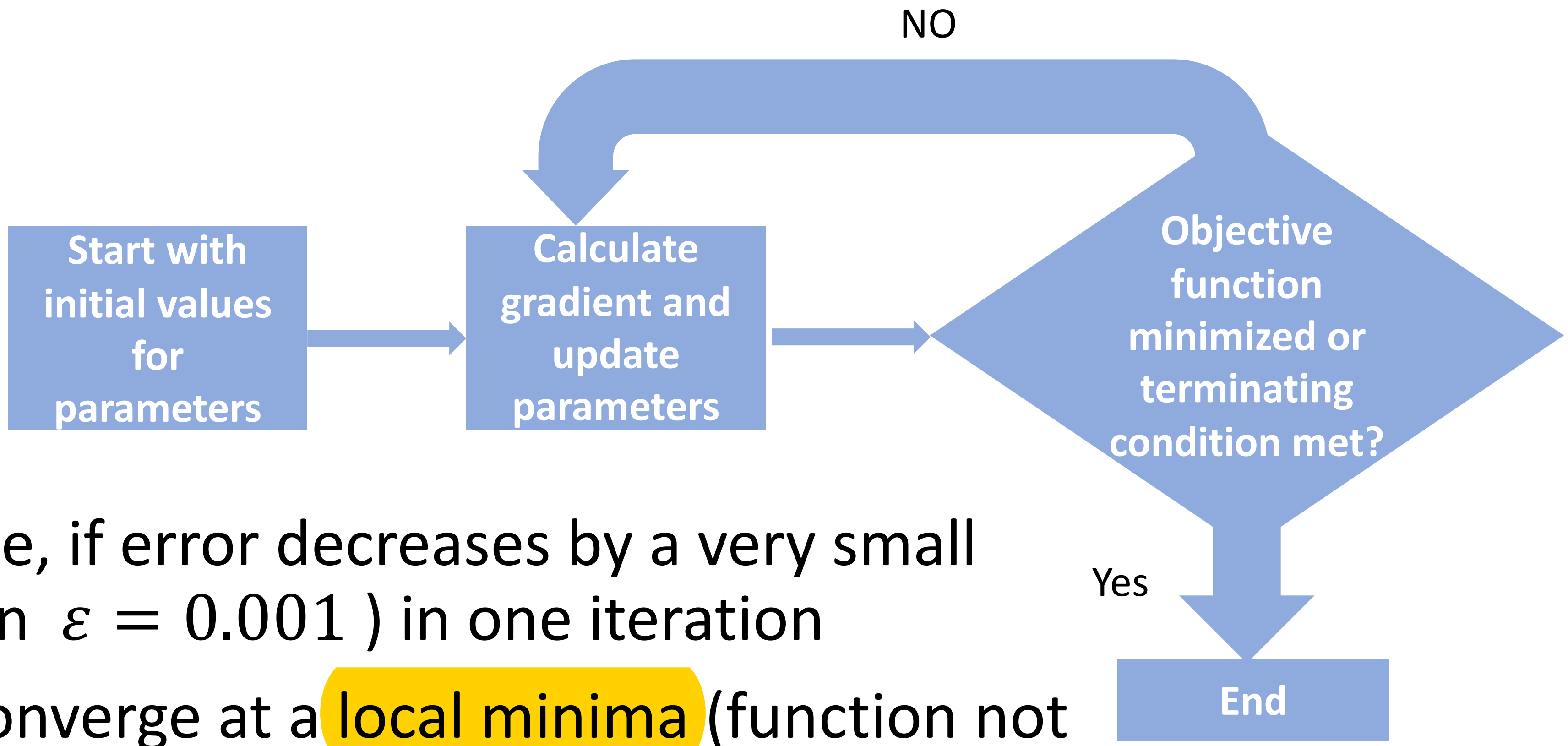
Impact of Learning Rate

- Large learning rate, e.g. α=2, the algorithm may not converge





Optimization – Gradient Descent



- Converge: for example, if error decreases by a very small amount (e.g. less than $\varepsilon = 0.001$) in one iteration
- Disadvantage: may converge at a local minima (function not convex)
- Other variant – e.g. stochastic gradient descent (update weight using one training example at a time)

Linear Regression in Python – Model

- Let's model the actual line as : $y=2+3X+\epsilon$, with ϵ be normal error with zero mean and unit variance

```
import numpy as np
beta0=2; beta1=3
stdDeviation=1; ErrMean=0; numberSamples=100

error = np.random.normal(loc=ErrMean, scale=stdDeviation, size=numberSamples)

x=np.linspace(-2,2,numberSamples) # let x be in value from -2 to 2 with numberSamples between them

y=beta0+beta1*x+error
```

Now, we use y as the label and x is the feature!

```
y=y.reshape(-1,1); x=x.reshape(-1,1)
```

Linear Regression in Python

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

X_train, X_test, Y_train, Y_test= train_test_split(x, y, random_state= 0)

linreg= **LinearRegression().fit(X_train, Y_train)**

print("The intercept is: ", **linreg.intercept_**)

print("The coefficient of TV feature is:",**linreg.coef_)**

Plot the Data and the Fitted Model

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

estimated_linearmodel= linreg.intercept_ + linreg.coef_ * x

plt.figure(figsize=(5,4))
plt.scatter(x, y, marker= 'o') #plot data points
plt.plot(x, estimated_linearmodel, 'r-')

plt.xlabel('Feature value (x)'); plt.ylabel('Target value (y)')
plt.show()
```

Linear Regression in Python: Model accuracy

from **sklearn.metrics** import **mean_squared_error**

Target_predicted**= linreg.predict(X_test)**

MSE=mean_squared_error(Y_test,Target_predicted)

print('mean square error', MSE)

- **Exercise:** Try to run couple of times. Comment on the coefficient estimates and the MSE in each run

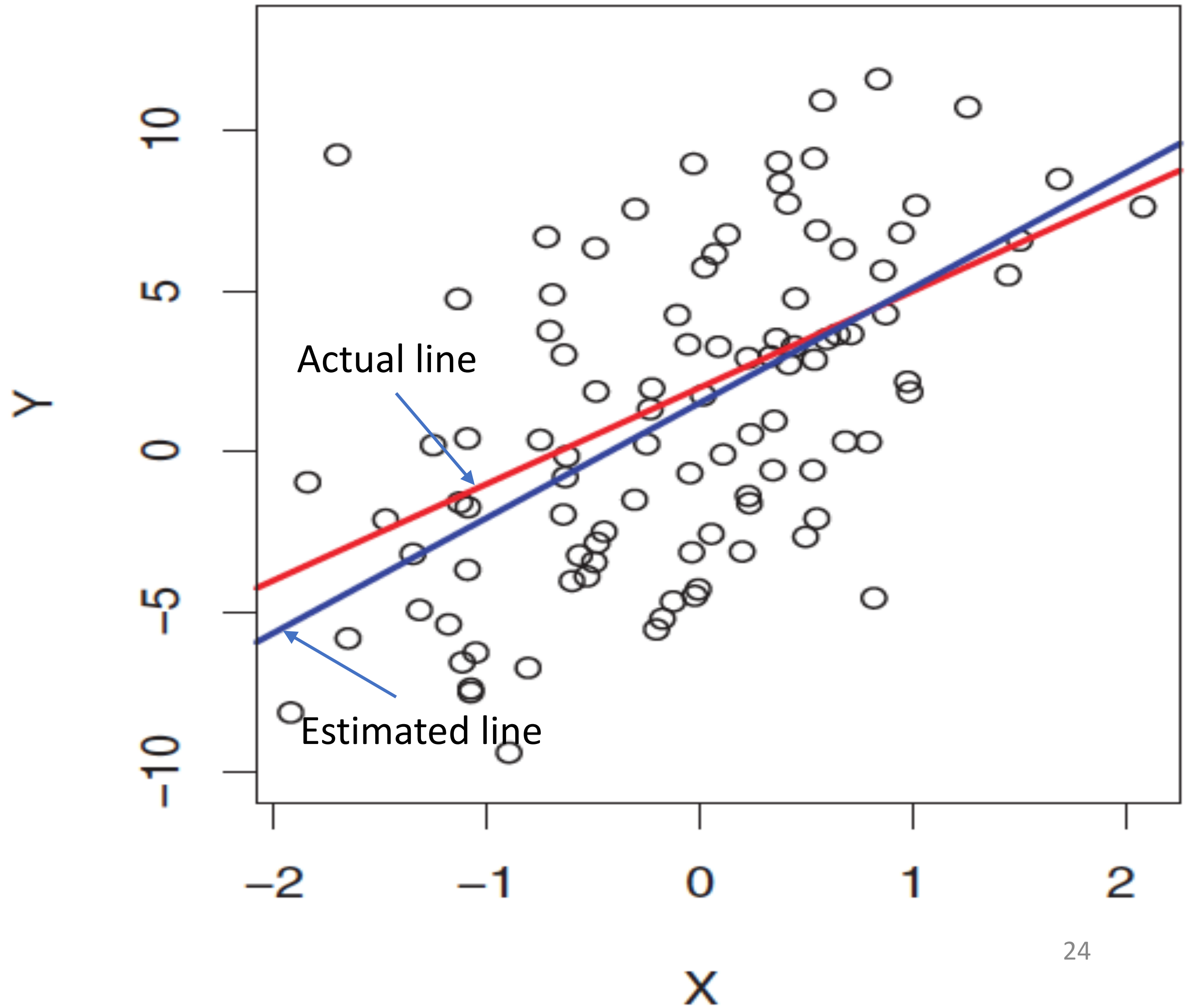
Accuracy of Coefficient Estimates

- Red line: actual f(x)=2+3X
  Population regression,

  $$y=2+3X+\epsilon$$

- Black circles are data points

- Blue line: least square line estimated from 100 training points

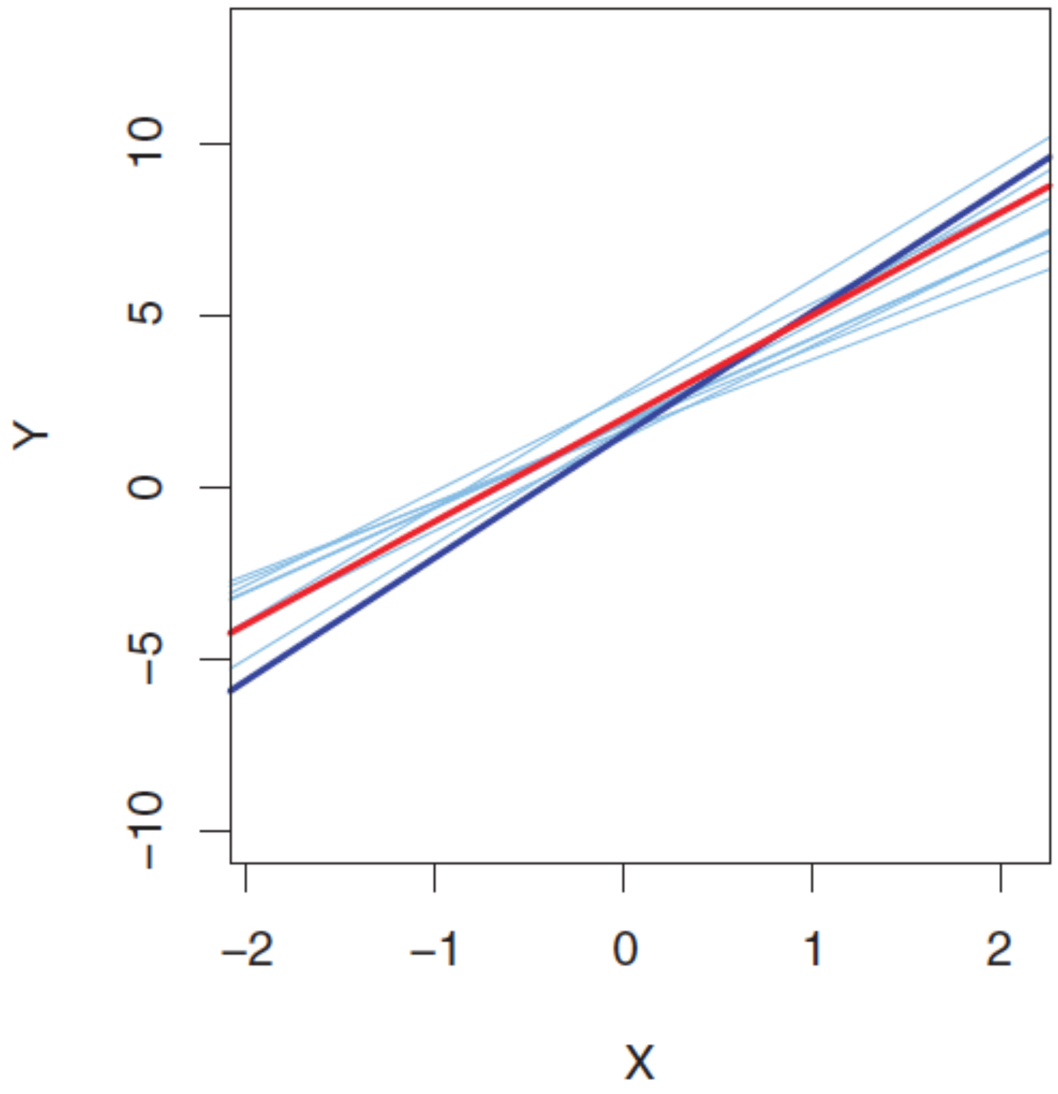


Accuracy of Coefficient Estimates

- Fig. shows 10 least square lines (light blue lines), each computed based on 10 random set of training observations

- The fitted model depends on the data

- How accurate is the estimated coefficients?



Accuracy of Coefficient Estimates - Variance

- We can measure the standard error (SE), which is also the variance, of each coefficient
  - Reflects how the coefficient varies under repeated sampling
  - Let $\sigma^2 = Var(\epsilon)$, and $\bar{x}$ be the average of feature *x*, *n* is the number of observations

$$\mathrm{SE}(\hat{\beta}_0)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \right]$$

$$\mathrm{SE}(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

  - Proof is beyond the scope

Accuracy of Coefficient Estimates - Confidence Interval

- ***Confidence interval:*** 95% confidence interval is the range of values that with 95% probability the range will contain the **true unknown value** of the coefficient.

- For β_1, this is approximately equal to:

$$\hat{\beta}_1 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_1) \Longrightarrow \left[\hat{\beta}_1 - 2 \cdot \mathrm{SE}(\hat{\beta}_1),\ \hat{\beta}_1 + 2 \cdot \mathrm{SE}(\hat{\beta}_1)\right]$$

Association Between Feature and Response

- Hypothesis test: procedure for deciding whether to accept or reject the assertion based on the observed data
  - A hypothesis is an assertion about the distribution of a random variable

- Hypothesis test:
  - **Null hypothesis H_0**: No association between feature and response, i.e.

$$\boldsymbol{\beta_1 = 0}$$

  - **Alternative hypothesis H_1**: There is an association between feature and response, i.e.

$$\boldsymbol{\beta_1 \neq 0} \Rightarrow \text{In this case } \hat{y} = \beta_0$$

Test the Null Hypothesis – T-statistics and P-value

- t-statistics: $t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)}$

  - Measures how far $\hat{\beta}_1$ is away from 0 (Null hypothesis).

  - If t-statistics is large, this reflects association between feature and response

Test the Null Hypothesis – T-statistics and P-value

- P-value: is the probability of observing a statistical value (here we use t-statistics) equal to the observed ($|t|$) or **larger if there is no association between response and the feature** (i.e when Null hypothesis is true)

  - Small value reflects that there is an association between features and output

  - Large value reflects low association between features and output

Association Between Feature and Response

- We care about the terms corresponding to the features (not the intercept)

	Coefficient	Std. Error	t-statistic	p-value
Intercept	7.0325	0.4578	15.36	< 0.0001
TV	0.0475	0.0027	17.67	< 0.0001

Calculate Statistical Summary in Python

- Get advertising dataset from courseweb or from this link: http://www-bcf.usc.edu/~gareth/ISL/data.html

from **pandas** import **read_csv**

AdvertisingData=read_csv('Advertising.csv’)

- Use statsmodels python module, to get **statistical summary** for ordinary least square: http://www.statsmodels.org/stable/index.html

import **statsmodels.formula.api** as **smf**

model=**smf.ols**('Sales ~ TV', AdvertisingData)

Fitting_results=**model.fit()**

print(Fitting_results.summary().tables[1])

print('p-values are: \n', **Fitting_results.pvalues**)

- The statistical summary you will get includes:

Coefficients, standard error or variance (std_err), t statistics, P-value (P>|t|), confidence interval ([0.025, 0.975])

- **Exercise**: what is the confidence interval of TV coefficient generated by the code?
    - Does the confidence interval include ‘zero’? What does that imply?

Linear Regression – Multiple Features

Linear Regression with Multiple Features

- Y= β_0+$\beta_1 X_1$+$\beta_2 X_2$+…+$\beta_p X_p$ $+ \epsilon$

β_j is the average effect on Y of one unit increase in X_j holding other features fixed

- *Challenge: In practice , features can be correlated and may change with each other (will be discussed)*

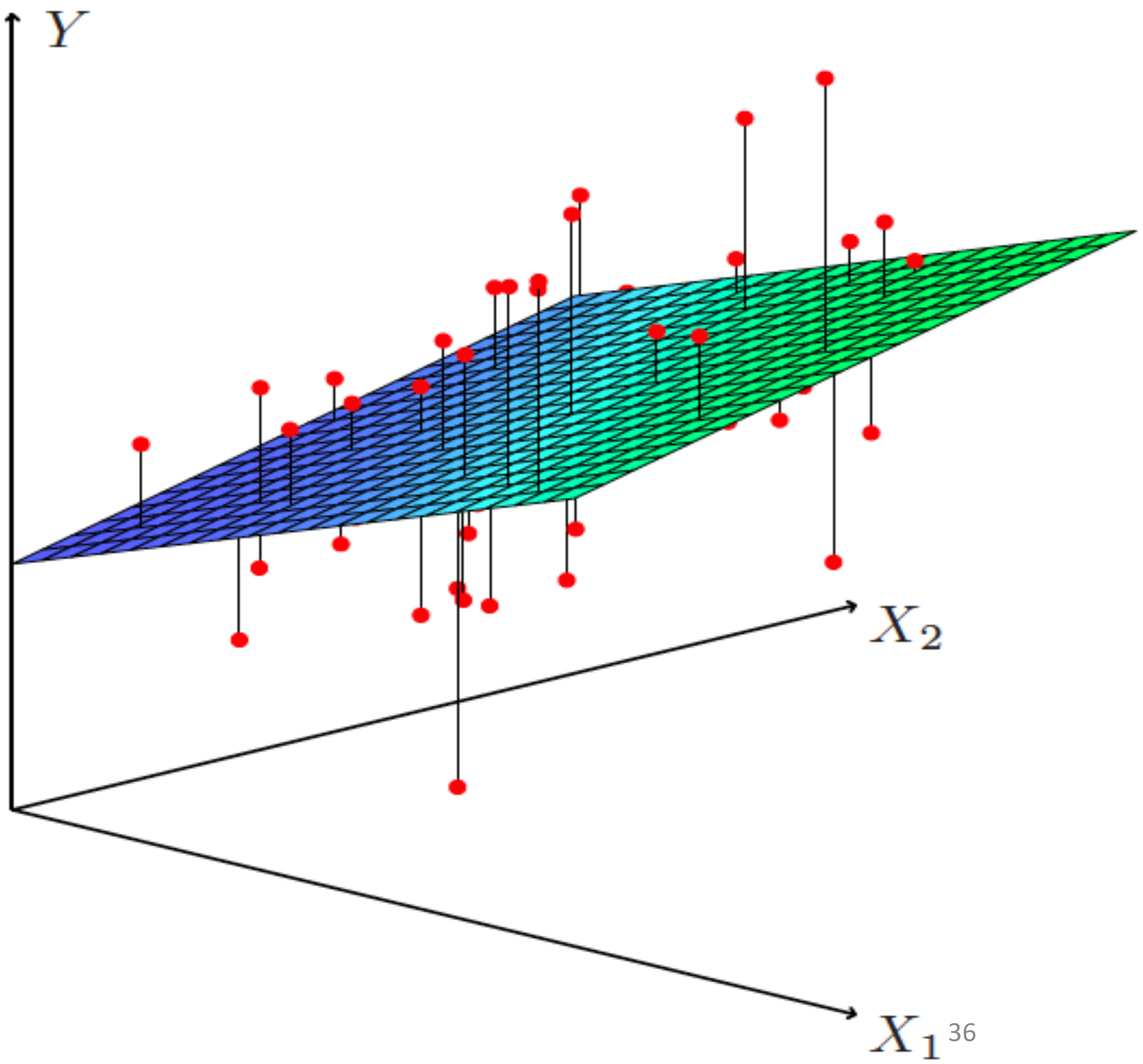
- Make predictions using
  $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \ldots + \hat{\beta}_p x_p$

- **Estimate coefficients** $(\beta_0, \beta_1, \beta_2 \ldots \beta_p)$ that minimizes the MSE or RSS

$$\text{RSS} = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

$$= \sum_{i=1}^{n} \left[\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \ldots + \hat{\beta}_p x_{ip} - y_i\right]^2$$

$x_{i,j}$: jth feature of ith trainning point



Multiple least squares regression estimate

Rule: number of observation points/training must be larger than number of features, i.e., P<n

Least Square Method: Matrix Form Closed-Form Solution

- Matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ . \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} \, .. & x_{1p} \\ 1 & x_{21} \, .. & x_{2p} \\ . & . & . \\ 1 & x_{n1} \, .. & x_{np} \end{bmatrix} \begin{bmatrix} \beta_o \\ \beta_1 \\ . \\ \beta_p \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ . \\ . \\ \epsilon_n \end{bmatrix}$$

$x_{i,j}$: jth feature of ith trainning point

- That is: y= $X\,\beta + \epsilon$
  - y is target of training data (n x 1) vector
  - X is n x (p+1) matrix , first column is all constant (1's)
  - $\hat{\beta}$ is (p+1)x1 vector of $[\beta_0 \,..\, \beta_p]^T$

Closed-form Solution

- RSS($\hat{\beta}$)=(y-X$\hat{\beta}$)^t (y-X$\hat{\beta}$)

- If we differentiate with respect to β, then equate to zero, we get:

$$\frac{\partial RSS}{\partial \hat{\beta}} = \frac{\partial}{\partial \hat{\beta}}(y^t y - \hat{\beta}^t X^t y - y^t \mathrm{X}\, \hat{\beta} + \hat{\beta}^t X^t\, \mathrm{X}\, \hat{\beta}\,) = 0$$

$Note$: $\hat{\beta}^t X^t y = y^t \mathrm{X}\, \hat{\beta} = scaler$ value

$$0 - 2X^t y + 2\, X^t\, \mathrm{X}\, \hat{\beta} = 0$$

$\frac{\partial}{\partial \hat{\beta}} \hat{\beta}^t X^t\, \mathrm{X}\, \hat{\beta} = 2X^t\, \mathrm{X}\, \hat{\beta}$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Optimization – Gradient Descent For Multiple Variables

- Same as before
- Objective is $\mathrm{J}(\beta) = J(\beta_1 \ldots \beta_p) = RSS = \sum_{i=1}^{n} \left[\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \ldots + \hat{\beta}_p x_{ip} - y_i\right]^2$
- Iterative method to find the optimal values of coefficients
  - Start with some values for the parameters $(\beta_0 \,..\, \beta_p)$
  - Calculate the gradient and then simultaneously update the parameters

$$\beta_i := \beta_i - \alpha \frac{\partial}{\partial \beta_i} \mathrm{J}(\beta),\ \alpha \text{ is the learning rate}$$

  - Keep changing the parameters to reduce the objective function until the minimum value of the objective function is obtained.

Features Association: Sales Example

$$\texttt{sales} = \beta_0 + \beta_1 \times \texttt{TV} + \beta_2 \times \texttt{radio} + \beta_3 \times \texttt{newspaper} + \epsilon$$

- Features with low p-values have association with output

	Coefficient	Std. Error	t-statistic	p-value
Intercept	2.939	0.3119	9.42	< 0.0001
TV	0.046	0.0014	32.81	< 0.0001
radio	0.189	0.0086	21.89	< 0.0001
newspaper	-0.001	0.0059	-0.18	0.8599

High p-value, low association with Sales

In python, use: model=smf.ols('Sales ~ TV+Radio+Newspaper', AdvertisingData)

- Find P-value corresponding to Newspaper budget when using **both *TV and Newspaper*** as features.
    - Do both features impact the sales?
        - Answer: Yes, both have low P-values

- Find P-value of Newspaper when using **TV, Radio and Newspaper** as features
    - Which is of least importance?
    - *Here, it is clear that **Newspaper has no impact** on Sales!*

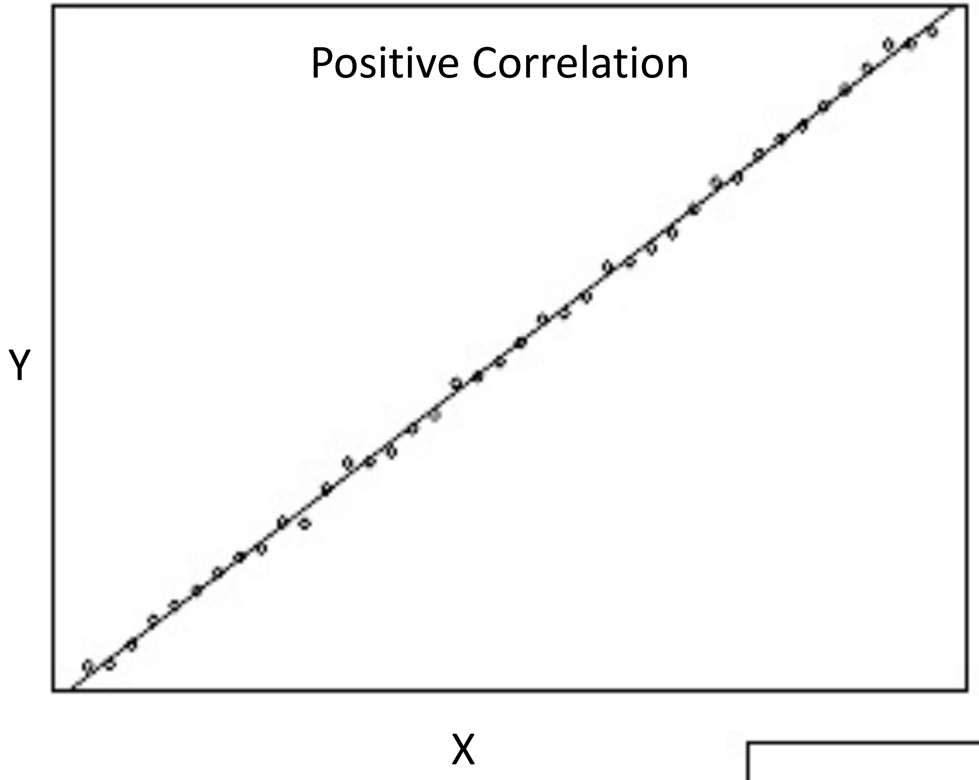
- Why did P-value change when adding Radio budget feature?
  - Due to correlation between Radio advertising and Newspaper advertising in markets where data was collected.
    - Tendency to spend more on newspaper advertising in markets where more is spent on radio advertising

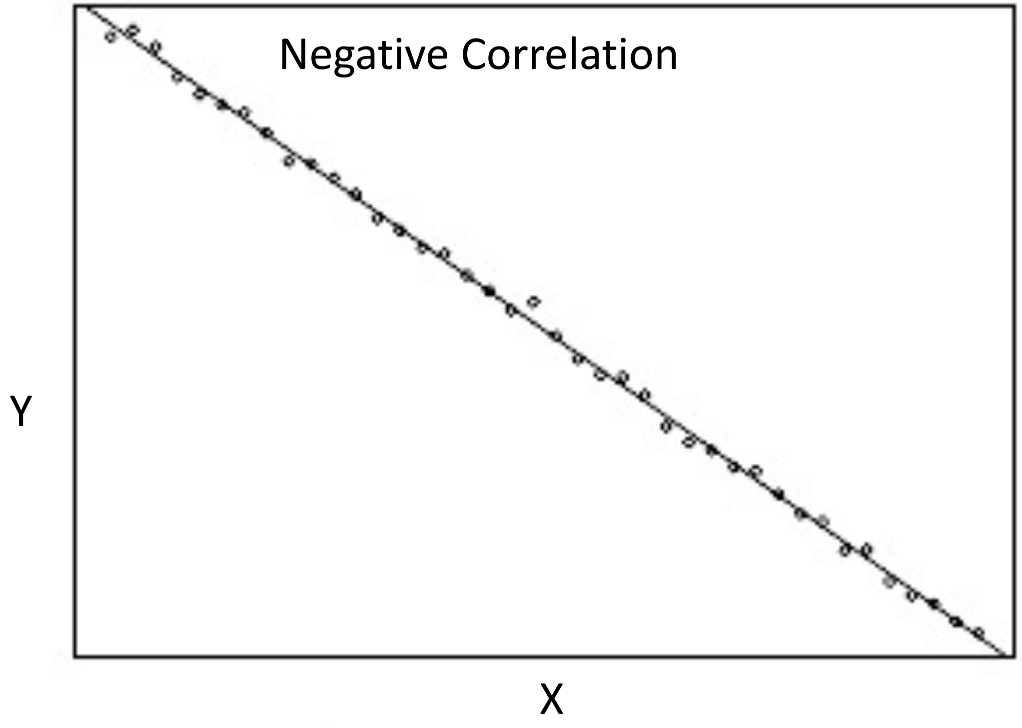
Correlation Matrix

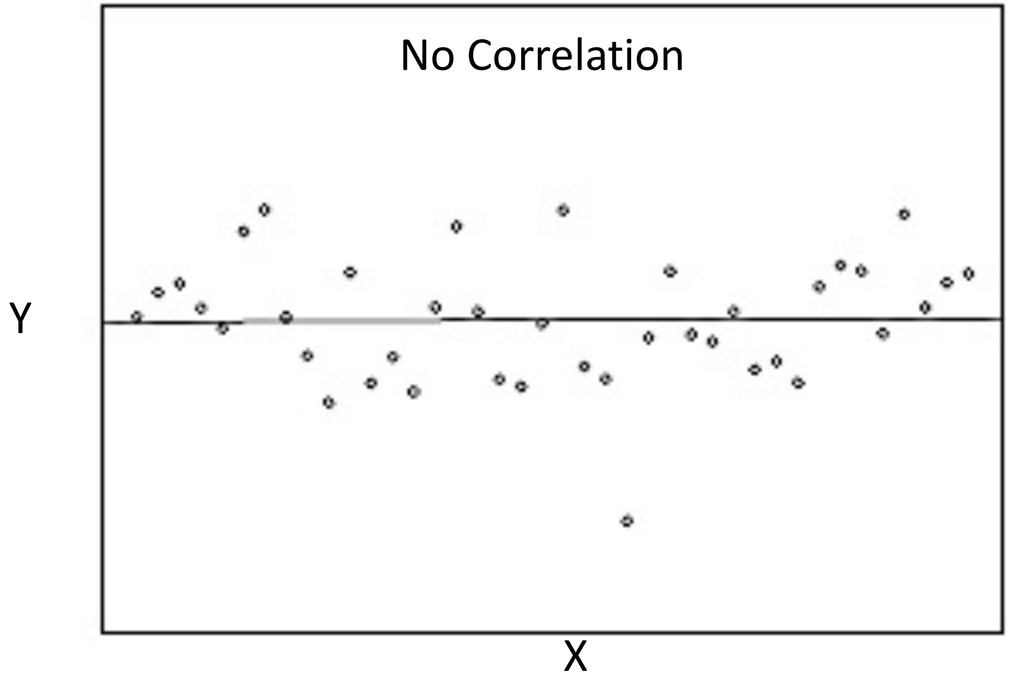
- Finding the correlation between features and target helps in inference
- Covariance between two variables X, Y: COV(X,Y) = E[(X-mean(X))(Y-mean(Y)]
- If covariance is zero, then X and Y are not correlated
- Correlation coefficient between X, Y ($\rho_{x,y}$) Is a value in the range of [-1,1]

$$\rho_{x,y} = \frac{COV(X,Y)}{\sqrt{VAR(X)}\sqrt{VAR(Y)}}$$

  - If $\rho_{x,y}$=0, then there is no correlation
  - If $\rho_{x,y}$=1, then there is a positive correlation
  - If $\rho_{x,y}$=-1, then there is a negative correlation


Positive Correlation
Y
X


Negative Correlation
Y
X


No Correlation
Y
X

- Correlation matrix contains correlation coefficient of different variables
- Can be obtained with numpy in python

Import numpy as np

correlation_coef2=np.corrcoef([AdvertisingData.TV, AdvertisingData.Radio, AdvertisingData.Newspaper, AdvertisingData.Sales])

	TV	Radio	Newspaper	Sales
TV	1.000000	0.054809	0.056648	0.782224
Radio	0.054809	1.000000	0.354104	0.576223
Newspaper	0.056648	0.354104	1.000000	0.228299
Sales	0.782224	0.576223	0.228299	1.000000

Each number presents correlation between the corresponding variables (indicated in row and column labels)

Calculate the MSE with and without Newspaper advertisement, you should find the difference is very low!

Features Association: Decide on Important Features

- Find least square fit for all possible subsets of the features, then find best one
- What if we have large number of features?
  - Very complex – 40 features, we have over a billion models
- Need more efficient approach to find the best subset of features

Features Association: Decide on Important Features

- Forward selection:
  - Start with null hypothesis
  - Fit p simple linear regression models, then add to the null model the feature that results in lowest RSS
  - Add to that model the feature that results in lowest RSS among all two-feature models
  - And so on until stopping criteria is met

- Backward selection:
  - Start with all variables/features in the model
  - Remove variable with largest p-value
  - The new (p-1 features) model is fitted, and feature with largest p-value is removed
  - Repeat until stopping criteria is met: all variables have p-value below some threshold

- Mixed selection: combination of forward and backward selection
  - Starts with null hypothesis,
  - Add one feature at a time.
  - If a feature has its p-value higher than a certain threshold when new features are added to the model, then this feature is removed.
  - Unlike forward and backward selection, here it is possible to remove feature after adding it

- Feature transformation: discussed later

Scaling and Normalization

- In some cases, feature scaling and normalization may improve the performance of a learning algorithm

  - House price prediction: area (2000 $feet^2$), number of bedrooms [1-5]

- Essential when using gradient descent
  - The derivative will lead to multiplying be a feature value. This will make the rate of parameter update depend on the value of the feature

- Python functions: MinMaxScaler or StandardScaler

Assessing Model Accuracy

- How well the model fits the data?
- Metrics:
  - Mean squared error : MSE= $\frac{1}{n}\sum_{i=1}^{n}[\hat{y}_i - y_i]^{\,2}$ =$\frac{1}{n}\,RSS$
  - Root mean squared error: RMSE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}[\hat{y}_i - y_i]^{\,2}} = \sqrt{\frac{1}{n}\ RSS}$
  - R^2 metric

Assessing Model Accuracy

- R^2 metric: is a number between [0,1]

$$R^2 = \frac{\mathrm{TSS} - \mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}}$$

  - TSS is the total sum of squares $\mathrm{TSS} = \sum (y_i - \bar{y})^2$
    - Measures total variance in the response Y (variability in Y) before the regression is performed
  - R^2 measures the proportion of variability in Y that can be explained using feature (X) .